

Міністерство освіти і науки України
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
з дисципліни
“Конструювання програмного забезпечення”
для студентів
спеціальностей
121 Інженерія програмного забезпечення,
122 Комп’ютерні науки та інформаційні технології
(всіх форм навчання)

2016

Методичні вказівки до виконання лабораторних робіт з дисципліни “Конструювання програмного забезпечення” для студентів спеціальностей 121 Інженерія програмного забезпечення, 122 Комп’ютерні науки та інформаційні технології / А. О. Олійник, Є. М. Федорченко. – Запоріжжя: ЗНТУ, 2016. – 104 с.

Автори: А. О. Олійник, к.т.н., доцент
Є. М. Федорченко, ст. викладач

Рецензент: С. О. Субботін, д.т.н., професор

Відповідальний
за випуск: С. О. Субботін, д.т.н., професор

Затверджено
на засіданні кафедри
програмних засобів

Протокол № 1
від “16” серпня 2016 р.

ЗМІСТ

Вступ	6
1 Лабораторна робота № 1 Знайомство з Java.....	7
1.1 Мета роботи	7
1.2 Основні теоретичні відомості.....	7
1.2.1 Константи Цілі Дійсні Символи Рядки.....	9
1.2.2 Імена.....	10
1.2.3 Примітивні типи даних і операції.....	10
1.2.4 Дійсні типи.....	13
1.2.5 Операції присвоювання.....	13
1.2.6 Оператори.....	13
1.2.7 Масиви.....	20
1.3 Завдання до роботи.....	21
1.4 Зміст звіту.....	22
1.5 Контрольні запитання	22
2 Лабораторна робота № 2 Створення графічного інтерфейсу	24
2.1 Мета роботи	24
2.2 Основні теоретичні відомості.....	24
2.3 Завдання до роботи.....	31
2.4 Зміст звіту.....	32
2.5 Контрольні запитання	32
3 Лабораторна робота № 3 Об'єктно-орієнтоване програмування в Java.....	33
3.1 Мета роботи	33
3.2 Основні теоретичні відомості.....	33
3.2.1 Як описати клас і підклас.....	33
3.2.2 Абстрактні методи й класи	37
3.2.3 Остаточні члени й класи	38
3.2.4 Клас Object	38
3.2.5 Конструктори класу.....	39
3.2.6 Операція new	40
3.2.7 Статичні члени класу	40
3.2.8 Клас Complex	43
3.3 Завдання до роботи.....	45

3.4 Зміст звіту.....	47
3.5 Контрольні запитання	47
4 Лабораторна робота № 4 Пакети й інтерфейси	48
4.1 Мета роботи	48
4.2 Основні теоретичні відомості.....	48
4.2.1 Права доступу до членів класу.....	49
4.2.2 Імпорт класів і пакетів.....	50
4.2.3 Інтерфейси.....	50
4.2.4 Design patterns	52
4.3 Завдання до роботи.....	54
4.4 Зміст звіту.....	55
4.5 Контрольні запитання	56
5 Лабораторна робота № 5 Класи-оболонки.....	57
5.1 Мета роботи	57
5.2 Основні теоретичні відомості.....	57
5.2.1 Клас Boolean Клас Character	58
5.2.2 Клас BigInteger	61
5.2.3 Клас BigDecimal.....	64
5.3 Завдання до роботи.....	67
5.4 Зміст звіту.....	68
5.5 Контрольні запитання	68
6 Лабораторна робота № 6 Робота з рядками.....	69
6.1 Мета роботи	69
6.2 Основні теоретичні відомості.....	69
6.3 Завдання до роботи.....	77
6.4 Зміст звіту.....	78
6.5 Контрольні запитання	79
7 Лабораторна робота № 7 Класи-колекції.....	80
7.1 Мета роботи	80
7.2 Основні теоретичні відомості.....	80
7.2.1 Клас Vector	80
7.2.2 Клас Stack Клас Hashtable Клас Properties.....	82
7.2.3 Інтерфейс Collection	86
7.2.4 Інтерфейс Iterator	87
7.2.5 Колекції	88
7.3 Завдання до роботи.....	90

7.4 Зміст звіту.....	93
7.5 Контрольні запитання	94
8 Лабораторна робота № 8 Класи-утиліти	95
8.1 Мета роботи	95
8.2 Основні теоретичні відомості.....	95
8.2.1 Робота з масивами	95
8.2.2 Локальні установки	96
8.2.3 Робота з датами й часом.....	97
8.2.4 Одержання випадкових чисел	100
8.2.5 Взаємодія із системою.....	101
8.3 Завдання до роботи.....	102
8.4 Зміст звіту.....	103
8.5 Контрольні запитання	103
Література.....	104

ВСТУП

Дане видання призначене для вивчення та практичного освоєння студентами усіх форм навчання основ мови Java

Відповідно до графіка студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою. Звичайно, у дані методичні вказівки неможливо було внести весь матеріал, необхідний для виконання та захисту лабораторних робіт. Тому тут містяться основні, базові теоретичні відомості, необхідні для виконання лабораторних робіт. Таким чином для виконання лабораторної роботи та при підготовці до її захисту необхідно ознайомитись з конспектом лекцій та проробити весь матеріал, наведень у переліку рекомендованої літературі. При цьому не варто обмежуватись лише наведеним списком.

Для одержання заліку з кожної роботи студент здає викладачу цілком оформлень звіт, а також демонструє на екрані комп'ютера результати виконання лабораторної роботи.

Звіт має містити:

- титульний аркуш(на ньому вказують назву міністерства, назву університету, назву кафедри, номер, вид і тему роботи, виконавця та особу, що приймає звіт, рік);
- тему та мету роботи;
- завдання до роботи;
- лаконічний опис теоретичних відомостей;
- результати виконання лабораторної роботи;
- змістовний аналіз отриманих результатів та висновки.

Звіт виконують на білому папері формату A4(210 × 297 мм). Текст розміщують тільки з однієї сторони аркуша. Поля сторінки з усіх боків - 20 мм. Аркуші скріплюють за допомогою канцелярських скріпок або вміщують у канцелярський файл.

Під година співбесіди при захисті лабораторної роботи студент повинний виявити знання про позначку роботи, по теоретичному матеріалу, про методи виконання шкірного етапу роботи, по змісту основних розділів оформленого звіту з демонстрацією результатів на конкретних прикладах. Студент повинний вміти правильно аналізувати отримані результати. Для самоперевірки при підготовці до виконання й захисту роботи студент винний відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

1 ЛАБОРАТОРНА РОБОТА № 1

ЗНАЙОМСТВО З JAVA

1.1 Мета роботи

Вивчити основні можливості та принципи роботи з мовою Java.

1.2 Основні теоретичні відомості

Почнемо, за традицією, з найпростішої програми.

У програмі 1.1 в найпростішому виді, записана мовою Java.

Програма 1.1. Перша програма мовою Java;

```
class HelloWorld{  
    public static void main(String[] args){  
        System.out.println("Hello, XXI Century World!");  
    }  
}
```

Навіть на цьому простому прикладі можна помітити цілий ряд істотних особливостей мови Java.

Будь-яка програма являє собою один або кілька класів, у цьому найпростішому прикладі тільки один клас(class).

Початок класу визначається службовим словом class, за яким повинно бути ім'я класу, у цьому випадку HelloWorld. Усе, що втримується в класі, записується у фігурних дужках і становить тіло класу(class body).

Всі дії проводяться за допомогою методів обробки інформації. Ця назва вживається в мові Java замість назви "функція", застосовуваного в інших мовах. Методи розрізняються по іменах. Один з методів обов'язково повинен називатися main, з нього починається виконання програми.

Перед типом повертаємого методом значення можуть бути записані модифікатори(modifiers). У прикладі їх два: слова public означає, що цей метод скрізь доступний; слово static забезпечує можливість виклику методу main() на самому початку виконання програми. Модифікатори взагалі необов'язкові, але для методу main() вони необхідні.

Мова Java розрізняє прописні й малі літери.

Отже, програма пишеться в будь-якому текстовому редакторі. Тепер її треба зберегти у файлі, ім'я якого збігається з ім'ям класу, що містить метод main() дотримуючись регістру букв, і дати імені файлу

розширення Java. При цьому система виконання Java буде швидко знаходити метод `main()` для початку роботи, просто відшуковуючи клас, що збігається з ім'ям файлу.

У нашому прикладі, збережемо програму у файлі з ім'ям `HelloWorld.java` у поточному каталозі. Потім запускаємо компілятор, передаючи йому ім'я файлу як аргумент:

```
javac HelloWorld.java
```

Компілятор створить файл з байт-кодами, дасть йому ім'я `HelloWorld.class` і запише цей файл у поточний каталог.

Залишилося викликати інтерпретатор, передавши йому як аргумент ім'я класу (а не файлу):

```
Java HelloWorld
```

На екрані з'явиться:

```
Hello, 21st Century World!
```

Не вказуйте розширення `class` при виклику інтерпретатора.

На рис. 1.1 показано, як все це виглядає у вікні Command Prompt операційної системи MS Windows 2000.

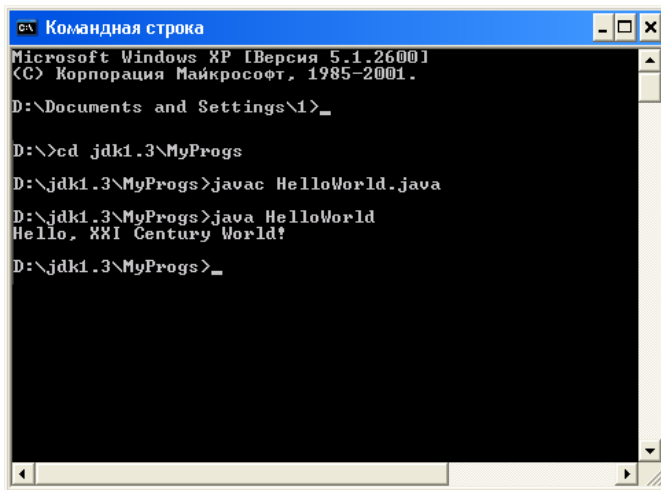


Рисунок 1.1 – Вікно Command Prompt

При роботі в інтегрованому середовищі всі ці дії викликаються вибором відповідних пунктів меню або "гарячими" клавішами - єдиних правил тут немає.

1.2.1 Константи Цілі Дійсні Символи Рядки

У мові Java можна записувати константи різних типів у різних видах.

Цілі

Цілі константи можна записувати в декількох системах числення:

– у десятковій формі: +5, -7, 12345678;

– у восьмеричній формі, починаючи з нуля: 027, -0326, 0777 ; у записі таких констант неприпустимі цифри 8 і 9;

Наприкінці цілої константи можна записати букву прописну L або рядкову l, тоді константа буде зберігатися в довгому форматі типу long(див. нижче): +25L, -0371, 0xff, OXDFDF1.

Наприкінці дійсної константи можна поставити букву F або f, тоді константа буде зберігатися у форматі типу float(див. нижче): 3.5f, -45.67F, 4.7e-5f. Можна приписати й букву D(або d): 0.045D, -456.77889d, що означає тип double, але це зайве, оскільки дійсні константи й так зберігаються у форматі типу double.

Символи

Для запису одиночних символів використовуються наступні форми.

Друковані символи можна записати в апострофах: 'a', 'N', '? '.

Керуючі символи записуються в апострофах зі зворотною похилою рисою:

'\n' - символ нової строки newline з кодом ASCII 10;

'\r' - символ повернення каретки CR з кодом 13;

'\f' - символ нової сторінки FF з кодом 12;

'\b' - символ повернення на крок BS з кодом 8;

'\t' - символ горизонтальної табуляції HT із кодом 9;

'\\' - зворотна похила риса;

'\"' - лапки;

'\'' - апостроф.

Символи зберігаються у форматі типу char(див. нижче).

Прописні російські букви в кодуванні Unicode займають діапазон від '\u0410' - заголовна літера А, до '\u042F' - заголовна Я, малі літери від '\u0430' - а, до '\u044F' - я.

Рядки

Рядки символів треба писати у лапках. Керуючі символи й коди записуються в рядках точно так само, зі зворотною похилою рисою,

але, зрозуміло, без апострофів, і роблять ту ж дію. Рядки можуть розташовуватися тільки на одному рядку вихідного коду, тобто не можна відкриваючі лапки поставити на одному рядку, а закриваючі - на наступній.

От деякі приклади:

"Це рядок\пс перенесенням"

"\\"Спартак\\" - Чемпіон!"

Рядки символів не можна починати на одному рядку вихідного коду, а закінчувати на іншій.

Для строкових констант визначена операція зчеплень, позначувана плюсом.

" Зчеплення " + "рядків" дає в результаті рядок "Зчеплення рядків".

1.2.2 Імена

Імена(names) змінних, класів, методів і інших об'єктів можуть бути простими (загальна назва – ідентифікатори(idenifiers)) і складовими (qualified names). Ідентифікатори в Java складаються з так званих букв Java (Java letters) і арабських цифр 0-9, причому першим символом ідентифікатора не може бути цифра. У число букв Java обов'язково входять прописні й рядкові латинські букви, знак долара \$ і знак підкреслення _, а так само символи національних алфавітів.

1.2.3 Примітивні типи даних і операції

Всі типи вихідних даних, вбудовані в мову Java, діляться на дві групи: примітивні типи(primitive types) і посилальні типи(reference types).

Посилальні типи діляться на масиви(arrays), маси(classes) і інтерфейси(interfaces).

Примітивних типів усього вісім. Їх можна розділити на логічний(іноді говорять булев) тип boolean і числові(numeric).

До числових типів відносяться цілі й дійсні типи.

Цілих типів п'ять: byte, short, int, long, char.

Дійсних типів два: float і double.

На рис. 1.2 показана ієрархія типів даних Java.

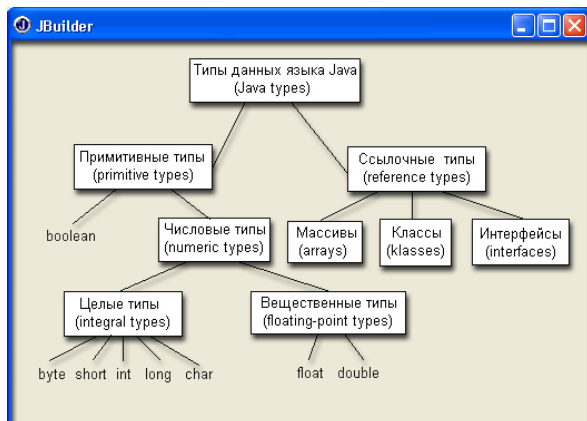


Рисунок 1.2 – Типи даних мови Java

Логічний тип

Значення логічного типу `boolean` виникають у результаті різних порівнянь, наприклад $2 > 3$, і використовуються, головним чином, в умовних операторах і операторах циклів. Логічних значень всього два: `true`(істина) і `false`(неправда). Це службові слова Java. Опис змінних цього типу виглядає так:

```
boolean b = true, bb = false, bool2;
```

Над логічними даними можна виконувати операції присвоювання, наприклад, `bool2 = true`, у тому числі й складові з логічними операціями; порівняння на рівність `b == bb` і на нерівність `b != bb`, а також логічні операції.

Логічні операції:

- заперечення(NOT) `!`(позначається знак оклику);
- кон'юнкція (AND) `&`(амперсанд);
- диз'юнкція(OR) `|`(вертикальна риса);
- що виключає АБО(XOR) `^`(каре).

Вони виконуються над логічними даними, їхнім результатом буде теж логічне значення `true` або `false`. Про них можна нічого не знати, крім того, що представлено в табл. 1.1.

Таблиця 1.1 – Логічні операції

b1	b2	!b1	b1&b2	b1 b2	b1^b2
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Крім описаних чотирьох логічних операцій є ще дві логічні операції скороченого обчислення:

- скорочена кон'юнкція (conditional-AND) && ;
- скорочена диз'юнкція (conditional-OR) ||.

Подвоєні знаки амперсанда й вертикальної риси варто записувати без пробілів.

Правий операнд скорочених операцій обчислюється тільки в тому випадку, якщо від нього залежить результат операції, тобто якщо лівий операнд кон'юнкції має значення true, або лівий операнд диз'юнкції має значення false.

Це правило дуже зручно використовувати, наприклад, можна записати вираження $(n \neq 0) \&\&(m/n > 0.001)$ або $(n == 0) \|(m/n > 0.001)$ не побоюючись ділення на нуль.

Цілі типи

Специфікація мови Java, JLS, визначає розрядність (кількість байтів, призначених для зберігання значень типу в оперативній пам'яті) і діапазон значень кожного типу. Для цілих типів вони наведені в табл. 1.2.

Таблиця 1.2 – Цілі типи

Тип	Розрядність (байт)	Діапазон
byte	1	від -128 до 127
short	2	від -32768 до 32767
int	4	від -2147483648 до 2147483647
long	8	від -9223372036854775808 до 9223372036854775807
char	2	від '\u0000' до '\uFFFF', у десятковій формі від 0 до 65535

Хоча тип char займає два байти, в арифметичних обчисленнях він бере участь як тип int, йому виділяється 4 байти, два старших байти заповнюються нулями.

Цілі типи зберігаються у двійковому виді з додатковим кодом. Останнє означає, що для негативних чисел зберігається не їхнє двійкове подання, а додатковий код цього двійкового подання.

Операції над цілими типами

Всі операції, які проводяться над цілими числами, можна розділити на наступні групи.

Побітові операції

Іноді доводиться змінювати значення окремих бітів у цілих даних. Це виконується за допомогою побітових(bitwise) операцій шляхом накладення маски. У мові Java є чотири побітові операції:

- доповнення(complement) $\sim$ (тильда);
- побітова кон'юнкція (bitwise AND) $\&$;
- побітова диз'юнкція(bitwise OR) $|$;
- побітове що виключає АБО(bitwise XOR) $\wedge$.

Зсуви

У мові Java є три операції зсування двійкових розрядів:

- зсув вліво $\ll$;
- зсув вправо $\gg$;
- беззнакове зсування вправо $\ggg$.

Ці операції своєрідні тим, що лівий і правий операнди в них мають різний сенс. Ліворуч стоїть значення цілого типу, а права частина показує, на скільки двійкових розрядів зрушується значення, що коштує в лівій частині.

1.2.4 Дійсні типи

Дійсних типів в Java два: float і double.

Приклади визначення речових типів:

```
float x = 0.001, y = -34.789;
double z1 = -16.2305, z2;
```

1.2.5 Операції присвоювання

Проста операція присвоювання (simple assignment operator) записується знаком рівності $=$, ліворуч від якого стоїть змінна, а праворуч вираження, сумісне з типом змінної:

```
x = 3.5, y = 2 * (x - 0.567) / (x + 2), b = x < y, bb = x >= y && b.
```

Крім простої операції присвоювання є ще 11 складних операцій присвоювання (compound assignment operators):

```
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>= ; >>>=.
```

1.2.6 Оператори

Набір операторів мови Java включає:

- оператори опису змінних і інших об'єктів;

- оператори-вираження;
 - оператори присвоювання;
 - умовний оператор if;
 - три оператори циклу while, do-while, for;
 - оператор варіанта switch;
 - оператори переходу break, continue і return;
 - блок {};
 - порожній оператор - просто крапка з комою.
- Усякий оператор завершується крапкою з комою.

True та False

Всі порівняльні вирази використовують правдивий або помилковий вираз порівняння для визначення шляху виконання. Прикладом порівняльного вираження є $A == B$. Тут використовується порівняльний оператор «==», щоб побачити, чи дорівнює значення A значенню B . Вираз повертає true або false. Java не допускає використання чисел, як значення типу boolean, незважаючи на те, що це припустимо в C й в C++ (де щирим є ненульове значення, а помилковим - нульове). Якщо ви хочете використовувати не boolean у булевських перевірках, таких як if(), ви повинні спочатку перетворити його в значення типу boolean, використовуючи вирази порівняння, такі як `if(a != 0)`.

If-else

Вираз if-else, імовірно, є основним способом керування ходом виконання програми. Вираз else є необов'язковим, тому можна використовувати if у двох формах:

```
if (Логічне вираження)
    інструкція
або
if (Логічне вираження)
    інструкція
else
    інструкція
```

Порівняння повинно давати результат типу boolean. Під інструкцією розуміється або проста інструкція, що завершується крапкою з комою, або складова інструкція, що групує прості інструкції, обрамлені фігурними дужками.

Як приклад if-else, тут наведений метод `test()`, що говорить вам чи є тестове значення більше, менше або рівним контрольному значенню:

```
public class IfElse {
    static int test(int testval, int target) {
```

```

int result = 0;
if(testval > target)
    result = +1;
else if(testval < target)
    result = -1;
else
    result = 0; // Збігається
return result;
}
public static void main(String[] args) {
    System.out.println(test(10, 5));
    System.out.println(test(5, 10));
    System.out.println(test(5, 5));
}
}

```

Оператор continue і мітки

Оператор continue використовується тільки в операторах циклу. Він має дві форми. Перша форма складається тільки зі слова continue і здійснює негайний перехід до наступної ітерації циклу. У черговому фрагменті коду оператор continue дозволяє обійти ділення на нуль:

```

for(int i = 0; i < N; i++){
    if(i == j) continue;
    s += 1.0 / (i - j); }

```

Друга форма містить мітку:

```
continue мітка
```

Мітка записується, як всі ідентифікатори, з букв Java, цифр і знака підкреслення, але не вимагає ніякого опису. Мітка ставиться перед оператором або відкриваючою фігурною дужкою й відділяється від них двокрапкою. Так виходить позначений оператор або позначений блок.

Друга форма використовується тільки у випадку декількох вкладених циклів для негайного переходу до чергової ітерації циклу.

Оператор break мітка

застосовується усередині позначених операторів циклу, оператора варіанту або позначеного блоку для негайного виходу за ці оператори. Наступна схема пояснює цю конструкцію.

```

M1: { // Зовнішній блок
M2: { // Вкладений блок - другий рівень
M3: { // Третій рівень вкладеності...
    if(щось трапилося) break M2;
    // Якщо true, те тут нічого не виконується
}
// Тут теж нічого не виконується
}
// Сюди передається керування
}

```

Break та Continue

Усередині тіла будь-якої інструкції ітерацій ви також можете використовувати керування роботою циклу, використовуючи break й continue. Break перериває цикл без виконання інструкцій, що залишилися, у циклі. Continue зупиняє виконання поточної ітерації й повертається до початку циклу, починаючи наступну ітерацію.

Ця програма показує приклад для break й continue усередині циклів for й while:

```
// Демонструє break й continue.
public class BreakAndContinue {
    public static void main(String[] args) {
        for(int i = 0; i < 100; i++) {
            if(i == 74) break; // вихід із циклу for
            if(i % 9 != 0) continue; // Наступна ітерація
            System.out.println(i);
        }
        int i = 0;
        // "Нескінченний цикл":
        while(true) {
            i++;
            int j = i * 27;
            if(j == 1269) break; // Вихід із циклу
            if(i % 10 != 0) continue; // У початок циклу
            System.out.println(i);
        } } }
```

У циклі for значення *i* ніколи не дійде до 100, тому що інструкція break перерве виконання циклу, коли *i* буде дорівнювати 74. Інструкція continue спричиняє повернення до початку циклу (при цьому інкриментуючи *i*) у кожному разі, коли *i* не ділиться на 9 без залишку. Якщо це так, значення друкується.

Другий розділ показує “нескінченний цикл”, що, теоретично, ніколи не закінчиться. Однак усередині циклу є інструкція break, що обірве цикл. Додатково, ви побачите, що continue повертає назад до початку циклу, не завершивши що залишилися. (Таким чином, друк відбувається в іншому циклі тільки тоді, коли значення *i* ділиться на 10.) Отримали такі результати:

```
0
9
18
27
36
45
54
63
72
10
```

20
30
40

Значення 0 друкується, тому що $0 \% 9$ дорівнює 0.

Друга форма нескінченного циклу: `for(;;)`. Компілятор трактує й `while(true)` і `for(;;)` однаково, що б ви не використовували – це питання стилю програмування.

Return

Ключове слово `return` має два призначення: воно вказує, яке значення повертає метод (якщо він не має повертає значення, яке типу `void`) і є причиною того, що значення повертається негайно. Метод `test()`, наведений вище, може бути переписаний з використанням цих переваг:

```
public class IfElse2 {
    static int test(int testval, int target) {
        int result = 0;
        if(testval > target)
            return +1;
        else if(testval < target)
            return -1;
        else
            return 0; // Збігається
    }
    public static void main(String[] args) {
        System.out.println(test(10, 5));
        System.out.println(test(5, 10));
        System.out.println(test(5, 5));
    }
}
```

Тут немає необхідності в `else`, тому що метод не буде виконуватися після виконання `return`.

Оператор варіанту

Оператор варіанта `switch` організує розгалуження по декількох напрямках. Кожна галузь відзначається константою або константним вираженням якого-небудь цілого типу (крім `long`) і вибирається, якщо значення певного вираження збігається із цією константою. Вся конструкція виглядає так.

```
switch(A) {
    case конствир1: оператор1
    case конствир2: оператор2
    case конствир: оператор
    default: операторDef
}
```

Вираз у дужках `A` може бути типу `byte`, `short`, `int`, `char`, але не `long`. Цілі числа або цілочисельні вираження, складені з констант, констивир теж не повинні мати тип `long`.

Константи у варіантах `case` відіграють роль тільки міток, точок входу в оператор варіанту, а далі виконуються всі оператори, що залишилися, у порядку їхнього запису.

Після виконання одного варіанту оператор `switch` продовжує виконувати всі варіанти, що залишилися.

Інструкція `switch` — це ясний спосіб для реалізації множинного вибору (тобто, вибору з великої кількості різних шляхів виконання), але це вимагає перемикача, при обчисленні якого виходить ціле значення типу `int` або `char`. Якщо ви хочете використовувати, наприклад, рядок або число із плаваючою крапкою як перемикач, вони не будуть працювати в інструкції `switch`. Для нецілих типів необхідні використовувати серію інструкцій `if`.

While

Форма циклу `while` наступна:

`while` (Логічне вираження)
інструкція

Логічний вираз обчислюється один раз на початку циклу, а потім щораз перед кожною майбутньою ітерацією для інструкції.

Тут наведено приклад, що генерує випадкові числа, поки не досягнутий певний стан:

```
// Демонстрація циклу while.
public class WhileTest {
    public static void main(String[] args) {
        double r = 0;
        while(r < 0.99d) {
            r = Math.random();
            System.out.println(r);
        }
    }
}
```

Тут використається статичний метод `random()` з бібліотеки `Math`, що генерує значення типу `double` у межах від 0 до 1 (0 включно, 1 невиключно). Порівняльний вираз для `while` говорить, “продовжувати вираз цього циклу, поки не зустрінеться число 0.99 або більше”. Щораз, коли ви запускаєте програму, ви будете одержувати список чисел різної довжини.

Do-while

Форма для do-while наступна:

```
do
    інструкція
while (логічне вираження);
```

Головна відмінність між while й do-while у тому, що інструкція в циклі do-while завжди виконується не менше одного разу, навіть якщо обчислений вираз є помилковим із самого початку. У циклі while, якщо умова є помилковою в перший раз, інструкція ніколи не виконається. На практиці do-while використовується рідше, ніж while.

For

Цикл for виконує ініціалізацію перед першою ітерацією. Потім він виконує порівняння та деякі дії, що відповідають одному кроку оператору for. Форма циклу for наступна:

```
for (ініціалізація; логічний вираз; крок)
    інструкція
```

Кожний з виразів: ініціалізація, логічний вираз або крок, може бути порожнім. Вираз перевіряється перед кожною ітерацією, і як тільки при обчисленні вийде false, виконання продовжитись з рядка, що іде за інструкцією for. Наприкінці кожного циклу виконується крок.

Цикл for звичайно використовується для завдань “обчислення”:

```
// Демонстрація циклу "for" для складання
// списку всіх ASCII символів.
public class ListCharacters {
    public static void main(String[] args) {
        for (char c = 0; c < 128; c++)
            if (c != 26) // ANSI Очищення екрана
                System.out.println(
                    "value: " + (int)c +
                    " character: " + c);
    }
}
```

Ви можете визначити декілька змінних усередині інструкції for, але вони повинні бути одного типу:

```
for(int i = 0, j = 1;
    i < 10 && j != 11;
    i++, j++)
    /* тіло циклу for */;
```

Визначення int в інструкції for розповсюджується на i та j. Здатність визначати змінні в керуючому виразі є обмеженням для циклу for. Ви не можете використовувати цей метод ні з яким іншим виразом вибору або ітераціями.

1.2.7 Масиви

Масиви

Як завжди в програмуванні масив – це сукупність змінних одного типу, що зберігаються в суміжних адресах оперативної пам'яті. Масиви в мові Java відносяться до посилальних типів. Опис повинен бути в три етапи.

Перший етап – оголошення (declaration). На цьому етапі визначається тільки змінна типу посилання (reference) на масив, що містить тип масиву. Для цього записується ім'я типу елементів масиву, квадратними дужками вказується, що оголошується посилання на масив, а не проста змінна, і перелік імен змінних типу посилання, наприклад,

```
double[] a, b;
```

Другий етап – визначення (installation). На цьому етапі вказується кількість елементів масиву – його довжина, виділяється місце для масиву в оперативній пам'яті, змінна одержує адресу масиву. Всі ці дії виробляються однією операцією мови Java – операцією new тип, що виділяє ділянку в оперативній пам'яті для об'єкта зазначеного в операції типу й повертає в якості результату адресу цієї ділянки. Наприклад,

```
a = new double[5];
b = new double[100];
ar = new int[50];
```

Третій етап – ініціалізація (initialization). На цьому етапі елементи масиву одержують початкові значення. Наприклад,

```
a[0] = 0.01; a[1] = -3.4; a[2] = 2.89; a[3] = 4.5; a[4] = -6.7;
for(int i = 0; i < 100; i++) b[i] = 1.0 /i;
for(int i = 0; i < 50; i++) ar[i] = 2 * i + 1;
```

Останній елемент масиву **a** можна записати так: **a [a.length - 1]**, передостанній — **a [a.length - 2]** і т.д. Елементи масиву звичайно перебираються в циклі виду:

```
double aMin = a[0], aMax = aMin;
for(int i = 1; i < a.length; i++){
    if (a[i] < aMin) aMin = a[i];
    if (a[i] > aMax) aMax = a[i];
}
double range = aMax - aMin;
```

Багатовимірні масиви

Елементами масивів в Java можуть бути знову масиви. Можна оголосити:

`char[] [] c;` що еквівалентно `char c[] c[];` або `char c[][];`

Потім визначаємо зовнішній масив:

`c = new char[3][];`

Стає ясно, що `c` – масив, що складається із трьох елементів-масивів. Тепер визначаємо його елементи-масиви:

`c[0] = new char[2];`

`c[1] = new char[4];`

`c[2] = new char[3];`

Нарешті, задаємо початкові значення `c[0][0] = 'a', c[0][1] = 'r', c[1][0] = 'r', c[1][1] = 'a', c[1][2] = 'y' і т.д.`

1.3 Завдання до роботи

1.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

1.3.2 Вивчити основні принципи роботи з Java.

1.3.3 Виконати наступні завдання:

Загальні завдання:

1. Ввести з консолі `n` цілих чисел і помістити їх у масив. На консоль вивести числа-паліндроми, значення яких у прямому й зворотному порядку збігаються.
2. Шляхом перестановки елементів квадратної матриці дійних чисел домогтися того, щоб її максимальний елемент перебував у лівому верхньому куті, наступний по величині – у позиції (2,2), наступний по величині – у позиції (3,3) і т.д., заповнивши в такий спосіб всю головну діагональ.

Індивідуальні завдання:

1. Упорядкувати рядки (стовпці) матриці в порядку зростання значень елементів `k`-го стовпця (рядка).
2. Знайти й вивести найбільше число зростаючих (спадаючих) елементів матриці, що йдуть підряд.
3. Транспонувати квадратну матрицю.

4. Повернути матрицю на 90 (180, 270) градусів проти часової стрілки.
5. Побудувати матрицю, віднімаючи з елементів кожного рядка матриці її середнє арифметичне.
6. Ущільнити матрицю, видаляючи з неї рядки й стовпці, заповнені нулями.
7. Перетворити рядки матриці таким чином, щоб елементи, які дорівнюють нулю, розташовувалися після всіх інших.
8. Знайти кількість всіх седлових точок матриці. (Матриця A має седлову точку $A_{i,j}$, якщо $A_{i,j}$ є мінімальним елементом в i -й рядку й максимальним в j -м стовпці).
9. Знайти число локальних мінімумів. (Сусідами елемента матриці назвемо елементи, що мають із ним загальну сторону або кут. Елемент матриці називається локальним мінімумом, якщо він строго менше всіх своїх сусідів).
10. Перебудувати задану матрицю, переставляючи в ній стовпці так, щоб значення їхніх характеристик убували. (Характеристикою стовпця прямокутної матриці називається сума модулів його елементів).

1.3.4 Оформити звіт з роботи.

1.3.5 Відповісти на контрольні питання.

1.4 Зміст звіту

1.4.1 Тема та мета роботи.

1.4.2 Завдання до роботи.

1.4.3 Короткі теоретичні відомості.

1.4.4 Текст розробленої програми.

1.4.5 Копії екрану, що відображають результати виконання лабораторної роботи.

1.4.6 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображають результати виконання роботи та їх критичний аналіз.

1.5 Контрольні запитання

1.5.1 Які типи даних Ви знаєте в мові Java?

- 1.5.2 Які логічні операції Ви знаєте в мові Java?
- 1.5.3 Які побітові операції Ви знаєте в мові Java?
- 1.5.4 Які операції зсуву Ви знаєте в мові Java?
- 1.5.5 Які способи запису цілих чисел Ви знаєте?
- 1.5.6 Які правила написання імен в Java Ви знаєте?
- 1.5.7 Які особливості операцій присвоювання Ви знаєте?
- 1.5.8 Які особливості логічних операцій Ви знаєте?
- 1.5.9 Як масиви розташовуються в пам'яті в мові Java?
- 1.5.10 Які особливості роботи Java з Unicode?

2 ЛАБОРАТОРНА РОБОТА № 2 СТВОРЕННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ

2.1 Мета роботи

Навчитися створити в NetBeans додаток з графічним інтерфейсом.

2.2 Основні теоретичні відомості

Середовище IDE NetBeans являє собою стандартне модульне інтегроване середовище розробки (Integrated Development Environment; IDE), написане мовою програмування Java. Проект NetBeans складається з інтегрованих середовищ розробки з відкритим вихідним кодом, написаним мовою програмування Java, яка може використовуватися як загальна платформа для створення додатків будь-якого типу.

Середовище NetBeans – це інтегроване середовище для розроблювачів, засіб для програмістів, що допомагає писати, компілювати та налагоджувати програми. Воно написана мовою програмування Java, однак може підтримувати розробку і на інших мовах. Існує також велика кількість модулів, що розширюють його функціональність. IDE NetBeans – це безкоштовний продукт без обмежень на область його застосування.

Доступна також платформа NetBeans – модульна й розширювальна основа, використовувана як програмне ядро для створення більших настільних додатків. Незалежні постачальники – партнери Sun, надають різні додаткові модулі, які легко інтегруються в платформу й можуть бути використані для розробки їхніх засобів і рішень.

Запустимо інтегроване середовище розробки (IDE) NetBeans. На екрані з'явиться середовище з її стартовою сторінкою Welcome, зображеною на рис. 2.1.

У лівому верхньому вікні “Projects” буде показуватися дерево проектів. Це вікно може бути використане для одночасного зображення довільного числа проектів. За замовчуванням усі дерева згорнуті, і потрібні вузли варто розвертати щигликом по вузлі з “плюсиком” або подвійним щигликом по відповідному імені.

У лівому нижньому вікні “Navigator” буде показуватися список імен членів класу додатка – імена змінних і підпрограм. Подвійний щиглик по імені приводить до того, що у вікні редактора вихідного коду відбувається перехід на те місце, де задана відповідна змінна або підпрограма.

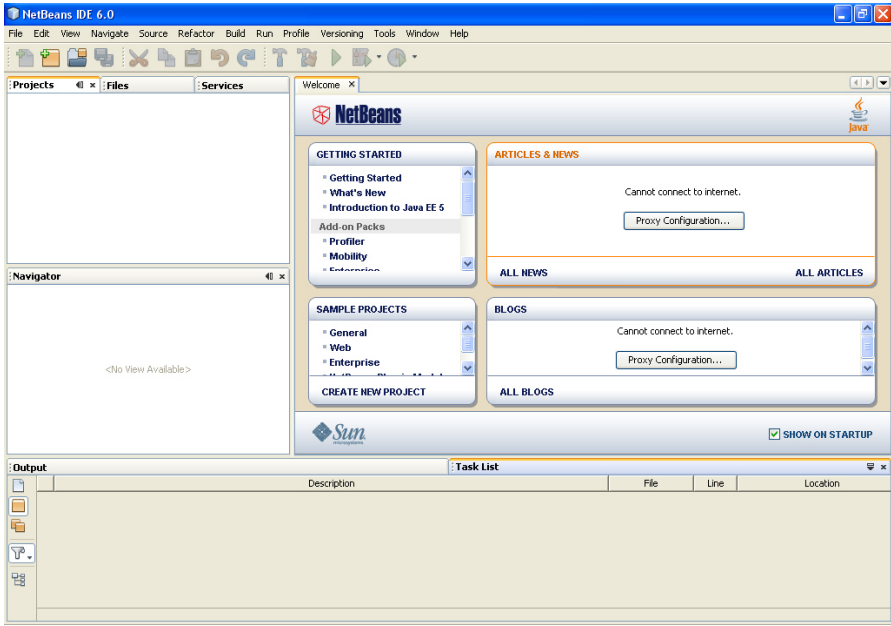


Рисунок 2.1 – Стартова сторінка

Розглянемо, як виглядає згенерований вихідний код нашого додатка Java:

```

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package javaapplication1;

/**
 *
 * @author User
 */
public class Main {

```

```

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    // TODO code application logic here
}

```

Спочатку йде багаторядковий коментар `/* ... */`. Він містить інформацію про ім'я класу й час його створення.

Потім оголошується, що наш клас буде знаходитися в пакеті `javaapplication1`. Після цього йде багаторядковий коментар `/** ... */`, призначений для автоматичного створення документації по класі. У ньому є присутня інструкція завдання метаданих за допомогою виразу `@author` – інформація про автора проекту для утиліти створення документації `javadoc`. Метадані – це якась інформація, що не відноситься до роботи програми й не включається в неї при компіляції, але супроводжує програму й може бути використана іншими програмами для перевірки прав на доступ до неї або її поширення, перевірки сумісності з іншими програмами, вказівки параметрів для запуску класу й т.п. У даному місці вихідного коду ім'я “User” береться середовищем розробки з операційної системи за ім'ям папки користувача.

Далі треба виконати оголошення класу `Main`, що є головним класом додатка. У ньому оголошена загальнодоступна (`public`) підпрограма-метод `main`. Він викликає всі класи й об'єкти додатка.

```

public static void main(String[] args) {
}

```

Він є методом класу, і тому для його роботи немає необхідності в створенні об'єкту, що є екземпляром класу `Main`. Хоча якщо цей об'єкт створюється, це відбувається під час роботи методу `main`.

Параметром `args` цього методу є масив рядків, що має тип `String[]`. Це параметри командного рядка, які передаються в додаток при його запуску.

Після закінчення виконання методу `main` додаток завершує свою роботу.

Розглянемо, з яких частин складається проект NetBeans. На рис. 2.2 показані основні елементи, що знаходяться в середовищі розробки. Це `Source Packages` (пакети вихідного коду), `Test Packages` (пакети тестування), `Libraries` (бібліотеки) і `Test Libraries` (бібліотеки підтримки тестування).

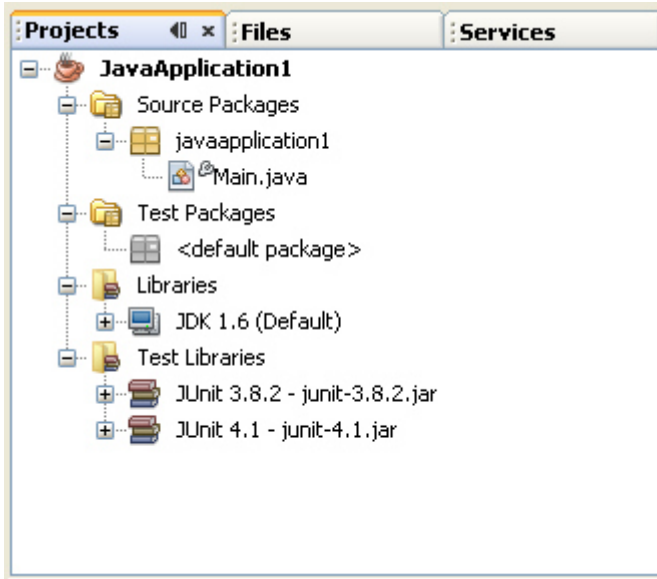


Рисунок 2.2 – Основні елементи проекту

У компонентній моделі NetBeans пакети додатку поєднуються в єдину конструкцію – модуль. Модулі NetBeans є базовою конструкцією не тільки для створення додатків, але й для написання бібліотек.

На відміну від бібліотек Java, скомпільований модуль – це не набір великої кількості файлів, а всього один файл, архів JAR (Java Archive, архів Java). У нашому випадку він має те ж ім'я, що й додаток, і розширення .jar: це файл `JavaApplication1.jar`.

У дереві проектів видні всі відкриті проекти. Один з відкритих проектів є головним (Main Project) – саме він буде запускатися на виконання по `Run/ Run Main Project`. Для того щоб установити який-небудь із відкритих проектів у якості головного, треба в дереві проектів за допомогою правої кнопки миші клацнути по імені проекту й вибрати пункт меню `Set Main Project`.

Створення Java додатка із графічним інтерфейсом. Оскільки, як відзначалося вище, консольний ввід-вивід використовується досить рідко, тепер спробуємо створити додаток із графічним інтерфейсом.

Екранною формою називається область, що видна на екрані у вигляді вікна з різними елементами – кнопками, текстом, списками, що випадають, й т.п. А самі ці елементи називаються компонентами.

Форма – це об'єкт, звичайної прямокутної форми, яку можна застосовувати для надання інформації користувачеві й для обробки введення інформації від користувача.

У нашому проєкті HelloWorld відкрисмо File /New File. У діалозі, що відкрився, у списку категорій виберіть Swing GUI Forms. Виберіть тип файлу JFrame Form. Натисніть Next> (рис. 2.3). Після цього введемо назву створюваного класу й пакет, де він буде зберігатися. І натискаємо Finish.

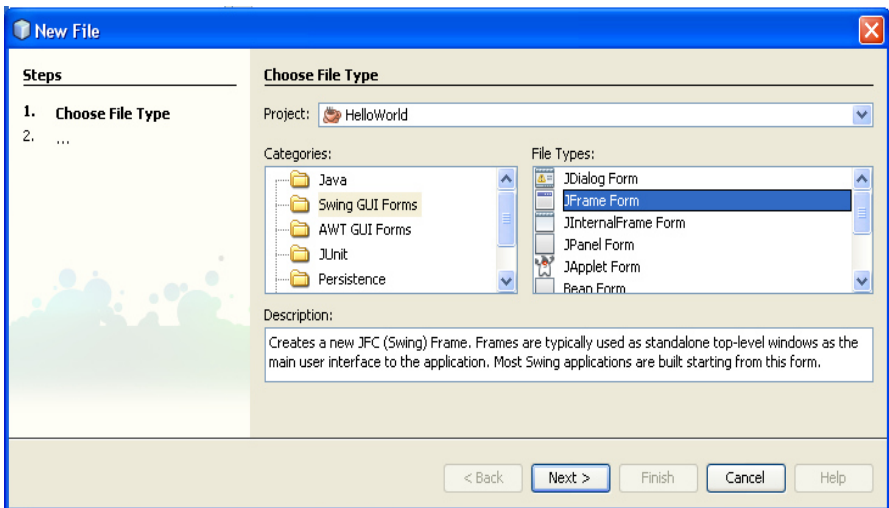


Рисунок 2.3 – Створення проєкту із графічним інтерфейсом

IDE створює новий клас. І відкриває його у візуальному редакторі форми. Тепер для додавання елементів на наш JFrame, досить просто використати простий Drag and Drop.

Додамо на форму кнопки JButton і текстове поле JTextField (рис. 2.4). Логікою роботи програми буде вивід вітання в текстовому полі при натисканні першої кнопки, і очищення поля виводу при натисканні іншої кнопки.

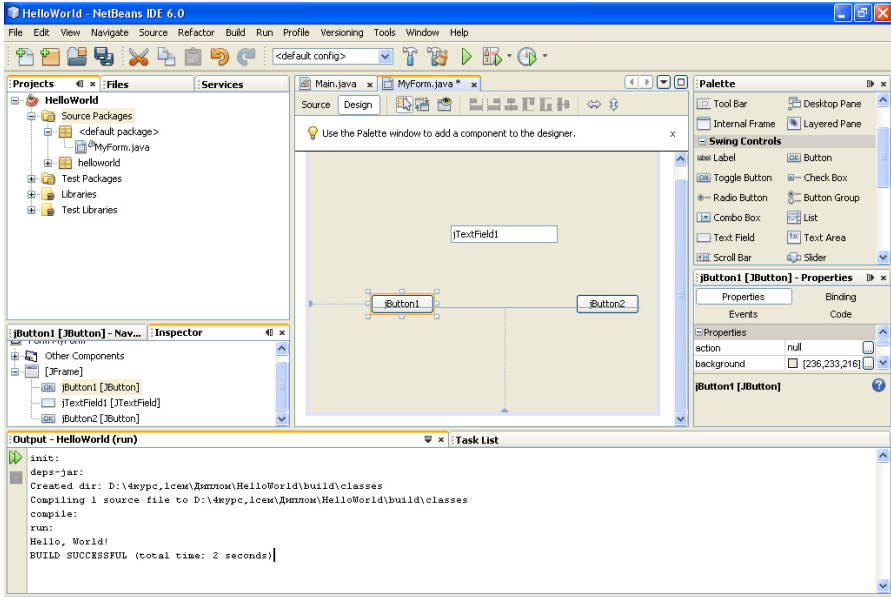


Рисунок 2.4 – Додавання елементів на форму

Вище на рис. 2.4. наведений дизайн форми. Всі елементи, що додають на форму, відображаються в панелі, що називається Inspector. У ній легко одержати доступ до будь-якого елемента на формі. Навіть якщо він прямо не видний на формі (наприклад, такі елементи як ButtonGroup).

Для текстового поля заберемо текст jTextField1. Для цього на панелі властивостей поля знайдемо властивість тексту і залишимо порожній рядок. Для кнопки 1 у полі властивості текст наберемо Greeting, а для кнопки 2 наберемо Clear. Ну й для всіх доданих нами елементів, поміняємо назви змінних на назви, які краще відображають їхню функціональність. Це робиться виділенням елемента, і в його контекстному меню вибором пункту Change Variable Name ... (рис. 2.5).

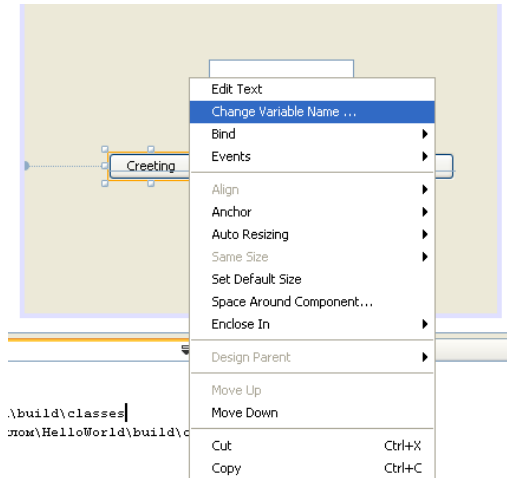


Рисунок 2.5 – Перейменування змінної

У діалозі, що з'явився (Rename), вводимо нове ім'я змінної `jbGreet`. Точно так само ім'я змінної для другої кнопки буде `jbClear`, а для текстового поля `tfGreetMessage`.

Після цих дій додамо два оброблювачі подій для кожної із кнопок, які будуть створюватися при натисканні. Це найпростіше зробити також з контекстного меню елементів (рис. 2.6).

Вибір пункту `ActionPerformed` означає, що ми хочемо додати оброблювач цієї події (натискання кнопки) для нашого елемента `jbGreet`. Після вибору цього пункту, середовище згенерує й покаже нам функцію, що буде викликатися при натисканні на кнопку.

```
private void jbGreetActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

Підкоригуємо цю функцію:

```
private void jbGreetActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    tfGreetMessage.setText("Hello");
}
```

У такий же спосіб створимо оброблювач події для натискання кнопки `jbClear`:

```
private void jbClearActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    tfGreetMessage.setText("");
}
```

А тепер запускаємо програму на виконання.

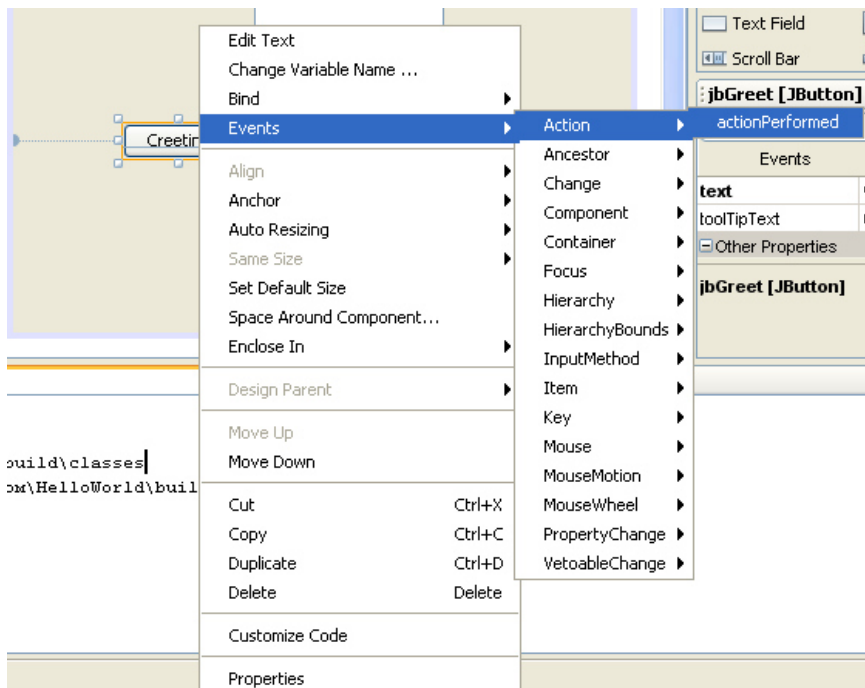


Рисунок 2.6 – Додавання оброблювача подій

2.3 Завдання до роботи

2.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

2.3.2 Виконати наступні завдання:

Загальні завдання:

1. Створити додаток Java, описаний вище.
2. Додайте до кожної кнопки спливаючу підказку.
3. На основі проекту з наявним кодом – додатку з графічним інтерфейсом, створити додаток з кнопкою «Вихід», яка буде закривати додаток, та кнопкою «Натисни мене», яка

буде визивати діалогову панель з повідомленням «Мене натиснули».

4. На основі проекту із графічним користувальницьким інтерфейсом створіть новий проект. У ньому задати змінні та зробити кнопки з назвами для кожного з дійсних та цілочисельних типів. При натисканні на кнопку повинні створюватися діалогові панелі з повідомленням про ім'я та значення кожної змінної.

2.3.3 Оформити звіт з роботи.

2.3.4 Відповісти на контрольні питання.

2.4 Зміст звіту

2.4.1 Тема та мета роботи.

2.4.2 Завдання до роботи.

2.4.3 Короткі теоретичні відомості.

2.4.4 Текст розробленої програми.

2.4.5 Копії екрану, що відображають результати виконання лабораторної роботи.

2.4.6 Висновки, що містять відповіді на контрольні запитання (3 шт. за вибором студента), а також відображають результати виконання роботи та їх критичний аналіз.

2.5 Контрольні запитання

2.5.1 Основні режими компіляції проекту.

2.5.2 Як запустити документацію по проекту?

2.5.3 Що знаходиться на панелі Inspector.

2.5.4 Як додати оброблювач подій?

2.5.5 У якому режимі редагуються вкладені пункти меню.

2.5.6 Назвіть основні керуючі конструкції.

2.5.7 У чому основне розходження операторів Break та Continue?

2.5.8 Чи можна використати числа, як значення типу Boolean?

3 ЛАБОРАТОРНА РОБОТА № 3 ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ В JAVA

3.1 Мета роботи

Вивчити основи об'єктно-орієнтовного підходу програмування в Java.

3.2 Основні теоретичні відомості

3.2.1 Як описати клас і підклас

Опис класу починається зі слова `class`, після якого записується ім'я класу. Згідно угоди "Code Conventions" рекомендують починати ім'я класу із заголовної букви.

Перед словом `class` можна записати модифікатори класу (`class modifiers`). Це одне зі слів `public`, `abstract`, `final`, `strictfp`. Перед ім'ям вкладеного класу можна поставити, крім того, модифікатори `protected`, `private`, `static`.

Тіло класу, у якому в будь-якому порядку записуються поля, методи, вкладені класи й інтерфейси, міститься у фігурних дужках.

При описі поля вказується його тип, потім, через пробіл, ім'я й, може бути, початкове значення після знака рівності, яке можна записати константним вираженням.

Опис поля може починатися з одного або декількох необов'язкових модифікаторів `public`, `protected`, `private`, `static`, `final`, `transient`, `volatile`. Якщо треба поставити кілька модифікаторів, то JLS рекомендує розташовувати їх в зазначеному порядку, оскільки деякі компілятори вимагають певного порядку запису модифікаторів.

При описі методу вказується тип повертаємого значення або слово `void`, потім, через пробіл, ім'я методу, потім, у дужках, список параметрів. Після цього у фігурних дужках розписується виконуваний метод.

Опис методу може починатися з модифікаторів `public`, `protected`, `private`, `abstract`, `static`, `final`, `synchronized`, `native`, `strictfp`.

У списку параметрів через кому записуються тип і ім'я кожного параметру. Перед типом якого-небудь параметра може стояти

модифікатор `final`. Такий параметр не можна змінювати усередині методу. Список параметрів може бути відсутнім, але дужки повинні бути.

Перед початком роботи методу для кожного параметра виділяється місце у оперативній пам'яті, у яке копіюється значення параметра, задане при звертанні до методу. Такий спосіб називається передачею параметрів за значенням.

У програмі 3.1 показано, як можна оформити метод розподілу навіпіл для знаходження кореня нелінійного рівняння.

Програма 3.1. Знаходження кореня нелінійного рівняння методом бісекції

```
class Bisection2{
    private static double final EPS = 1e-8; // Константа
    private double a = 0.0, b = 1.5, root; // Закриті поля
    public double getRoot(){return root;} // Метод доступу
    private double f(double x)
    {
        return x*x*x - 3*x*x + 3; // Або щось інше
    }
    private void bisect() { // Параметрів немає -
        // метод працює з полями екземпляра
        double y = 0.0; // Локальна змінна - не поле
        do{
            root = 0.5 * (a + b); v = f(root);
            if(Math.abs(y) < EPS) break;
            // Корінь знайдений. Виходимо із циклу
            // Якщо на кінцях відрізка [a; root]
            // функція має різні знаки:
            if(f(a) * y < 0.0) b = root;
            // виходить, корінь тут
            // Переносимо крапку b у крапку root
            // У протилежному випадку:
            else a = root;
            // переносимо крапку a в крапку root
            // Продовжуємо, поки [a; b] не стане малий
        } while(Math.abs(b-a) >= EPS);
    }
    public static void main(String[] args){
        Bisection2 b2 = new Bisection2();
        b2.bisect();
        System.out.println("x = " +
            b2.getRoot() + // Звертаємося до кореня через метод доступу
            ", f() = " + b2.f(b2.getRoot()));
    } }
```

В описі методу `f()` використовується старий, процедурний стиль: метод одержує аргумент, обробляє його й повертає результат. Опис методу `bisect` про виконано в дусі ООП: метод активний, він сам звертається до полів екземпляра `b2` і сам заносить результат у

потрібне поле. Метод `bisect()` - це внутрішній механізм класу `Bisection2`, тому він закритий(`private`).

Ім'я методу, число й типи параметрів утворюють сигнатуру(signature) методу. Компілятор розрізняє методи не по їхніх іменах, а по сигнатурах. Це дозволяє записувати різні методи з однаковими іменами, що розрізняються числом і/або типами параметрів.

Тип повертаємого значення, не входить у сигнатуру методу, виходить, методи не можуть розрізнятися тільки типом результату їхньої роботи.

У класі `Automobile` можна записати метод `moveTo(int x, int y)`, позначивши пункт призначення географічними координатами. Можна визначити ще метод `moveTo(string destination)` для вказівки географічної назви пункту призначення й звертатися до нього так:

```
oka.moveTo("Москва");
```

Таке дублювання методів називається перевантаженням(`overloading`). Перевантаження методів дуже зручне у використанні. Наприклад метод `println()` можна використовувати для виводу не піклуючись про те, дані якого саме типу треба виводити на екран. Насправді використовуються різні методи з тим самим ім'ям `println`. Звичайно, всі ці методи треба ретельно спланувати й заздалегідь описати в класі. Це й зроблено в класі `Printstream`, де представлено біля двадцяти методів `print()` і `println()`.

Якщо ж записати метод з тим же ім'ям у підкласі, наприклад:

```
class Truck extends Automobile{
    void moveTo(int x, int y){
        // Якісь дії
    }
    // Щось ще
}
```

то він перекриє метод суперкласу. Визначивши екземпляр класу `Truck`, наприклад:

```
Truck gazel = new Truck();
```

і записавши `gazel.moveTo(25, 150)`, ми звернемося до методу класу `Truck`. Відбудеться перевизначення(`overriding`) методу.

При перевизначенні права доступу до методу можна тільки розширити. Відкритий метод `public` повинен залишитися відкритим, захищений `protected` може стати відкритим.

Можна з середини підкласу звернутися до методу суперкласу, якщо уточнити ім'я методу, словом `super`, наприклад,

`super.moveTo(30, 40)`. Можна уточнити й ім'я методу, записаного в цьому ж класі, словом `this`, наприклад, `this.moveTo(50, 70)`, але в цьому випадку це вже зайво. У такий же спосіб можна уточнювати й співпадаючі імена полів, а не тільки методів.

Перевизначення методів приводить до цікавих результатів. Якщо у класі `Pet` описати метод `voice()`. Перевизначити його в підкласах і використовувати в класі `Chorus`, як показано в програмі 3.2.

Програма 3.2. Приклад поліморфного методу

```
abstract class Pet{
    abstract void voice();
}
class Dog extends Pet{
    int k = 10;
    void voice(){
        System.out.println("Gav-gav!");
    }
}
class Cat extends Pet{
    void voice() {
        System.out.println("Miaou!");
    }
}
class Cow extends Pet{
    void voice(){
        System.out.println("Mu-u-u!");
    }
}
public class Chorus(
    public static void main(String[] args){
        Pet[] singer = new Pet[3];
        singer[0] = new Dog();
        singer[1] = new Cat();
        singer[2] = new Cow();
        for(int i = 0; i < singer.length; i++)
            singer[i].voice();
    }
}
```

На рис. 3.1 показаний вивід цієї програми.

Вся справа тут у визначенні поля `singer[]`. Хоча масив посилань `singer []` має тип `Pet`, кожен його елемент посилається на об'єкт свого типу `Dog`, `Cat`, `cow`. При виконанні програми викликається метод конкретного об'єкта, а не метод класу, яким визначалося ім'я посилання. Так в Java реалізується поліморфізм.

У мові Java всі методи є віртуальними функціями. Клас `Pet` і метод `voice()` є абстрактними.

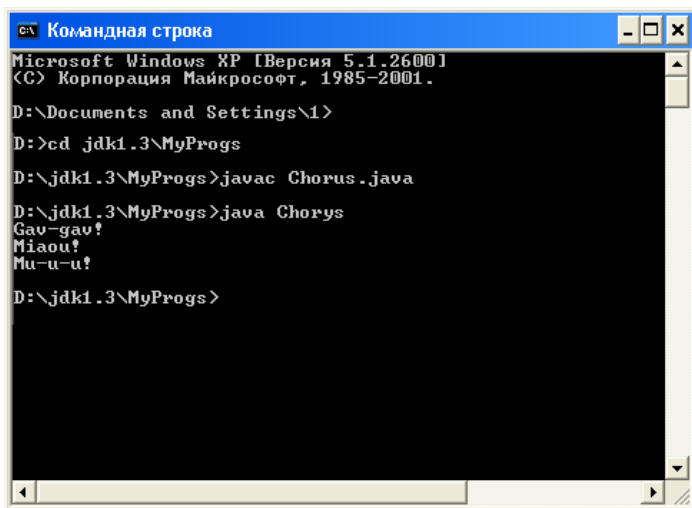


Рисунок 3.1 – Результат виконання програми Chorus

3.2.2 Абстрактні методи й класи

Якщо клас містить хоч один абстрактний метод, то створити його екземпляри, а тим більше використати їх, не вдасться. Такий клас стає абстрактним, що обов'язково треба вказати модифікатором `abstract`.

Хоча елементи масиву `singer []` посилаються на підкласи `Dog`, `Cat`, `Cow`, але все-таки це змінні типу `Pet` і посилатися вони можуть тільки на поля й методи, описані в суперкласі `Pet`. Додаткові поля підкласу для них недоступні. Тому метод, що реалізується в декількох підкласах, доводиться виносити в суперклас, а якщо там його не можна реалізувати, то оголосити абстрактним. Таким чином, абстрактні класи групуються на вершині ієрархії класів.

Можна задати порожню реалізацію методу, просто поставивши пари фігурних дужок, нічого не написавши між ними, наприклад:

```
void voice() {}
```

Вийде повноцінний метод. Але це штучне рішення, що заплутує структуру класу.

Замкнути ж ієрархію можна остаточними класами.

3.2.3 Остаточні члени й класи

Позначивши метод модифікатором `final`, можна заборонити його перевизначення в підкласах. Це зручно з метою безпеки. Ви можете бути впевнені, що метод виконує ті дії, які ви задали. Саме так визначені математичні функції `sin()`, `cos()` та інші в класі `Math`. Метод `Math.cos(x)` обчислює саме косинус числа `x`. Зрозуміло, такий метод не може бути абстрактним.

Для повної безпеки, поля, які оброблюються остаточними методами, варто зробити закритими(`private`).

Якщо ж позначити модифікатором `final` весь клас, то його взагалі не можна буде розширити. Так визначений, наприклад, клас `Math`:

```
public final class Math{... }
```

Для змінних модифікатор `final` має зовсім інший зміст. Якщо позначити модифікатором `final` опис змінної, то її значення(а воно повинне бути обов'язково задане або тут же, або в блоці ініціалізації або в конструкторі) не можна змінити ні в підкласах, ні в самому класі. Змінна перетворюється в константу. Саме так у мові Java визначаються константи:

```
public final int MIN_VALUE = -1, MAX_VALUE = 9999;
```

Згідно зі згодою "Code Conventions" константи записуються прописними буквами, слова в них розділяються знаком підкреслення.

На самій вершині ієрархії класів Java стоїть клас `Object`.

3.2.4 Клас Object

Якщо при описі класу ми не використовуємо ніяке розширення, тобто не пишемо слово `extends` і ім'я класу за ним, як при описі класу `Pet`, то Java вважає цей клас розширенням класу `Object`, і компілятор дописує це за нас:

```
class Pet extends Object{... }
```

Сам же клас `Object` не є може бути спадкоємцем, від нього починається ієрархія будь-яких класів Java. Зокрема, всі масиви - прямі спадкоємці класу `Object`.

Оскільки такий клас може містити тільки загальні властивості всіх класів, у нього включено лише кілька самих загальних методів, наприклад, метод `equals()`, що порівнює даний об'єкт на рівність із об'єктом, заданим в аргументі, і повертаючий логічне значення. Його можна використати так:

```
Object obj1 = new Dog(), obj 2 = new Cat();
if(obj1.equals(obj2)) ...
```

Посилання можна порівнювати на рівність і нерівність:

```
obj1 == obj2; obj1 != obj 2;
```

У цьому випадку зіставляються адреси об'єктів, чи не вказують посилання на один і той самий об'єкт.

Метод equals() же порівнює вміст об'єктів у їхньому поточному стані, фактично він реалізований у класі Object як тотожність: об'єкт дорівнює тільки самому собі. Тому його часто перевизначають у підкласах, більше того, правильно спроектовані, класи повинні перевизначити методи класу Object, якщо їх не влаштовує стандартна реалізація.

Другий метод класу Object, який варто перевизначати в підкласах, – метод toString(). Це метод без параметрів, що намагається вміст об'єкта перетворити в рядок символів і повертає об'єкт класу String.

До цього методу виконуюча система Java звертається щораз, коли потрібно представити об'єкт у вигляді рядка, наприклад, у методі printing.

3.2.5 Конструктори класу

В операції new, що визначає екземпляри класу, повторюється ім'я класу з дужками. Такий "метод" називається конструктором класу (class constructor). Конструктор є в будь-якому класі. Навіть якщо ви його не написали, компілятор Java сам створить конструктор за замовчуванням (default constructor), який буде порожній, він не робить нічого, крім виклику конструктора суперкласу.

Конструктор виконується автоматично при створенні екземпляра класу, після розподілу пам'яті й обнуління полів, але до початку використання створюваного об'єкта.

Конструктор не повертає ніякого значення. Тому в його описі не пишеться навіть слово void, але можна задати один із трьох модифікаторів public, protected або private.

Конструктор не є методом, він навіть не вважається членом класу. Тому його не можна успадковувати або перевизначити в підкласі.

Тіло конструктора може починатися:

– з виклику одного з конструкторів суперкласу, для цього записується слово super() з параметрами в дужках, якщо вони потрібні;

– з виклику іншого конструктора того ж класу, для цього записується слово `this()` з параметрами в дужках, якщо вони потрібні.

Якщо ж `super()` на початку конструктора не має, то спочатку виконується конструктор суперкласу без аргументів, потім відбувається ініціалізація полів значеннями, зазначеними при їхньому оголошенні, а вже потім те, що записано в конструкторі.

У всьому іншому конструктор можна вважати звичайним методом, у ньому дозволяється записувати будь-які оператори, навіть оператор `return`, але тільки порожній, без усякого повертаємого значення.

У класі може бути декілька конструкторів. Оскільки в них те саме ім'я, що збігається з ім'ям класу, то вони повинні відрізнятися типом і/або кількістю параметрів.

3.2.6 Операція new

У першому випадку в якості операнда вказується тип елементів масиву й кількість його елементів у квадратних дужках, наприклад:

```
double a[] = new double[100];
```

У другому випадку операндом є конструктор класу. Якщо конструктора в класі немає, то викликається конструктор за замовчуванням.

Числові поля класу одержують нульові значення, логічні поля - значення `false`, посилання - значення `null`.

Результатом операції `new` буде посилання на створений об'єкт. Це посилання може бути привласнене змінній типу посилання на даний тип:

```
Dog k9 = new Dog() ;
```

але може використовуватися й безпосередньо

```
new Dog().voice();
```

Тут після створення безіменного об'єкта відразу виконується його метод `voice()`.

3.2.7 Статичні члени класу

Різні екземпляри одного класу мають зовсім незалежні друг від друга поля-, що приймають різні значення. Зміна поля в одному екземплярі ніяк не впливає на те ж поле в іншому екземплярі. У кожному екземплярі для таких полів виділяється своя пам'ять. Тому

такі поля називаються змінними екземпляра класу(instance variables) або змінними об'єкта.

Іноді треба визначити поле, загальне для всього класу, зміна якого в одному екземплярі спричинить зміну того ж поля у всіх екземплярах. Наприклад, ми хочемо в класі Automobile відзначити порядковий заводський номер автомобіля. Такі поля називаються змінними класу(class variables). Для змінних класу виділяється тільки одна адреса у пам'яті, загальна для всіх екземплярів. Змінні класу утворюються в Java модифікатором static. У програмі 3.3 ми записуємо цей модифікатор при визначенні змінної number.

Програма 3.3. Статична змінна

```
class Automobile {
    private static int number;
    Automobile(){
        number++;
        System.out.println("From Automobile constructor:"+
            " number = "+number);
    }
}

public class AutomobileTest{
    public static void main(String[] args){
        Automobile lada2105    = new Automobile(),
        fordScorpio = new Automobile(),
        oka                = new Automobile!());
    }
}
```

Одержуємо результат, показаний на рис. 3.2.

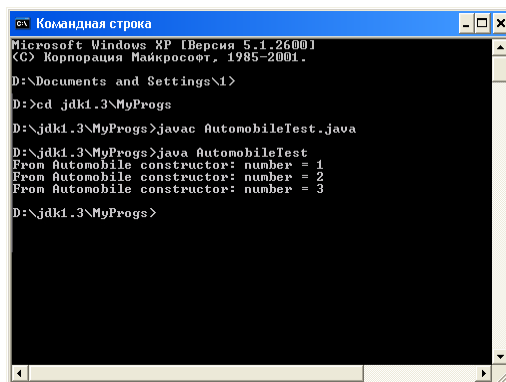


Рисунок 3.2 – Зміна статичної змінної

До статичних змінних можна звертатися з ім'ям класу, Automobile.number, а не тільки з ім'ям екземпляра, lada2105.number,

причому це можна робити, навіть якщо не створений жоден екземпляр класу.

Для роботи з такими статичними змінними звичайно створюються статичні методи, позначені модифікатором `static`. Для методів слово `static` має зовсім інший зміст. Виконуюча система Java завжди створює в пам'яті тільки одну копію машинного коду методу, яка поділяється між всіма екземплярами, незалежно від того, статичний це метод чи ні.

Основна особливість статичних методів - вони виконуються відразу у всіх екземплярах класу. Більше того, вони можуть виконуватися, навіть якщо не створений жоден екземпляр класу. Досить уточнити ім'я методу ім'ям класу(а не ім'ям об'єкта), щоб метод міг працювати. Саме так ми використовується метод класу `Math`, не створюючи його екземпляри, а просто записуючи `Math.abs(x)`, `Math.sqrt(x)`.

Тому статичні методи називаються методами класу(class methods), на відміну від нестатичних методів, які називаються методами екземпляра(instance methods).

Звідси випливають інші особливості статичних методів:

- у статичному методі не можна використовувати посилання `this` і `super`;
- у статичному методі не можна прямо, не створюючи екземплярів, посилатися на нестатичні поля й методи;
- статичні методи не можуть бути абстрактними;
- статичні методи перевизначаються в підкласах тільки як статичні.

Статичні змінні ініціалізуються ще до початку роботи конструктору, але при ініціалізації можна використовувати тільки константні вирази. Якщо ж ініціалізація вимагає складних обчислень, наприклад, циклів для завдання значень елементам статичних масивів або звертань до методів, то ці обчислення записуються у блоці `static`, що виконується до запуску конструктора:

```
static int[] a = new a[10];
static {
    for(int k = 0; k < a.length; k++)
        a[k] = k * k;
}
```

Оператори, вкладені в такий блок, виконуються тільки один раз, при першому завантаженні класу, а не при створенні кожного екземпляру.

3.2.8 Клас Complex

Комплексні числа часто застосовуються в графічних перетвореннях, у побудові фракталів, фізиці, математиці.

Програма 3.4. Клас Complex

```
class Complex {
    private static final double EPS = 1e-12; // Точність обчислень
    private double re, im;                  // Дійсна й мніма частина
                                           // Чотири конструктори

    Complex(double re, double im) {
        this, re = re; this.im = im;
    }
    Complex(double re){this(re, 0.0); }
    Complex(){this(0.0, 0.0); }
    Complex(Complex z){this(z.re, z.im) ; }

    // Методи доступу
    public double getRe(){return re;}
    public double getImF(){return im;}
    public Complex get(){return new Complex(re, im);}
    public void setRe(double re){this.re = re;}
    public void setIm(double im){this.im = im;}
    public void set(Complex z){re = z.re; im = z.im;}

    // Модуль і аргумент комплексного числа
    public double mod(){return Math.sqrt(re * re + im * im);}
    public double arg(){return Math.atan2(re, im);}

    // Перевірка: дійсне число?
    public boolean isReal(){return Math.abs(im) < EPS;}
    public void pr(){ // Вивід на екран
        System.out.println(re +(im < 0.0 ? "" : "+" ) + im + "i");
    }

    // Перевизначення методів класу Object
    public boolean equals(Complex z){
        return Math.abs(re -z.re) < EPS &&
            Math.abs(im - z.im) < EPS;
    }
    public String toString(){
        return "Complex: " + re + " " + im;
    }

    // Методи, що реалізують операції +=, -=, *=, /=
    public void add(Complex z){re += z.re; im += z.im;}
    public void sub(Complex z){re -= z.re; im -= z.im;}
    public void mul(Complex z){
        double t = re * z.re - im * z.im;
        im = re * z.im + im * z.re;
        re = t;
    }
    public void div(Complex z){
        double m = z.mod();
        double t = re * z.re - im * z.im;
        im =(im * z.re - re * z.im) / m;
```

```

        re = t / m;
    }
    // Методи, що реалізують операції +, -, *, /
    public Complex plus(Complex z){
        return new Complex(re + z.re, im + z.im);
    }
    public Complex minus(Complex z){
        return new Complex(re - z.re, im - z.im);
    }
    public Complex asterisk(Complex z){
        return new Complex(
            re * z.re - im * z.im, re * z.im + im * z.re);
    }
    public Complex slash(Complex z){
        double m = z.mod();
        return new Complex(
            (re * z.re - im * z.im) / m, (im * z.re - re * z.im) / m);
    }
}
// Перевіримо роботу класу Complex
public class ComplexTest{
    public static void main(String[] args){
        Complex z1 = new Complex(),
            z2 = new Complex(1.5),
            z3 = new Complex(3.6, -2.2),
            z4 = new Complex(z3);
        System.out.println(); // Залишаємо порожній рядок
        System.out.print("z1 = "); z1.pr();
        System.out.print("z2 = "); z2.pr();
        System.out.print("z3 = "); z3.pr();
        System.out.print("z4 = "); z4.pr();
        System.out.println(z4); // Працює метод toString()
        z2.add(z3);
        System.out.print("z2 + z3 = "); z2.pr();
        z2.div(z3);
        System.out.print("z2 / z3 = "); z2.pr();
        z2 = z2.plus(z2);
        System.out.print("z2 + z2 = "); z2.pr();
        z3 = z2.slash(z1);
        System.out.print("z2 / z1 = "); z3.pr();
    }
}

```

На рис. 3.3 показаний вивід цієї програми.

```

Командная строка
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

D:\Documents and Settings\1>
D:>cd jdk1.3\MyProgs
D:\jdk1.3\MyProgs>javac ComplexTest.java
D:\jdk1.3\MyProgs>java ComplexTest

z1 = 0.0+0.0i
z2 = 1.5+0.0i
z3 = 3.6-2.2i
z4 = 3.6-2.2i
Complex: 3.6 -2.2
z2 + z3 = 5.1-2.2i
z2 / z3 = 3.204547330826246+0.7821750141809625i
z2 + z2 = 6.409094661652492+1.564350028361925i
z2 / z1 = NaN+NaNi

D:\jdk1.3\MyProgs>

```

Рисунок 3.3 – Вивід програми ComplexTest

3.3 Завдання до роботи

3.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

3.3.2 Виконати наступні завдання:

Загальні завдання:

5. Реалізувати клас, що представляє геометричне тіло-циліндр. Необхідно реалізувати методи обчислення площі його поверхні й обсягу.
6. Реалізувати клас, що представляє геометричне тіло-тетраedr. Необхідно реалізувати методи обчислення площі його поверхні й обсягу.

Індивідуальні завдання:

1. Реалізувати клас, що представляє геометричну фігуру – окружність. Необхідно реалізувати методи обчислення периметра й діаметра. Для визначення вершин використати координати декартової системи координат

2. Реалізувати клас, що представляє геометричну фігуру – ромб. Необхідно реалізувати методи обчислення периметра й діаметр описаної окружності. Для визначення вершин використати координати декартової системи координат
3. Реалізувати клас, що представляє геометричну фігуру – паралелограм. Необхідно реалізувати методи обчислення периметра й діаметр вписаної окружності. Для визначення вершин використати координати декартової системи координат
4. Реалізувати клас, що представляє геометричну фігуру – трапецію. Необхідно реалізувати методи обчислення периметра й площі. Для визначення вершин використати координати декартової системи координат
5. Реалізувати клас, що представляє геометричну фігуру – трикутник. Необхідно реалізувати методи обчислення периметра й довжини ребер. Для визначення вершин використати координати декартової системи координат
6. Реалізувати клас, що представляє геометричну фігуру – окружність. Необхідно реалізувати методи обчислення площі й радіуса. Для визначення вершин використати координати декартової системи координат
7. Реалізувати клас, що представляє геометричну фігуру – ромб. Необхідно реалізувати методи обчислення площі й бісектрис. Для визначення вершин використати координати декартової системи координат
8. Реалізувати клас, що представляє геометричну фігуру – паралелограм. Необхідно реалізувати методи обчислення площі й радіуса описаної окружності. Для визначення вершин використати координати декартової системи координат
9. Реалізувати клас, що представляє геометричну фігуру – трапецію. Необхідно реалізувати методи обчислення площі й радіуса вписаної окружності. Для визначення вершин використати координати декартової системи координат
10. Реалізувати клас, що представляє геометричну фігуру – трикутник. Необхідно реалізувати методи обчислення

площі й бісектрис. Для визначення вершин використати координати декартової системи координат

3.3.3 Оформити звіт з роботи.

3.3.4 Відповісти на контрольні питання.

3.4 Зміст звіту

3.4.1 Тема та мета роботи.

3.4.2 Завдання до роботи.

3.4.3 Короткі теоретичні відомості.

3.4.4 Текст розробленої програми.

3.4.5 Копії екрану, що відображають результати виконання лабораторної роботи.

3.4.6 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

3.5 Контрольні запитання

3.5.1 Які модифікатори класів Ви знаєте й у чому їхня суть?

3.5.2 Які модифікатори методів класів Ви знаєте й у чому їхня суть?

3.5.3 Що таке абстрактні класи й для чого вони необхідні?

3.5.4 Для чого служить клас Object в Java?

3.5.5 Які особливості конструктора класу в Java?

3.5.6 Для чого призначені та суть статичних членів класу в Java?

3.5.7 Розкажіть про основні можливості класу Complex в Java.

3.5.8 Які особливості використання методу Main в Java?

3.5.9 Які особливості області видимості змінних в Java?

3.5.10 Вкладення класів, для чого використовується й особливості застосування в java?

4 ЛАБОРАТОРНА РОБОТА № 4 ПАКЕТИ Й ІНТЕРФЕЙСИ

4.1 Мета роботи

Навчитися працювати з пакетами та інтерфейсами в Java.

4.2 Основні теоретичні відомості

У стандартну бібліотеку Java API входять сотні класів. Розроблювачі Java включили в мову додаткову конструкцію – пакети (packages). Всі класи Java розподіляються по пакетах. Крім класів пакети можуть містити в собі інтерфейси й вкладені під пакети (subpackages). Утворюється деревоподібна структура пакетів і підпакетів.

Всі файли з розширенням class (утримуючі байти-коди), що утворюють пакет, зберігаються в одному каталозі файлової системи.

Кожний пакет утворює один простір імен (namespace). Це означає, що всі імена класів, інтерфейсів і підпакетів у пакеті повинні бути унікальні. Імена в різних пакетах можуть збігатися, але це будуть різні програмні одиниці. Таким чином, жоден клас, інтерфейс або підпакет не може бути відразу у двох пакетах. Якщо треба використати два класи з однаковими іменами з різних пакетів, то ім'я класу уточнюється ім'ям пакета: пакет.клас. Таке уточнене ім'я називається повним ім'ям класу (fully qualified name).

Пакетами користуються ще й для того, щоб додати до прав доступу до членів класу private, protected і public ще один, "пакетний" рівень доступу.

Якщо член класу не відзначений жодним з модифікаторів private, protected, public, те, за замовчуванням, до нього здійснюється пакетний доступ (default access), а саме, до такого члена може звернутися будь-який метод будь-якого класу з того ж пакета. Пакети обмежують і доступ до класу цілком - якщо клас не позначений модифікатором public, те всі його члени, навіть відкриті, public, не будуть видні з інших пакетів.

Пакет і підпакет

Щоб створити пакет треба просто в першому рядку Java-файлу з кодом записати рядок package ім'я;, наприклад:

```
package mypack;
```

Тим самим створюється пакет із зазначеним ім'ям `mypack` і всі класи, записані в цьому файлі, потраплять у пакет `mypack`. Повторюючи цей рядок на початку кожного вихідного файлу, включаємо в пакет нові класи.

Ім'я підпакету уточнюється ім'ям пакету. Щоб створити підпакет з ім'ям, наприклад, `subpack`, треба в першому рядку вихідного файлу написати;

```
package mypack.subpack;
```

і всі класи цього файлу й всіх файлів з таким ж першим рядком потраплять у підпакет `subpack` пакета `mypack`.

Можна створити й підпакет підпакета:

```
package mypack.subpack.sub;
```

і т.д. без обмеження.

Оскільки рядок `package ім'я;` тільки один й це обов'язково перший рядок файлу, кожний клас попадає тільки в один пакет або підпакет.

Компілятор Java може сам створити каталог з тим же ім'ям `mypack`, а у ньому підкаталог `subpack`, і розмістити в них class-файли з байтами-кодами.

Повні імена класів `A`, у будуть виглядати так: `mypack.A`, `mypack.subpack.B`.

Компілятор завжди створює для таких класів безіменний пакет(`unnamed package`), якому відповідає поточний каталог(`current working directory`) файлової системи. Тому в нас class-файл завжди виявлявся у тому же каталозі, що й відповідний Java-файл.

Безіменний пакет служить звичайно сховищем невеликих пробних або проміжних класів. Більші проекти краще зберігати в пакетах. Наприклад, бібліотека класів Java 2 API зберігається в пакетах `java`, `javax`, `org.omg`. Пакет Java містить тільки підпакети `applet`, `awt`, `beans`, `io`, `lang`, `math`, `net`, `rmi`, `security`, `sql`, `text`, `util` і жодного класу. Ці пакети мають свої підпакети, наприклад, пакет створення ГПІ і графіки `java.awt` містить підпакети `color`, `datatransfer`, `dnd`, `event`, `font`, `geometry`, `im`, `image`, `print`.

4.2.1 Права доступу до членів класу

З незалежного класу можна звернутися тільки до відкритих, `public` полей класу іншого пакета. З підкласу можна звернутися ще й

до захищених, `protected` полей, але тільки успадкованим безпосередньо, а не через екземпляр суперкласу.

Все зазначене відноситься не тільки до полів, але й до методів. Більш детальні дані про доступ всередині класів в табл. 4.1.

Таблиця 4.1 – Права доступу до полів і методів класу

	Клас	Пакет	Пакет і підкласи	Всі класи
<code>private</code>	+			
<code>"package"</code>	+	+		
<code>protected</code>		+	+	*
<code>public</code>	+	+	+	+

Особливість доступу до `protected`-полів і методів із чужого пакета відзначено зірочкою.

4.2.2 Імпорт класів і пакетів

Правила використання оператора `import`: використовується слово `import` і, через пробіл, повне ім'я класу. Скільки класів треба вказати, стільки операторів `import` і пишеться.

Також використовується друга форма оператора `import` - вказується ім'я пакету або підпакету, а замість короткого імені класу ставиться зірочка *. Цим записом компіляторіві пропонується переглянути весь пакет. Наприклад, можна було написати

```
import pl.*;
```

Нагадаємо, що імпортувати можна тільки відкриті класи, позначені модифікатором `public`.

Пакет `java.lang` проглядається завжди, його необов'язково імпортувати. Інші пакети стандартної бібліотеки треба вказувати в операторах `import`, або записувати повні імена класів.

Оператор `import` не еквівалентний директиві препроцесора `include` – він не підключає ніякі файли.

4.2.3 Інтерфейси

Всі класи походять тільки від класу `object`. Але часто виникає необхідність породити клас від двох класів. Це називається множинним спадкуванням (`multiple inheritance`).

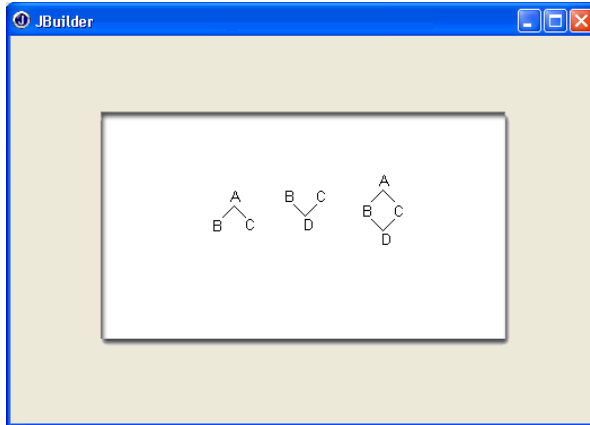


Рисунок 4.1 – Різні варіанти спадкування

Творці мови Java після довгих суперечок і міркувань зробили радикально - заборонили множинне спадкування взагалі. При розширенні класу після слова `extends` можна написати тільки одне ім'я суперкласу. За допомогою уточнення `super` можна звернутися тільки до членів безпосереднього суперкласу.

Але що робити, якщо все-таки при породженні треба використати декілька предків? Наприклад, у нас є загальний клас автомобілів `Automobile`, від якого можна породити клас вантажівок `Truck` і клас легкових автомобілів `Car`. Але треба описати пікап `Pickup`. Цей клас повинен успадковувати властивості й вантажних, і легкових автомобілів.

У таких випадках використовується ще одна конструкція мови Java- інтерфейс. Уважно проаналізувавши ромбовидне спадкування, теоретики ООП з'ясували, що проблему створює тільки реалізація методів, а не їхній опис.

Інтерфейс(`interface`), на відміну від класу, містить тільки константи й заголовки методів, без їхньої реалізації.

Інтерфейси розміщуються в тих же пакетах і підпакетах, що й класи, і компілюються теж в `class`-файли.

Опис інтерфейсу починається зі слова `interface`, перед яким може стояти модифікатор `public`, що означає, як і для класу, що інтерфейс доступний усюди. Якщо ж модифікатора `public` немає, інтерфейс буде доступний тільки у своєму пакеті.

Після слова `interface` записується ім'я інтерфейсу, потім може стояти слово `extends` і список інтерфейсів-предків через кому. Таким чином, інтерфейси можуть породжуватися від інтерфейсів, утворюючи свій, незалежний вид класів, ієрархію, причому в ній допускається множинне спадкування інтерфейсів. У цій ієрархії немає кореня, загального предка.

Потім, у фігурних дужках, записуються в будь-якому порядку константи й заголовки методів. Можна сказати, що в інтерфейсі всі методи абстрактні, але слово `abstract` писати не треба. Константи завжди статичні, але слова `static` і `final` вказувати не потрібно.

Всі константи й методи в інтерфейсах завжди відкриті, не треба навіть вказувати модифікатор `public`.

Таку схему можна запропонувати для ієрархії автомобілів:

```
interface Automobile{... }
interface Car extends Automobile{... }
interface Truck extends Automobile{... }
interface Pickup extends Car, Truck{... }
```

Використовувати потрібно не інтерфейс, а його реалізацію(`implementation`). Реалізація інтерфейсу - це клас, у якому розписуються методи одного або декількох інтерфейсів. У заголовку класу після його імені або після імені його суперкласу, якщо він є, записується слово `implements` і, через кому, перераховуються імена інтерфейсів.

Так можна реалізувати ієрархію автомобілів:

```
interface Automobile{... }
interface Car extends Automobile{... }
class Truck implements Automobile{... }
class Pickup extends Truck implements Car{... }
```

або так:

```
interface Automobile{... }
interface Car extends Automobile{... }
interface Truck extends Automobile{... }
class Pickup implements Car, Truck{... }
```

Реалізація інтерфейсу може бути неповної, деякі методи інтерфейсу є, а інші – не вказані. Така реалізація - абстрактний клас, його обов'язково треба позначити модифікатором `abstract`.

4.2.4 Design patterns

Схема була розроблена ще в 80-х роках XX століття в мові Smalltalk і одержала назву MVG(Model-View-Controller). Проектують її із трьох частин.

Перша частина, назовемо її Контролером(controller), приймає дані від датчиків і перетворює їх у якусь однакову форму, придатну для подальшої обробки, наприклад, приводить до одного масштабу. При цьому для кожного датчика треба написати свій модуль, на вхід якого надходять сигнали конкретного пристрою, а на виході утвориться уніфікована інформація.

Друга частина, назовемо її Моделлю(model), приймає цю уніфіковану інформацію від Контролера, нічого не знаючи про датчик і не цікавлячись тим, від якого саме датчика вона надійшла, і перетворить її по своїх алгоритмах знов-таки до якогось одноманітного виду, наприклад, до послідовності чисел.

Третя частина системи, Вид(view), безпосередньо пов'язана із пристроями виводу й перетворює послідовність, що надійшла від Моделі, чисел у графік, діаграму або пакет для відправлення по мережі. Для кожного пристрою треба написати свій модуль, що враховує особливості саме цього пристрою.

Найпростіша із цих схем. Треба написати клас, у якого можна створити тільки один екземпляр, але цим екземпляром повинні користуватися об'єкти інших класів. Для рішення цього завдання запропонована схема Singleton, представлена в програмі 4.1.

Програма 4.1. Схема Singleton

```
final class Singleton{
    private static Singleton s = new Singleton(0);
    private int k;
    private Singleton(int i){k = i;}
    public static Singleton getReference() {return s;}
    public int getValue(){return k;}
    public void setValue(int i){k = i;}
}

public class SingletonTest {
    public static void main(String[] args){
        Singleton ref = Singleton.getReference();
        System.out.println(ref.getValue());
        ref.setValue(ref.getValue() + 5);
        System.out.println(ref.getValue());
    }
}
```

Клас singleton остаточний - його не можна розширити. Його конструктор закритий - ніякий метод не може створити екземпляр цього класу. Єдиний екземпляр класу singleton - статичний, він створюється усередині класу. Зате будь-який об'єкт може одержати

посилання на екземпляр методом `getReference()`, Змінити стан екземпляра `s` методом `setValue()` або переглянути його поточний стан методом `getValue()`.

4.3 Завдання до роботи

4.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

4.3.2 Виконати наступні завдання:

Загальні завдання:

1. Реалізувати бібліотеку класів, що представляють собою абстракцію організаційної структури підприємства. Розробити наступні класи – «Людина», «Співробітник», «Підрозділ», «Посада», що відповідають наступним вимогам:
 - клас «Людина» повинен володіти як мінімум наступними властивостями – прізвище, ім'я, по батькові, дата народження, стать.
 - клас «Співробітник» повинен розширювати клас «Людина» володіти як мінімум наступними властивостями - підрозділ, посада, зарплата. Для співробітника, повинні бути доступні прийом і звільнення на роботу.
 - клас «Підрозділ» повинен розширювати клас «Співробітник» володіти як мінімум наступними властивостями – назва підрозділу, кількість людей, час роботи.
 - клас «Посада» повинен розширювати клас «Підрозділ» володіти як мінімум наступними властивостями - повна назва посади, перелік обов'язків, список підлеглих. Для неї, повинні бути доступні - віддати розпорядження й викликати співробітника.

Індивідуальні завдання:

1. Створити об'єкт класу Текст, використовуючи клас Абзац. Методи: доповнити текст, вивести на консоль текст, заголовок тексту.
2. Створити об'єкт класу Літак, використовуючи клас Крило. Методи: літати, задавати маршрут, вивести на консоль маршрут.

3. Створити об'єкт класу Планета, використовуючи клас Материк. Методи: вивести на консоль назва материка, планети, кількість материків додавання планети в систему.
4. Створити об'єкт класу Комп'ютер, використовуючи класи Вінчестер, Дискковод, ОЗУ. Методи: включити, виключити, перевірити на віруси, вивести на консоль розмір вінчестера.
5. Створити об'єкт класу Коло, використовуючи класи Крапка, Окружність. Методи: завдання розмірів, зміна радіуса, визначення принад-лежности крапки даному колу.
6. Створити об'єкт класу Квочка, використовуючи класи Птах, Зозуля. Методи: літати, співати, нести яйця, висиджувати пташенят.
7. Створити об'єкт класу Одномірний масив, використовуючи клас Масив. Методи: створити, вивести на консоль, виконати операції (скласти, відняти, перемножити).
8. Створити об'єкт класу Будинок, використовуючи класи Вікно, Двері. Методи: закрити на ключ, вивести на консоль кількість вікон, дверей.
9. Створити об'єкт класу Дерево, використовуючи класи Лист. Методи: зацвісти, обпати листям, покритися інеєм, пожовкнути листям.
10. Створити об'єкт класу Фотоальбом, використовуючи клас Фотографія. Методи: задати назву фотографії, доповнити фотоальбом фотографією, вивести на консоль кількість фотографій.

4.3.3 Оформити звіт з роботи.

4.3.4 Відповісти на контрольні питання.

4.4 Зміст звіту

4.4.1 Тема та мета роботи.

4.4.2 Завдання до роботи.

4.4.3 Короткі теоретичні відомості.

4.4.4 Текст розробленої програми.

4.4.5 Копії екрану, що відображають результати виконання лабораторної роботи.

4.4.6 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

4.5 Контрольні запитання

4.5.1 Що таке пакети в мові Java?

4.5.2 Що таке «інтерфейс» у мові Java?

4.5.3 Для чого використовуються пакети в мові Java?

4.5.4 Які особливості прав доступу до членів класів у мові Java?

4.5.5 Які особливості імпорту класів у мові Java?

4.5.6 Які особливості імпорту пакетів у мові Java?

4.5.7 Які особливості імпорту класів у мові Java?

4.5.8 Які особливості множинного спадкування в мові Java?

4.5.9 Які особливості використання інтерфейсів у мові Java?

4.5.10 Що таке design patterns? І які з них Ви знаєте в мові Java?

5 ЛАБОРАТОРНА РОБОТА № 5 КЛАСИ-ОБОЛОНКИ

5.1 Мета роботи

Вивчити основні принципи роботи із класами-оболонками в Java.

5.2 Основні теоретичні відомості

Java – повністю об'єктно-орієнтовна мова. Це означає, що все, що тільки можна, в Java представлено об'єктами.

Вісім примітивних типів порушують це правило. Вони залишені в Java через багаторічну звичку до чисел і символів. Та й арифметичні дії зручніше й швидше робити зі звичайними числами, а не з об'єктами класів.

Але й для цих типів у мові Java є відповідні класи – класи-оболонки (wrapper) примітивних типів. Звичайно, вони призначені не для обчислень, а для дій, типових при роботі із класами – створення об'єктів, перетворення об'єктів, одержання чисельних значень об'єктів у різних формах і передачі об'єктів у методи по посиланню.

На рис. 5.1 показана одна з галузей ієрархії класів Java.

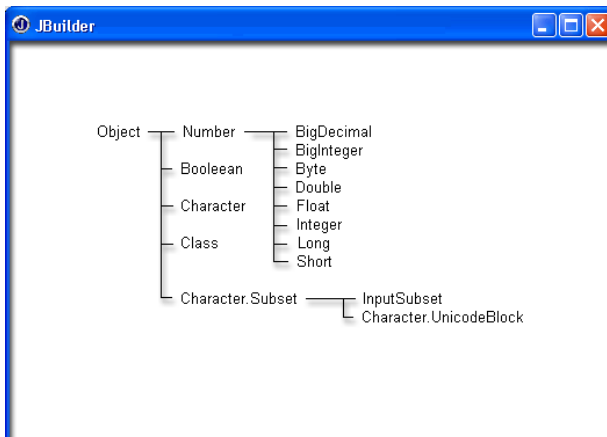


Рисунок 5.1 – Класи примітивних типів

Для кожного примітивного типу є відповідний клас. Числові класи мають загального предка – абстрактний клас `Number`, у якому описані шість методів, що повертають числове значення, що втримується в класі, приведені до відповідного примітивного типу: `byteValue()`, `doubleValue()`, `floatValue()`, `intValue()`, `longValue()`, `shortValue()`. Ці методи перевизначені в кожному з шести числових класів-оболонок.

Крім методу порівняння об'єктів `equals()`, перевизначеного з класу `Object`, всі класи, крім `Boolean` і `Class`, мають метод `compareTo()`, що порівнює числове значення із числовим значенням об'єкта - аргументу методу `compareTo()`.

Числові класи

У кожному із шести числових класів-оболонок є статичні методи перетворення рядка символів типу `String`, який представляє собою число, у відповідний примітивний тип: `Byte.parseByte()`, `Double.parseDouble()`, `Float.parseFloat()`, `Integer.parseInt()`, `Long.parseLong()`, `Short.parseShort()`. Вихідний рядок типу `String` задається як аргумент методу. Ці методи корисні всюди, де числа представляються рядками цифр зі знаками плюс або мінус і десяткова крапка.

5.2.1 Клас Boolean Клас Character

Клас Boolean

Це дуже невеликий клас, призначений головним чином для того, щоб передавати логічні значення в методи по посиланню.

Конструктор `Boolean(String s)` створює об'єкт, що містить значення `true`, якщо рядок `s` дорівнює "true" у будь-якому сполученні регістру букв, і значення `false` - для будь-якого іншого рядку.

Логічний метод `booleanValue()` повертає логічне значення, що зберігається в об'єкті.

Клас Character

У цьому класі зібрані статичні константи й методи для роботи з окремими символами.

Статичний метод
`digit(char ch, in radix)`

переводить цифру `ch` системи числення з основою `radix` у її числове значення типу `int`.

Статичний метод

```
forDigit(int digit, int radix)
```

робить зворотнє перетворення цілого числа digit у відповідну цифру(тип char) у системі числення з основою radix.

Основа системи числення повинне перебувати в діапазоні від Character.MIN_RADIX до Character.MAX_RADIX.

Метод toString() переводить символ, що знаходиться в класі, у рядок з тим же символом.

Статичні методи toLowerCase(), toUpperCase(), toLowerCase() повертають символ, що знаходиться в класі, у зазначеному регістрі. Останній із цих методів призначений для правильного переводу у верхній регістр чотирьох кодів Unicode, що не виражаються одним символом.

Множина статичних логічних методів перевіряють різні характеристики символу, переданого як аргумент:

- isDefined() – з'ясовує, чи визначений символ у кодуванні Unicode;

- isDigit() – перевіряє, чи є символ цифрою Unicode;

- isIdentifierIgnorable() – з'ясовує, чи не можна використати символ в ідентифікаторах;

- isISOControl() – визначає, чи є символ керуючим;

- isJavaIdentifierPart() – з'ясовує, чи можна використати символ в ідентифікаторах;

- isJavaIdentifierStart() – визначає, чи може символ починати ідентифікатор;

- isLetter() – перевіряє, чи є символ буквою Java;

- IsLetterOrDigit() – перевіряє, чи є символ буквою або цифрою Unicode;

- isLowerCase() – визначає, чи записаний символ у нижньому регістрі;

- isSpaceChar() – з'ясовує, чи є символ пробілом у Unicode;

- isTitleCase() – перевіряє, чи є символ титульним;

- isUnicodeIdentifierPart() – з'ясовує, чи можна використати символ в іменах Unicode;

- isUnicodeIdentifierStart() – перевіряє, чи є символ буквою Unicode;

- isUpperCase() – перевіряє, чи записаний символ у верхньому регістрі;

– `isWhitespace()` – з'ясовує, чи є символ пробільним.

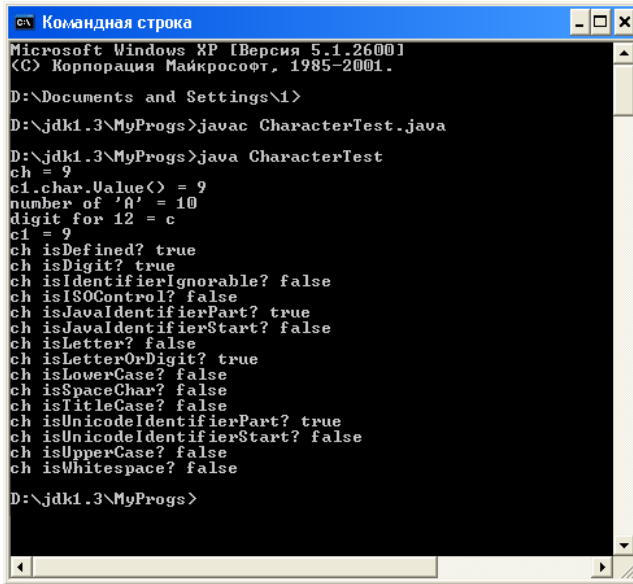
Програма 5.1 демонструє використання цих методів, а на рис.

5.3 показаний вивід цієї програми.

Програма 5.1. Методи класу `Character` у програмі `CharacterTest`

```
class CharacterTest{
    public static void main(String[] args){
        char ch = '9';
        Character cl = new Character(ch);
        System.out.println("ch = " + ch);
        System.out.println("cl.charValue() = " +
            cl.charValue());
        System.out.println("number of 'A' = " +
            Character.digit('A', 16));
        System.out.println("digit for 12 = " +
            Character.forDigit(12, 16));
        System.out.println("cl = " + cl.toString());
        System.out.println("ch isDefined? " +
            Character.isDefined(ch));
        System.out.println("ch isDigit? " +
            Character.isDigit(ch));
        System.out.println("ch isIdentifierIgnorable? " +
            Character.isIdentifierIgnorable(ch));
        System.out.println("ch isISOControl? " +
            Character.isISOControl(ch));
        System.out.println("ch isJavaIdentifierPart? " +
            Character.isJavaIdentifierPart(ch));
        System.out.println("ch isJavaIdentifierStart? " +
            Character.isJavaIdentifierStart(ch));
        System.out.println("ch isLetter? " +
            Character.isLetter(ch));
        System.out.println("ch isLetterOrDigit? " +
            Character.isLetterOrDigit(ch));
        System.out.println("ch isLowerCase? " +
            Character.isLowerCase(ch));
        System.out.println("ch isSpaceChar? " +
            Character.isSpaceChar(ch));
        System.out.println("ch isTitleCase? " +
            Character.isTitleCase(ch));
        System.out.println("ch isUnicodeIdentifierPart? " +
            Character.isUnicodeIdentifierPart(ch));
        System.out.println("ch isUnicodeIdentifierStart? " +
            Character.isUnicodeIdentifierStart(ch));
        System.out.println("ch isUpperCase? " +
            Character.isUpperCase(ch));
        System.out.println("ch isWhitespace? " +
            Character.isWhitespace(ch)); } }
```

У клас `Character` вкладені класи `Subset` і `UnicodeBlock`, причому клас `Unicode` і ще один клас, `inputSubset`, є розширеннями класу `Subset`, як це видно на рис. 5.2. Об'єкти цього класу містять підмножини `Unicode`.



```

Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

D:\Documents and Settings\1>
D:\jdk1.3\MyProgs>javac CharacterTest.java
D:\jdk1.3\MyProgs>java CharacterTest
ch = 9
c1.char.Value() = 9
number of 'A' = 10
digit for 12 = c
c1 = 9
ch isDefined? true
ch isDigit? true
ch isIdentifierIgnorable? false
ch isISOControl? false
ch isJavaIdentifierPart? true
ch isJavaIdentifierStart? false
ch isLetter? false
ch isLetterOrDigit? true
ch isLowerCase? false
ch isSpaceChar? false
ch isTitleCase? false
ch isUnicodeIdentifierPart? true
ch isUnicodeIdentifierStart? false
ch isUpperCase? false
ch isWhitespace? false

D:\jdk1.3\MyProgs>

```

Рисунок 5.2 – Методи класу Character у програмі CharacterTest

Разом із класами-оболонками зручно розглянути два класи для роботи з будь-якими числами.

5.2.2 Клас BigInteger

Всі примітивні цілі типи мають обмежений діапазон значень. У цілочисельній арифметиці Java немає переповнення, цілі числа приводяться по модулю, рівному діапазону значень.

Для того щоб було можна робити цілочисельні обчислення з будь-якою розрядністю, до складу Java API введений клас BigInteger, що зберігається в пакеті java.math. Цей клас розширює клас Number, отже, у ньому перевизначені методи doubleValue(), floatValue(), intValue(), longValue(). Методи byteValue() і shortValue() не перевизначені, а прямо успадковуються від класу Number.

Дії з об'єктами класу BigInteger не приводять ні до переповнення, ні до приведення по модулю. Якщо результат операції великий, то число розрядів просто збільшується. Числа зберігаються у двійковій формі з додатковим кодом.

Перед виконанням операції числа вирівнюються по довжині доповненням знакового розряду.

Шість конструкторів класу створюють об'єкт класу `BigDecimal` з рядка символів(знака числа й цифр) або з масиву байтів.

Дві константи - `ZERO` і `ONE` - моделюють нуль і одиницю в операціях з об'єктами класу `BigInteger`.

Метод `toByteArray()` перетворює об'єкт у масив байтів.

Більшість методів класу `BigInteger` моделюють цілочисельні операції й функції, повертаючи об'єкт класу `BigInteger`:

– `abs()` – повертає об'єкт, що містить абсолютне значення числа, що зберігається в даному об'єкті `this`;

– `add(x)` – операція `this + x` ;

– `and(x)` – операція `this & x` ;

– `andNot(x)` – операція `this & (~x)` ;

– `divide(x)` – операція `this / x` ;

– `divideAndRemainder(x)` – повертає масив із двох об'єктів класу `BigInteger`, що містять частку й залишок від розподілу `this` на `x` ;

– `gcd(x)` – найбільший загальний дільник, абсолютних, значень об'єкта `this` і аргументу `x` ;

– `max(x)` – найбільше зі значень об'єкта `this` і аргументу `x` ;

– `min(x)` – найменше зі значень об'єкта `this` і аргументу `x` ;

– `mod(x)` – залишок від розподілу об'єкта `this` на аргумент методу `x` ;

– `modInverse(x)` – залишок від розподілу числа, зворотного об'єкту `this`, на аргумент `x` ;

– `modPow(n, m)` – залишок від розподілу об'єкта `this`, зведеного в ступінь `n`, на `m` ;

– `multiply(x)` – операція `this * x` ;

– `negate()` – зміна знака числа, що зберігається в об'єкті;

– `not()` – операція `~this` ;

– `or(x)` – операція `this | x` ;

– `pow(n)` – операція зведення числа, що зберігається в об'єкті, у ступінь `n` ;

– `remainder(x)` – операція `this % x` ;

– `shiftLeft(n)` – операція `this « n` ;

– `shiftRight(n)` – операція `this » n` ;

– `signum()` – функція `sign(x)` ;

– `subtract(x)` – операція `this - x` ;

– xor(x) – операція $\text{this} \wedge x$.

У програмі 5.3 наведені приклади використання даних методів, а рис. 5.3 показує результати виконання цієї програми.

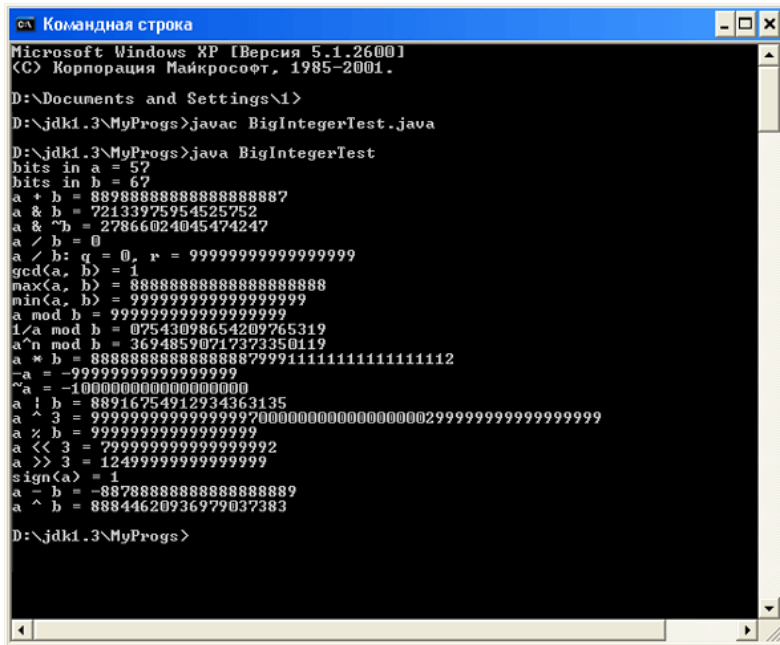


Рисунок 5.3 – Методи класу BigInteger у програмі BigIntegerTest

Програма 5.2. Методи класу BigInteger у програмі BigIntegerTest

```
import java.math.BigInteger;
class BigIntegerTest {
    public static void main(String[] args) {
        BigInteger a = new BigInteger("9999999999999999999");
        BigInteger b = new BigInteger("88888888888888888888888888888888");
        System.out.println("bits in a = " + a.bitLength());
        System.out.println("bits in b = " + b.bitLength());
        System.out.println("a + b = " + a.add(b));
        System.out.println("a & b = " + a.and(b));
        System.out.println("a & ~b = " + a.andNot(b));
        System.out.println("a / b = " + a.divide(b));
        BigInteger[] r = a.divideAndRemainder(b);
        System.out.println("a / b: q = " + r[0] + ", r = " + r[1]);
        System.out.println("gcd(a, b) = " + a.gcd(b));
    }
}
```

```

System.out.println("max(a, b) = " + a.max(b));
System.out.println("min(a, b) = " + a.min(b));
System.out.println("a mod b = " + a.mod(b));
System.out.println("1/a mod b = " + a.modInverse(b));
System.out.println("a mod b = " + a.modPow(a, b));
System.out.println("a * b = " + a.multiply(b));
System.out.println("-a = " + a.negate());
System.out.println("~a = " + a.not());
System.out.println("a | b = " + a.or(b));
System.out.println("a л 3 = " + a.pow(3));
System.out.println("a % b = " + a.remainder(b));
System.out.println("a « 3 = " + a.shiftLeft(3));
System.out.println("a » 3 = " + a.shiftRight(3));
System.out.println("sign(a) = " + a.signum());
System.out.println("a - b = " + a.subtract(b));
System.out.println("a л b = " + a.xor(b));
}
}

```

Зверніть увагу на те, що в програму 5.2 треба імпортувати пакет `Java.math`.

5.2.3 Клас `Big Decimal`

Клас `BigDecimal` розташований у пакеті `java.math`.

Кожний об'єкт цього класу зберігає два цілочисельних значення: мантису дійсного числа у вигляді об'єкту класу `BigInteger`, і ненегативний десятковий порядок числа типу `int`.

Наприклад, для числа 76.34862 буде зберігатися мантиса 7634862 в об'єкті класу `BigInteger`, і порядок 5 як ціле число типу `int`. Таким чином, мантиса може містити будь-яку кількість цифр, а порядок обмежений значенням константи `integer.MAX_VALUE`. Результат операції над об'єктами класу `BigDecimal` округляється по одному з восьми правил, обумовлених наступними статичними цілими константами:

- `ROUND_CEILING` – округлення убік більшого цілого;
- `ROUND_DOWN` – округлення до нуля, до меншого по модулю цілого значенню;
- `ROUND_FLOOR` – округлення до меншого цілого;
- `ROUND_HALF_DOWN` – округлення до найближчого цілого, середнє значення округляється до меншого цілого;
- `ROUND_HALF_EVEN` – округлення до найближчого цілого, середнє значення округляється до парного числа;

- `ROOND_HALF_UP` – округлення до найближчого цілого, середнє значення округляється до більшого цілого;

- `ROUND_UNNECESSARY` – припускається, що результат буде цілим, і округлення не знадобиться;

- `ROUND_UP` – округлення від нуля, до більшого по модулю цілого значенню.

У класі `BigDecimal` чотири конструктори:

- `BigDecimal(BigInteger bi)` – об'єкт буде зберігати велике ціле `bi`, порядок дорівнює нулю;

- `BigDecimal(BigInteger mantissa, int scale)` – задається мантиса `mantissa` і ненегативний порядок `scale` об'єкта; якщо порядок `scale` негatifний, виникає виняткова ситуація;

- `BigDecimal(double d)` – об'єкт буде містити речовинне число подвоєної точності `d`; якщо значення `d` нескінченно або `Na`, те виникає виняткова ситуація;

- `BigDecimal(String val)` – число задається рядком символів `val`, де повинен бути запис числа за правилами мови Java.

При використанні третього з перерахованих конструкторів виникає неприємна особливість, відзначена в документації. Оскільки речовинне число при перетворенні у двійкову форму представляється, як правило, нескінченним двійковим дробом, то при створенні об'єкта, наприклад, `BigDecimal(0.1)`, мантиса, що зберігається в об'єкті, виявиться дуже великою. Вона показана на рис. 4.5. Але при створенні такого ж об'єкта четвертим конструктором, `BigDecimal("0.1")`, мантиса буде дорівнює просто 1.

У Класі перевизначені методи `doubleValue()`, `floatValue()`, `intValue()`, `longValue()`.

Більшість методів цього класу моделюють операції з дійсними числами. Вони повертають об'єкт класу `BigDecimal`. Тут буква `x` позначає об'єкт класу `BigDecimal`, буква `n` - ціле значення типу `int`, буква `r` – спосіб округлення, одну з восьми перерахованих вище констант:

- `abs()` – абсолютне значення об'єкта `this` ;

- `add(x)` – операція `this + x` ;

- `divide(x, r)` – операція `this / x` з округленням за способом `r` ;

- `divide(x, n, r)` – операція `this / x` зі зміною порядку й округленням за способом `r` ;

- `max(x)` – найбільше з `this` і `x` ;

- `min(x)` – найменше з `this` і `x` ;
- `movePointLeft(n)` – зсув вліво на `n` розрядів;
- `movePointRight(n)` – зсув вправо на `n` розрядів;
- `multiply(x)` – операція `this * x` ;
- `negate()` – повертає об'єкт зі зворотним знаком;
- `scale()` – повертає порядок чисел;
- `setScale(n)` – встановлює новий порядок `n` ;
- `setScale(n, r)` – встановлює новий порядок `n` і округляє число

при необхідності за способом `r` ;

- `signum()` – знак числа, що зберігається в об'єкті;
- `subtract(x)` – операція `this - x` ;
- `toBigInteger()` – округлення числа, що зберігається в об'єкті;
- `unscaledValue()` – повертає мантису числа.

Програма 5.3 показує приклади використання цих методів, а рис. 5.4 – вивід результатів.

```

Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

D:\Documents and Settings\I>
D:\jdk1.3\MyProgs>javac BigDecimalTest.java

D:\jdk1.3\MyProgs>java BigDecimalTest
|x| = -12345.67890123456789
x + y = -12345.64432227456789
x / y = -357028.63536770822170
x \ y = -357028.635368
max(x, y) = 0.03457896
min(x, y) = -12345.67890123456789
x << 3 = -12.34567890123456789
x >> 3 = -12345678.90123456789
x*y = -426.9007368986340736855944
-x = 12345.67890123456789
scale of x = 14
increase scale of x to 20 = -12345.6789012345678900000000
decrease scale of x to 10 = -12345.6789012346
sign(x) = -1
x - y = -12345.71348019456789
round x = -12345
mantissa of x = -1234567890123456789
mantissa of 0.1 =
= 1000000000000000000055511151231257827021181583404541015625

D:\jdk1.3\MyProgs>
  
```

Рисунок 5.4 – Методи класу `BigDecimal` у програмі `BigDecimalTest`

Програма 5.3. Методи класу BigDecimal У програмі BigDecimalTest

```
import java.math.*;
class BigDecimalTest{
    public static void main,( String [] args) {
        BigDecimal x = new BigDecimal("-12345.67890123456789");
        BigDecimal y = new BigDecimal("345.7896e-4");
        BigDecimal z = new BigDecimal(new BigInteger("123456789"),8);
        System.out.println("|x| = " + x.abs());
        System.out.println("x + y = " + x.add(y));
        System.out.println("x / y = " + x.divide(y, BigDecimal.ROUND__DOWN));
        System.out.println("x/y = " +
            x.divide(y, 6, BigDecimal.ROUND_HALF_EVEN));
        System.out.println("max(x, y) = " + x.max(y));
        System.out.println("min(x, y) = " + x.min(y));
        System.out.println("x « 3 = " * x.movePointLeft(3));
        System.out.println("x » 3 = " + x.mpoqePqintRight(3));
        System.out.println("x * y = " + x.multiply(y));
        System.out.println("-x = " + x.negate());
        System.out.println("scale of x = " + x.scale());
        System.out.println("increase scale of x to 20 = " + x.setScale(20));
        System.out.println("decrease scale of x to 10 = " +
            x.setScale(10, BigDecimal.ROUND_HALF_UP));
        System.out.println("sign(x) = " + x.signum());
        System.out.println("x - y = " + x.subtract(y));
        System.out.println("round x = " + x.toBigInteger());
        System.out.println("mantissa of x = " + x.unscaledValue());
        System.out.println("mantissa of 0.1 =\n= " +
            new BigDecimal(0.1).unscaledValue()); } }
```

5.3 Завдання до роботи

5.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

5.3.2 Вивчити основні принципи роботи Java з класами-оболонками.

5.3.3 Виконати наступні завдання:

1. $\sin(x/n)$, $\tan(x/n)$ для заданого діапазону у вигляді значень одного з наступних типів: double, int. Тип значення, що повертає, повинен збігатися з типом переданих параметрів. Обчислення зробити з точністю в 15 знаків.
2. $\cos(x/n)$, $\tan(x/n)$ для заданого діапазону у вигляді значень одного з наступних типів: long, float. Тип значення, що

повертає, повинен збігатися з типом переданих параметрів. Обчислення зробити з точністю в 15 знаків.

5.3.4 Оформити звіт з роботи.

5.3.5 Відповісти на контрольні питання.

5.4 Зміст звіту

5.4.1 Тема та мета роботи.

5.4.2 Завдання до роботи.

5.4.3 Короткі теоретичні відомості.

5.4.4 Текст розробленої програми.

5.4.5 Копії екрану, що відображають результати виконання лабораторної роботи.

5.4.6 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

5.5 Контрольні запитання

5.5.1 Що таке клас-оболонка?

5.5.2 Для чого потрібні класи-оболонки?

5.5.3 Основні особливості їхнього використання?

5.5.4 Основне застосування класу Boolean?

5.5.5 Основне застосування класу Character?

5.5.6 Назвіть основні методи класу Character.

5.5.7 Особливості застосування класу BigInteger?

5.5.8 Назвіть основні методи класу BigInteger.

5.5.9 Особливості застосування класу BigDecimal?

5.5.10 Назвіть основні методи класу BigDecimal.

6 ЛАБОРАТОРНА РОБОТА № 6 РОБОТА З РЯДКАМИ

6.1 Мета роботи

Навчитися основним принципам роботи з рядками в Java.

6.2 Основні теоретичні відомості

Дуже велике місце в обробці інформації займає робота з текстами. Як і багато чого іншого, текстові рядки в мові Java є об'єктами. Вони представляються екземплярами класу `string` або класу `stringBuffer`.

В об'єктах класу `string` зберігаються рядки-константи незмінної довжини й змісту. Це значно прискорює обробку рядків і дозволяє заощаджувати пам'ять, розділяючи рядок між об'єктами, що використовують їх. Довжину рядків, що зберігаються в об'єктах класу `stringBuffer`, можна міняти, вставляючи й додаючи рядки й символи, видаляючи підстроки або зчіплюючи кілька рядків в один рядок. У багатьох випадках, коли треба змінити довжину рядку типу `string`, компілятор Java неявно перетворить її до типу `stringBuffer`, міняє довжину, потім перетворить назад у тип `string`. Наприклад, що впливає дія

```
String s = "Це" + " одна " + "рядок";
```

компілятор виконає так:

```
String s=new StringBuffer().append("Це").append(" одна ").append("рядок").toString();
```

Буде створений об'єкт класу `stringBuffer`, у нього послідовно додані рядки "Це", " одна ", "рядок", і об'єкт, що вийшов, класу `StringBuffer` буде наведений до типу `String` методом `toString()`.

Перед роботою з рядком його варто створити. Це можна зробити різними способами.

Найпростіший спосіб створити рядок - це організувати посилання типу `string` на рядок-константу:

```
String s1 = "Це рядок.";
```

Якщо константа довга, можна записати її в декількох рядках текстового редактора, зв'язуючи їх операцією зчеплення:

```
String s2 = "Це довгий рядок, " +  
"записана у двох рядках вихідного тексту";
```

Самий правильний спосіб створити об'єкт із погляду ООП - це викликати його конструктор в операції new. Клас string надає вам дев'ять конструкторів:

- string() - створюється об'єкт із порожнім рядком;
- string(String str) – з одного об'єкту створюється інший, тому цей конструктор використовується рідко;
- string(StringBuffer str) – перетворена копія об'єкта класу StringBuffer;
- string(byte[] byteArray) – об'єкт створюється з масиву байтів byteArray;
- string(char[] charArray) – об'єкт створюється з масиву charArray символів Unicode;
- string(byte[] byteArray, int offset, int count) – об'єкт створюється із частини масиву байтів byteArray, що починається з індексу offset і містить count байтів;
- string(char[] charArray, int offset, int count) – те ж, але масив складається із символів Unicode;
- string(byte[] byteArray, String encoding) – символи, записані в масиві байтів, задаються в Unicode-рядку, з урахуванням кодування encoding ;
- string(byte[] byteArray, int offset, int count, String encoding) – те ж саме, але тільки для частини масиву.

Крім того, з масиву символів c[] створюється рядок s1, з масиву байтів, записаного в кодуванні CP866, створюється рядок s2. Нарешті, створюється посилання s3 на рядок-константу.

Програма 6.1. Створення кириличних рядків

```
class StringTest{
    public static void main(String[] args){
        String winLikeWin = null, winLikeDOS = null, winLikeUNIX = null;
        String dosLikeWin = null, dosLikeDOS = null, dosLikeUNIX = null;
        String unixLikeWin = null, unixLikeDOS = null, unixLikeUNIX = null;
        String msg = null;
        byte[] byteCp1251 = {
            (byte) 0x0, (byte) 0xEE, (byte) 0xF1,
            (byte) 0xF1, (byte) 0xES, (byte) 0xFF
        };
        byte[] byteCp866 = {
            (byte) 0x90, (byte) 0xAE, (byte) 0x1,
            (byte) 0xE1, (byte) 0x8, (byte) 0xEF
        };
        byte[] byteKOISR = (
            (byte) 0x2, (byte) 0xCF, (byte) 0x3,
```

```

(byte) 0x3, (byte) 0x9, (byte) 0xD1
};
char[] c = {'P', 'o', 'c', 'i', 'я', 'я'};
String s1 = new String(c);
String s2 = new String(byteCp866);    // Для консолі MS Windows
String s3 = "Россия";
System.out.println();
try{
    // Повідомлення в Cp866 для виводу на консоль MS Windows.
    msg = new String("\\"Pociя\\" в ".getBytes("Cp866"), "Cp1251");
    winLikeWin = new String(byteCp1251, "Cp1251"); //Правильно
    winLikeDOS = new String(byteCp1251, "Cp866");
    winLikeUNIX = new String(byteCp1251, "KOI8-R");
    dosLikeWin = new String(byteCp866, "Cp1251"); // Для консолі
    dosLikeDOS = new String(byteCp866, "Cp866"); // Правильно
    dosLikeUNIX = new String(byteCp866, "KOI8-R");
    unixLikeWin = new String(byteKOISR, "Cp1251");
    unixLikeDOS = new String(byteKOISR, "Cp866");
    unixLikeUNIX = new String(byteKOISR, "KOI8-R"); // Правильно
    System.out.print(msg + "Cp1251: ");
    System.out.write(byteCp1251);
    System.out.println();
    System.out.print(msg + "Cp866: ");
    System.out.write(byteCp866);
    System.out.println();
    System.out.print(msg + "KOI8-R: ");
    System.out.write(byteKOISR);
} catch (Exception e) {
    e.printStackTrace();
}
System.out.println();
System.out.println();
System.out.println(msg + "char array      : " + s1);
System.out.println(msg + "default encoding: " + s2);
System.out.println(msg + "string constant : " + s3);
System.out.println();
System.out.println(msg + "Cp1251 -> Cp1251: " + winLikeWin);
System.out.println(msg + "Cp1251 -> Cp866: " + winLikeDOS);
System.out.println(msg + "Cp1251 -> KOI8-R: " + winLikeUNIX);
System.out.println(msg + "Cp866 -> Cp1251: " + dosLikeWin);
System.out.println(msg + "Cp866 -> Cp866: " + dosLikeDOS);
System.out.println(msg + "Cp866 -> KOI8-R: " + dosLikeUNIX);
System.out.println(msg + "KOI8-R -> Cp1251: " + unixLikeWin);
System.out.println(msg + "KOI8-R -> Cp866: " + unixLikeDOS);
System.out.println(msg + "KOI8-R -> KOI8-R: " + unixLikeUNIX);
}
}

```

Всі ці дані виводяться на консоль MS Windows 2000, як показано на рис. 6.1.

другого рядків. Його можна застосовувати й до констант, і до змінних. Наприклад:

```
String attention = "Увага: ";
String s = attention + "невідомий символ";
```

Друга операція - присвоювання += - застосовується до змінної в лівій частині:

```
attention += s;
```

Маніпуляції рядками

Для того щоб довідатися довжину рядка, тобто кількість символів у ній, треба звернутися до методу length():

```
String s = "Write once, run anywhere.";
int len = s.length();
```

або ще простіше

```
int len = "Write once, run anywhere.".length();
```

оскільки рядок-константа - повноцінний об'єкт класу string.

Помітьте, що рядок - це не масив, у нього немає поля length.

Вибрати символ з індексом ind(індекс першого символу дорівнює нулю) можна методом charAt(int ind) Якщо індекс ind від'ємний або не менше ніж довжина рядка, виникає виняткова ситуація. Наприклад, після визначення

```
char ch = s.charAt(3);
```

змінна ch буде мати значення 't'

Всі символи рядка у вигляді масиву символів можна одержати методом

```
toCharArray(); // що повертає масив символів.
```

Якщо ж треба включити в масив символів dst, починаючи з індексу ind масиву підстроку від індексу begin включно до індексу end винятково, то використайте метод getChars(int begin, int end, char[] dst, int ind) типу void.

У масив буде записане end - begin символів, які займуть елементи масиву, починаючи з індексу ind до індексу ind + (end - begin) - 1.

Якщо треба одержати масив байтів, що містить всі символи рядка в байтовому кодуванні ASCII, то використовуйте метод getBytes().

Метод substring(int begin, int end) виділяє підстроку від символу з індексом begin включно до символу з індексом end виключно. Довжина підстроки буде дорівнює end - begin.

Метод substring(int begin) виділяє підстроку від індексу begin включно до кінця рядка.

Якщо індекси від'ємні, індекс `end` більше довжини рядка або `begin` більше чим `end`, то виникає виняткова ситуація.

Наприклад, після виконання

```
String s = "Write once, run anywhere.";
String sub1 = s.substring(6, 10);
String sub2 = s.substring(16);
```

одержимо в рядку `sub1` значення "once", а в `sub2` - значення "anywhere".

Операція порівняння `==` порівнює тільки посилання на рядки. Вона з'ясовує, чи вказують посилання на той самий рядок. Наприклад, для рядків

```
String s1 = "Якийсь рядок";
String s2 = "Інший рядок";
```

порівняння `s1 == s2` дає в результаті `false`.

Значення `true` вийде, тільки якщо обидві посилання вказують на той самий рядок, наприклад, після присвоювання `s1 = s2`.

Цікаво, що якщо ми визначимо `s2` так:

```
String s2 = "Якийсь рядок";
```

то порівняння `s1 == s2` дасть у результаті `true`, тому що компілятор створить тільки один екземпляр константи "Якийсь рядок" і направить на нього всі посилання.

Логічний метод `equals(object obj)`, перевизначений із класу `object`, повертає `true`, якщо аргумент `obj` не дорівнює `null`, та є об'єктом класу `string`, та рядок повністю ідентичний даному рядку аж до збігу регістра букв. В інших випадках вертається значення `false`.

Логічний метод `equalsIgnoreCase(object obj)` працює так само, але однакові букви, записані в різних регістрах, уважаються співпадаючими.

Метод `compareTo(string str)` повертає ціле число типу `int`, обчислене за наступними правилами:

Рівняються символи даного рядка `this` і рядку `str` з однаковим індексом, поки не зустрінуться різні символи з індексом, допустимо `k`, або поки один з рядків не закінчиться.

У першому випадку повертається значення `this.charAt(k) - str.charAt(k)`, тобто різниця кодувань Unicode перших незбіжних символів. У другому випадку вертається значення `this.length() - str.length()`, тобто різниця довжин рядків. Якщо рядки співпадають, вертається 0. Якщо значення `str` дорівнює `null`, виникає виняткова ситуація. Нуль повертається в тій же ситуації, у якій метод `equals()` повертає `true`.

Метод `compareToIgnoreCase(string str)` робить порівняння без врахування регістру букв, точніше кажучи, виконується метод

```
this.toUpperCase().toLowerCase().compareTo(
    str.toUpperCase().toLowerCase());
```

Ще один метод- `compareTo(Object obj)` створює виняткову ситуацію, якщо `obj` не є рядком. В іншому випадку він працює як метод `compareTo(String str)`.

Ці методи не враховують алфавітне розташування символів у локальному кодуванні.

Зрівняти підстроку даного рядку `this` з підстрокою тієї ж довжини `len` іншого рядка `str` можна логічним методом

```
regionMatches(int ind1, String str, int ind2, int len)
```

Тут `ind1` - індекс початку підстроки даного рядку `this`, `ind2` - індекс початку підстроки іншого рядку `str`. Результат `false` виходить у наступних випадках:

- хоча б один з індексів `ind1` або `ind2` від'ємний;
- хоча б одне з `ind1 + len` або `ind2 + len` більше довжини відповідного рядка;
- хоча б одна пара символів не збігається.

Цей метод розрізняє символи, записані в різних регістрах. Якщо треба порівнювати підстроки без врахування регістру букв, то використовуйте логічний метод:

```
regionMatches(boolean flag, int ind1, String str, int ind2, int len)
```

Якщо перший параметр `flag` дорівнює `true`, то регістр букв при порівнянні підстрок не враховується, якщо `false` - враховується.

Пошук завжди ведеться з врахуванням регістра букв.

Перша поява символу `ch` у даному рядку `this` можна відстежити методом `indexOf(int ch)`, що повертає індекс цього символу в рядку або `-1`, якщо символу `ch` у рядку `this` немає.

Наприклад, "Молоко", `indexOf('o')` видасть у результаті `1`.

Звичайно, цей метод виконує в циклі послідовні порівняння `this.charAt(k++) == ch`, поки не одержить значення `true`.

Друга й наступна появи символу `ch` у даному рядку `this` можна відстежити методом `indexOf(int ch, int ind)`.

Цей метод починає пошук символу `ch` з індексу `ind`. Якщо `ind < 0`, то пошук іде з початку рядка, якщо `ind` більше довжини рядка, то символ не шукається, тобто вертається `-1`.

Наприклад, "молоко".`indexOf('o', indexOf('o') + 1)` дасть у результаті `3`.

Остання поява символу `ch` у даному рядку `this` відслідковує метод `lastIndexOf(int ch)`. Він переглядає рядок у зворотному порядку. Якщо символ `ch` не знайдений, вертається `-1`.

Наприклад, `"Молоко".lastIndexOf('o')` дасть у результаті `5`.

Передостання й попередня появи символу `ch` у даному рядку `this` можна відстежити методом `lastIndexOf(int ch, int ind)`, що переглядає рядок у зворотному порядку, починаючи з індексу `ind`.

Якщо `ind` більше довжини рядка, то пошук іде від кінця рядка, якщо `ind < 0`, те вертається `-1`.

Перше входження підстроки `sub` у даний рядок `this` відшукує метод `indexOf(String sub)`. Він повертає індекс першого символу першого входження підстроки `sub` у рядок або `-1`, якщо підстрока `sub` не входить у рядок `this`. Наприклад, `"Розфарбування".indexOf("роз")` дасть у результаті `4`.

Останнє входження підстроки `sub` у даний рядок `this` можна відшукати методом `lastIndexOf(string sub)`, що повертає індекс першого символу останнього входження підстроки `sub` у рядок `this` або `(-1)`, якщо підстрока `sub` не входить у рядок `this`.

Останнє входження підстроки `sub` не у весь рядок `this`, а тільки в її початок до індексу `ind` можна відшукати методом `lastIndexOf(String stf, int ind)`. Якщо `ind` більше довжини рядка, то пошук іде від кінця рядка, якщо `ind < 0`, те вертається `-1`.

Для того щоб перевірити, чи не починається даний рядок `this` з підстроки `sub`, треба використовувати логічний метод `startsWith(string sub)`, що повертає `true`, якщо даний рядок `this` починається з підстроки `sub`, або збігається з нею, або підстрока `sub` порожня.

Можна перевірити й появу підстроки `sub` у даному рядку `this`, починаючи з деякого індексу `ind` логічним методом `startsWith(String sub, int ind)`. Якщо індекс `ind` негативний або більше довжини рядка, вертається `false`.

Для того щоб перевірити, чи не закінчується даний рядок `this` підстрокою `sub`, треба використовувати логічний метод `endsWith(String sub)`. Він повертає `true`, якщо підстрока `sub` збігається з усім рядком або підстрока `sub` порожня.

Наприклад, `if(fileName.endsWith(".Java"))` відстежить імена файлів з вихідними текстами `Java`.

Перелічені вище методи створюють виняткову ситуацію, якщо `sub == null`.

Якщо треба здійснити пошук, що не враховує регістр букв, треба змінити попередньо регістр всіх символів рядка.

Змінити локаль, можна використовуючи методи `toLowerCase(Locale loc)` і `toUpperCase(Locale loc)`.

Метод `replace(int old, int new)` повертає новий рядок, у якому всі входження символу `old` замінені символом `new`. Якщо символу `old` у рядку немає, то повертається посилання на вихідний рядок.

Наприклад, після виконання "Рука в руку сує хліб", `replace('y', 'e')` одержимо рядок "Ріка в ріці сее хліб".

Регістр букв при заміні враховується.

Метод `trim()` повертає новий рядок, у якому вилучені початкові й кінцеві символи з кодами, що не перевищують `'\u0020'`.

Клас `string` містить вісім статичних методів `valueOf(type elem)` перетворення в рядок примітивних типів `boolean`, `char`, `int`, `long`, `float`, `double`, масиву `char[]`, і просто об'єкта типу `object`.

Дев'ятий метод `valueOf(char[] ch, int offset, int len)` перетворить у рядок підмасив масиву `ch`, що починається з індексу `offset` і має `len` елементів.

6.3 Завдання до роботи

6.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

6.3.2 Вивчити основні принципи роботи Java з рядками.

6.3.3 Виконати наступні завдання:

Загальні завдання:

1. Вивести букви російського алфавіту(у будь-якому регістрі) і їхню кількість.
2. Вивести для кожної букви код юнікоду в десятковому (приведення до `int`) і шестнадцятиричному поданні(16-ичное число можна вивести у зворотному порядку)
3. Написати 2 функції перетворення букв до верхнього й нижнього регістру (`toUpperCase`, `toLowerCase`); написати функцію, що використовує дві попередні, яка на вході приймає букву й повертає її в протилежному регістрі (тобто малу літеру робить прописною і навпаки).

Індивідуальні завдання:

1. Ввести n рядків з консолі, знайти самий короткий й самий довгий рядок. Вивести знайдені рядки і їхню довжину.
2. Ввести n рядків з консолі. Упорядкувати й вивести рядки в порядку зростання (убування) значень їхньої довжини.
3. Ввести n рядків з консолі. Вивести на консоль ті рядки, довжина яких менше (більше) середньої, а також довжину.
4. Ввести n слів з консолі. Знайти слово, у якому число різних символів мінімально. Якщо таких слів декілька, знайти перше з них.
5. Ввести n слів з консолі. Знайти кількість слів, що містять тільки символи латинського алфавіту, а серед них - кількість слів з рівним числом голосних і приголосних букв.
6. Ввести n слів з консолі. Знайти слово, символи в якому йдуть у строгому порядку зростання їхніх кодів. Якщо таких слів декілька, знайти перше з них.
7. Ввести n слів з консолі. Знайти слово, що складається тільки з різних символів. Якщо таких слів декілька, знайти перше з них.
8. Ввести n слів з консолі. Серед слів, що складаються тільки із цифр, знайти слово-паліндром. Якщо таких слів більше одного, знайти друге з них.
9. Використовуючи оператор `switch`, написати програму, що виводить на екран повідомлення о приналежності деякого значення k інтервалам $(-10k, 5]$, $[0, 10]$, $[5, 15]$, $[10, 10k]$.
10. Ввести число від 1 до 12. Вивести на консоль назву місяця, що відповідає даному числу. (Здійснити перевірку коректності введення чисел).

6.3.4 Оформити звіт з роботи.

6.3.5 Відповісти на контрольні питання.

6.4 Зміст звіту

6.4.1 Тема та мета роботи.

6.4.2 Завдання до роботи.

6.4.3 Короткі теоретичні відомості.

6.4.4 Текст розробленої програми.

6.4.5 Результати виконання роботи.

6.4.6 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

6.5 Контрольні запитання

6.5.1 Назвіть особливості роботи з рядками в мові Java.

6.5.2 Для чого призначений клас String?

6.5.3 Назвіть основні методи класу String?

6.5.4 Особливості використання класу String?

6.5.5 Які особливості роботи Java з різними кодуваннями Ви знаєте?

6.5.6 Які особливості роботи з рядками в мові Java?

6.5.7 Для чого призначений клас StringBuffer?

6.5.8 Які основні методи класу StringBuffer Ви знаєте?

6.5.9 У чому відмінності класу String від StringBuffer?

6.5.10 Які особливості зчеплення рядків і маніпуляції ними, у мові Java Ви знаєте?

7 ЛАБОРАТОРНА РОБОТА № 7 КЛАСИ-КОЛЕКЦІЇ

7.1 Мета роботи

Навчитися основним принципам роботи із класами-колекціями в Java.

7.2 Основні теоретичні відомості

7.2.1 Клас Vector

У мові Java з найперших версій є клас `vector`, призначений для зберігання різного числа елементів самого типу `object`.

Клас `vector` з пакету `java.util` і зберігає елементи типу `object`, тобто будь-якого типу. Кількість елементів може бути будь-якою і наперед не визначатися. Елементи одержують індекси 0, 1, 2,... До кожного елементу вектору можна звернутися по індексу, як і до елементу масиву.

Крім кількості елементів, розмір (`size`) вектору, є ще розмір буферу – ємність (`capacity`) вектору. Звичайно ємність збігається з розміром вектору, але можна її збільшити методом `ensureCapacity(int minCapacity)` або зрівняти з розміром вектора методом `trimToSize()`.

У класі чотири конструктори:

```
vector();  
Vector(int capacity);  
vector(int capacity, int increment);  
vector(Collection c);
```

Метод `add(Object element)` дозволяє додати елемент у кінець вектора(те ж робить старий метод `addElement(Object element)`).

Методом `add(int index, Object element)` або старим методом `insertElementAt(Object element, int index)` можна вставити елемент у зазначене місце `index`. Метод `addAll(Collection coll)` дозволяє додати в кінець вектора всі елементи колекції `coll`. Методом `addAll(int index, Collection coll)` можливо вставити в позицію `index` всі елементи колекції `coll`. Метод `set(int index, object element)` замінює елемент, що стояв у векторі в позиції `index`, на елемент `element`. Кількість елементів у векторі завжди можна дізнатися методом `size()`. Метод `capacity()` повертає ємність вектора.

Логічний метод `isEmpty()` повертає `true`, якщо у векторі немає жодного елемента. Звернутися до першого елементу вектору можна методом `firstElement()`, до останнього - методом `lastElement()`, до будь-якого елемента - методом `get(int index)` або старим методом `elementAt(int index)`.

Ці методи повертають об'єкт класу `Object`. Перед використанням його варто привести до потрібного типу.

Одержати всі елементи вектора у вигляді масиву типу `Object[]` можна методами `toArray()` і `toArray(Object[] a)`. Другий метод заносить всі елементи вектора в масив `a`, якщо в ньому досить місця. Логічний метод `contains(Object element)` повертає `true`, якщо елемент `element` перебуває у векторі. Логічний метод `containsAll(Collection c)` повертає `true`, якщо вектор містить всі елементи зазначеної колекції.

Чотири методи дозволяють відшукати позицію зазначеного елемента `element`:

- `indexOf(Object element)` - повертає індекс першої появи елемента у векторі;
- `indexOf(Object element, int begin)` - веде пошук, починаючи з індексу `begin` включно;
- `lastIndexOf(Object element)` - повертає індекс останньої появи елемента у векторі;
- `lastIndexOf(Object element, int start)` - веде пошук від індексу `start` включно до початку вектора. Якщо елемент не знайдений, повертається `-1`.

Логічний метод `remove(Object element)` видаляє з вектора перше входження зазначеного елемента `element`. Метод повертає `true`, якщо елемент знайдений і видалення успішне.

Метод `remove(int index)` видаляє елемент із позиції `index` і повертає його як свій результат типу `Object`.

Аналогічні дії дозволяють виконати старі методи типу `void`: `removeElement(Object element)` і `removeElementAt(int index)`, що не повертають результату.

Видалити діапазон елементів можна методом `removeRange(int begin, int end)`, що не повертає результату. Видаляються елементи від позиції `begin` включно до позиції `end` виключно.

Видалити з даного вектора всі елементи колекції `coll` можна логічним методом `removeAll(Collection coll)`. Видалити останні елементи можна, просто урізавши вектор методом `setSize(int newSize)`. Видалити

всі елементи, крім вхідних у зазначену колекцію coll, дозволяє логічний метод retainAll(Collection coll). Видалити всі елементи вектора можна методом clear() або старим методом removeAllElements() або обнуливши розмір вектору методом setSize(). Програма 7.1 обробляє виділені з рядка слова за допомогою вектора.

Програма 7.1. Робота з вектором

```
Vector v = new Vector();
String s = "Рядок, який ми хочемо розібрати на слова.";
StringTokenizer st = new StringTokenizer(s, " \\t\\n\\r,.");
while(st.hasMoreTokens()){
    // Одержуємо слово й заносимо у вектор
    v.add(st.nextToken()); // Додаємо в кінець вектора
}
System.out.println(v.firstElement()); // Перший елемент
System.out.println(v.lastElement()); // Останній елемент
v.setSize(4); // Змінюємо число елементів
v.add("зібрали."); // Додаємо в кінець вкороченого вектора
v.set(3, "знову"); // Ставимо в позицію 3
for(int i = 0; i < v.size; i++) // Перебираємо весь вектор
System.out.print(v.get(i) + " ");
System.out.println();
```

Клас vector являє приклад того, як можна об'єкти класу object, тобто будь-які об'єкти, об'єднати в колекцію. Цей тип колекції впорядковує й навіть нумерує елементи. До кожного елементу звертаються безпосередньо по індексу. При додаванні й видаленні, елементи які залишилися, автоматично перенумеровуються.

7.2.2 Клас Stack Клас Hashtable Клас Properties

Клас stack з пакета java.util поєднує елементи в стек.

Стек(stack) реалізує порядок роботи з елементами який називається LIFO(Last In - First Out). Перед роботою створюється порожній стек конструктором stack(). Потім у стек додають і видаляють елементи, причому доступний тільки "верхній" елемент, той, що покладено у стек останнім.

Додатково до методів класу vector, клас stack містить п'ять методів, що дозволяють працювати з колекцією як зі стеком:

```
push(Object item) - поміщає елемент item у стек;
pop() - витягає верхній елемент зі стека;
peek() - читає верхній елемент, не витягаючи його зі стека;
empty() - перевіряє, чи не порожній стек;
search(object item) - знаходить позицію елемента item у стеці. Верхній елемент має позицію 1, під ним елемент 2 і т.д. Якщо елемент не знайдений, повертається - 1.
```

Програма 7.2 показує, як можна використовувати стек для перевірки парності символів.

Програма 7.2. Перевірка парності дужок

```
import java.util.*;
class StackTest{
    static boolean checkParity(String expression, String open, String close){
        Stack stack = new Stack();
        StringTokenizer st = new StringTokenizer(expression, " \\t\\n\\r+*/-(){}\"", true);
        while(st.hasMoreTokens()) {
            String tmp = st.nextToken();
            if(tmp.equals(open)), stack.push(open);
            if(tmp.equals(close)) stack.pop();
        }
        if(stack.isEmpty()) return true/return false;
    }
    public static void main(String[] args){
        System.out.println(
            checkParity(a -(b -(c - a) /(b + c) - 2), "(", ")");
        }
    }
}
```

Клас Hashtable

Клас Hashtable розширює абстрактний клас Dictionary. В об'єктах цього класу зберігаються пари "ключ - значення".

Кожний об'єкт класу Hashtable крім розміру(size) - кількості пар, має ще дві характеристики: ємність(capacity) - розмір буферу, і показник завантаженості(load factor) - відсоток заповнювання буферу, по досягненні якого збільшується його розмір.

Для створення об'єктів класу Hashtable використовує чотири конструктори:

```
Hashtable();
Hashtable(int capacity);
Hashtable(int capacity, float loadFactor);
Hashtable(Map f);
```

Для заповнення об'єкта класу Hashtable використовуються два методи:

Object put(Object key, Object value) - додає пари " key- value ", якщо ключа key не було в таблиці, і змінює значення value ключа key, якщо він вже є в таблиці;

void putAll(Map f) - додає всі елементи відображення f.

Метод get(Object key) повертає значення елемента із ключем key у вигляді об'єкта класу object.

Логічний метод containsKey(object key) повертає true, якщо в таблиці є ключ key.

Логічний метод `containsValue(Object value)` або старий метод `contains(object value)` повертають `true`, якщо в таблиці є ключі зі значенням `value`. Логічний метод `isEmpty()` повертає `true`, якщо в таблиці немає елементів. Метод `values()` представляє всі значення `value` таблиці у вигляді інтерфейсу `Collection`. Всі модифікації в об'єкті `collection` змінюють таблицю, і навпаки. Метод `keySet()` надає всі ключі `key` таблиці у вигляді інтерфейсу `set`. Всі зміни в об'єкті `set` коректують таблицю, і навпаки.

У програмі 7.3 показано, як можна використати клас `Hashtable` для створення телефонного довідника, а на рис. 7.1 - вивід цієї програми.

Програма 7.3. Телефонний довідник

```
import java.util.*;
class PhoneBook{
public static void main(String[] args){
    Hashtable yp = new Hashtable();
    String name = null;
    yp.put("John", "123-45-67");
    yp.put("Lemon", "567-34-12");
    yp.put("Bill", "342-65-87");
    yp.put("Gates", "423-83-49");
    yp.put("Batman", "532-25-08");
    try{
        name = args[0];
    } catch (Exception e){
        System.out.println("Usage: Java PhoneBook Name");
    }
    return;
}
if (yp.containsKey(name))
    System.out.println(name + "'s phone = " + yp.get(name));
else
    System.out.println("Sorry, no such name");
} }
```

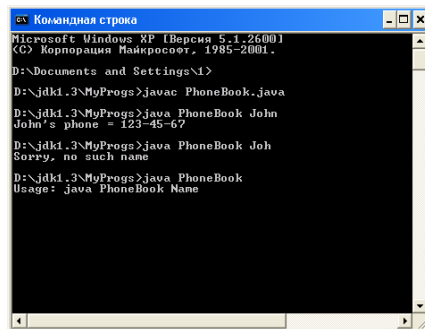


Рисунок 7.1 – Робота з телефонною книгою

Клас Properties

Клас Properties розширює клас Hashtable. Він призначений в основному для введення й виводу пари властивостей системи і їхніх значень. Пари зберігаються у вигляді рядків типу string. У класі Properties два конструктори:

Properties() - створює порожній об'єкт;

Properties(Properties default) - створює об'єкт із заданими парами властивостей default.

Два методи, що повертають значення ключа-рядка у вигляді рядка: string getProperty(string key) - повертає значення по ключу key ; String getProperty(String key, String defaultValue) - повертає значення по ключу key; якщо такого ключа немає, повертається defaultValue. Метод setProperty(String key, String value) додає нову пару, якщо ключа key немає, і змінює значення, якщо ключ key є. Метод load(InputStream in) завантажує властивості із вхідного потоку in. Методи list(PrintStream out) И list(PrintWriter out) виводять властивості у вихідний потік out. Метод store(OutputStream out, String header) виводить властивості у вихідний потік out із заголовком header.

Дуже проста програма 7.4 демонструє вивід всіх системних властивостей Java.

Програма 7.4. Вивід системних властивостей

```
class Prop{
    public static void main(String[] args){
        System.getProperties().list(System.out);
    } }
```

Приклади класів Vector, Stack, Hashtabie, Properties показують зручність класів-колекцій. Тому в Java2 розроблена ціла ієрархія колекцій. Вона показана на рис. 7.2. Курсивом записані імена інтерфейсів. Пунктирні лінії вказують класи, що реалізують ці інтерфейси. Всі колекції розбиті; на три групи, описані в інтерфейсах List, Set і Map.

Прикладом реалізації інтерфейсу List може служити клас Vector, прикладом реалізації інтерфейсу map - клас Hashtable.

Колекції List і Set мають багато спільного, тому їхні загальні методи об'єднані й винесені в суперінтерфейс Collection.

- `boolean removeAll(Collection coll)` – видаляє елементи зазначеної колекції, що лежать у даній колекції;
- `boolean retainAll(Collection coll)` – видаляє всі елементи даної колекції, крім елементів колекції `coll` ;
- `int size()` – повертає кількість елементів у колекції;
- `object[] toArray()` – повертає всі елементи колекції у вигляді масиву;
- `Object[] toArray(object[] a)` – записує всі елементи колекції в масив `a`, якщо в ньому досить місця.

7.2.4 Інтерфейс `ListIterator`

Інтерфейс `ListIterator` розширює інтерфейс `iterator`, забезпечуючи переміщення по колекції як у прямому, так і у зворотному напрямку. Він може бути реалізований тільки в тих колекціях, у яких є поняття наступного й попереднього елемента й де елементи пронумеровані.

В інтерфейс `ListIterator` додані наступні методи:

- `void add(Object element)` – додає елемент `element` перед поточним елементом;
- `boolean hasPrevious()` – повертає `true`, якщо в колекції є елементи, що стоять перед поточним елементом;
- `int nextIndex()` – повертає індекс поточного елементу; якщо поточним є останній елемент колекції, повертає розмір колекції;
- `Object previous()` – повертає попередній елемент і робить його поточним;
- `int previous index()` – повертає індекс попереднього елементу;
- `void set(Object element)` – заміняє поточний елемент елементом `element`; виконується відразу після `next()` або `previous()`.

Програма 7.5

```
ListIterator list = v.listIterator(); // Одержуємо ітератор вектору
// Показчик зараз перебуває перед початком вектору
try{
    while(list.hasNext())           // Поки у векторі є елементи
        System.out.println(list.next()); // Переходимо до наступного
                                         // елементу й виводимо його
    // Тепер показчик за кінцем вектору. Пройдемо до початку
    while(list.hasPrevious())
        System.out.println(list.previous());
} catch (Exception e){}
```

Цікаво, що повторне застосування методів `next()` і `previous()` один за одним буде видавати той самий поточний елемент.

Класи, що створюють списки

Клас `ArrayList` повністю реалізує інтерфейс `List` і ітератор типу `iterator`. Клас `ArrayList` дуже схожий на клас `Vector`, має той же набір методів і може використовуватися в тих же ситуаціях.

У класі `ArrayList` три конструктори: `ArrayList()`-створює порожній об'єкт; `ArrayList(Collection coll)` - створює об'єкт, що містить всі елементи колекції `coll`; `ArrayList(int initCapacity)` - створює порожній об'єкт ємності `initCapacity`.

Двонаправлений список

Клас `LinkedList` повністю реалізує інтерфейс `List` і містить додаткові методи, що перетворюють його у двонаправлений список. Він реалізує ітератор типу `iterator` і `listiterator`. Цей клас можна використати для обробки елементів у стеці, деці або двонаправленому списку.

У класі `LinkedList` два конструктори: `LinkedList` - створює порожній об'єкт; `LinkedList(Collection coll)` - створює об'єкт, що містить всі елементи колекції `coll`.

7.2.5 Колекції

Інтерфейс `Comparator` описує два методи порівняння:

`int compare(Object obj1, Object obj2)` — повертає від'ємне число, якщо `obj1` менше `obj2`; нуль, якщо вони вважаються рівними; позитивне число, якщо `obj1` більше `obj2`;

`boolean equals(Object obj)` — порівнює даний об'єкт із об'єктом `obj`, повертаючи `true`, якщо об'єкти рівні заданому цим методом.

Для кожної колекції можна реалізувати ці два методи, задавши конкретний спосіб порівняння елементів, і визначити об'єкт класу `SortedMap` другим конструктором. Елементи колекції будуть автоматично відсортовані в заданому порядку.

Класи, що створюють множини

Клас `HashSet` повністю реалізує інтерфейс `set` і ітератор типу `iterator`. Клас `HashSet` використовується в тих випадках, коли треба зберігати тільки одну копію кожного елемента.

У класі `HashSet` чотири конструктори:

`HashSet()`; `HashSet(int capacity)`; `HashSet(int capacity, float loadFactor)`; `HashSet(Collection coll)`.

Впорядковані множини

Клас `TreeSet` повністю реалізує інтерфейс `sortedSet` і ітератор типу `iterator`. Клас `TreeSet` реалізований як бінарне дерево пошуку, тобто його елементи зберігаються в упорядкованому виді. Це значно прискорює пошук потрібного елемента. Порядок задається або природним проходженням елементів, або об'єктом, що реалізує інтерфейс порівняння `Comparator`.

Цей клас зручний при пошуку елемента у множині, наприклад, для перевірки, чи володіє який-небудь елемент властивістю, що визначає множину.

У класі `TreeSet` чотири конструктори: `TreeSet()`; `TreeSet(Comparator c)`; `TreeSet(Collection coll)`; `TreeSet(SortedMap sf)`.

У програмі 7.6 показано, як можна зберігати комплексні числа в упорядкованому виді.

Програма 7.6. Зберігання комплексних чисел в упорядкованому виді

```
TreeSet ts = new TreeSet(new ComplexCompare());
ts.add(new Complex(1.2, 3.4));
ts.add(new Complex(-1.25, 33.4));
ts.add(new Complex(1.23, -3.45));
ts.add(new Complex(16.2, 23.4));
Iterator it = ts.iterator();
while(it.hasNext()),((Complex)it.next()).pr();
```

Дії з колекціями

Колекції призначені для зберігання елементів у зручному для подальшої обробки виді. Дуже часто обробка полягає в сортуванні елементів і пошуку потрібного елемента. Ці й інші методи обробки зібрані в клас `Collections`.

Методи класу `Collections`

Всі методи класу `collections` статичні, ними можна користуватися, не створюючи екземпляри класу `Collections`.

Сортування може бути зроблене тільки в упорядкованій колекції, яка реалізує інтерфейс `List`. Для сортування в класі `collections` є два методи:

`static void sort(List coll)` - сортує в природному порядку зростання колекцію `coll`, що реалізує інтерфейс `List`;

`static void sort(List coll, Comparator c)` - сортує колекцію `coll` у порядку, заданому об'єктом `c`. Після сортування можна здійснити бінарний пошук у колекції:

`static int binarySearch(List coll, Object element)` - відшукує елемент `element` у відсортованій у природному порядку зростання колекції `coll` і повертає індекс елемента або від'ємне число, якщо елемент не знайдений; від'ємне число показує індекс, з яким елемент `element` був би вставлений у колекцію, зі зворотним знаком;

static int binarySearchfList coll, Object element, Comparator c) - те ж, але колекція відсортована в порядку, певному об'єктом c.

Чотири методи знаходять найбільший і найменший елементи в упорядкованій колекції:

static Object max(Collection coll) - повертає найбільший у природному порядку елемент колекції coll;

static Object max(Collection coll, Comparator c) – те ж у порядку, заданому об'єктом c;

static Object min(Collection coll) - повертає найменший у природному порядку елемент колекції coll;

static Object min(Collection coll, Comparator c) - те ж у порядку, заданому об'єктом c.

Два методи "перемішують" елементи колекції у випадковому порядку:

static void shuffle(List coll) - випадкові числа задаються за замовчуванням;

static void shuffle(List coll, Random r) - випадкові числа визначаються об'єктом r.

Метод reverse(List coll) змінює порядок розташування елементів на зворотний.

Метод copy(List from, List to) копіює колекцію from у колекцію to.

Метод fill(List coll, Object element) замінює всі елементи існуючої колекції coll елементом element.

7.3 Завдання до роботи

7.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

7.3.2 Вивчити основні принципи роботи Java з класами-колекціями.

7.3.3 Виконати наступні завдання:

1) Student: id, Прізвище, Ім'я, По батькові, Дата народження, Адреса, Телефон, Факультет, Курс, Група. Створити масив об'єктів. Вивести:

- а) список студентів заданого факультету;
- б) списки студентів для кожного факультету й курсу;
- в) список студентів, що народилися після заданого року;
- г) список навчальної групи.

Ввести рядки з файлу, записати в список. Вивести рядки в файл у зворотному порядку.

2) Customer: id, Прізвище, Ім'я, По батькові, Адреса, Номер кредитної картки, Номер банківського рахунку. Створити масив об'єктів. Вивести:

- а) список покупців за абеткою;
- б) список покупців, у яких номер кредитної картки перебуває в заданому інтервалі.

Ввести число, занести його цифри в стек. Вивести число, у якого цифри йдуть у зворотному порядку.

3) Patient: id, Прізвище, Ім'я, По батькові, Адреса, Телефон, Номер медичної карти, Діагноз. Створити масив об'єктів. Вивести:

а) список пацієнтів, що мають даний діагноз;

б) список пацієнтів, в яких номер медичної карти перебуває в заданому інтервалі.

Створити в стеці індексний масив для швидкого доступу до записів у бінарному файлі.

4) Abiturient: id, Прізвище, Ім'я, По батькові, Адреса, Телефон, Оцінки. Створити масив об'єктів. Вивести:

а) список абітурієнтів, що мають незадовільні оцінки;

б) список абітурієнтів, середній бал у яких вище заданого;

в) вибрати задане число n абітурієнтів, що мають найвищий середній бал (вивести також повний список абітурієнтів, що мають напівпрохідний бал).

Створити список з елементів каталогу і його підкаталогів.

5) Book: id, Назва, Автор(и), Видавництво, Рік видання, Кількість сторінок, Ціна, Обкладинка. Створити масив об'єктів. Вивести:

а) список книг заданого автора;

б) список книг, випущених заданим видавництвом;

в) список книг, випущених після заданого року.

Занести вірша одного автора в список. Провести сортування по зростанню довжин рядків.

6) House: id, Номер квартири, Площа, Поверх, Кількість кімнат, Вулиця, Тип будинку, Строк експлуатації. Створити масив об'єктів. Вивести:

а) список квартир, що мають задане число кімнат;

б) список квартир, що мають задане число кімнат і розташованих на поверсі, що перебуває в заданому проміжку;

в) список квартир, що мають площу, що перевершує задану.

Створити стек з номерів запису. Організувати прямий доступ до елементів запису.

7) Phone: id, Прізвище, Ім'я, По батькові, Адреса, Номер кредитної картки, Дебет, Кредит, Час міських і міжміських розмов. Створити масив об'єктів. Вивести:

а) відомості про абонентів, у яких час внутріміських розмов перевищує задане;

б) відомості про абонентів, які користувалися міжміським зв'язком;

в) відомості про абонентів за абеткою.

Задати дві стеки, поміняти інформацію місцями.

8) Car: id, Марка, Модель, Рік випуску, Кольори, Ціна, Реєстраційний номер. Створити масив об'єктів. Вивести:

а) список автомобілів заданої марки;

б) список автомобілів заданої моделі, які експлуатуються більше n років;

в) список автомобілів заданого року випуску, ціна яких більше зазначеної.

Визначити множину на основі множини цілих чисел. Створити методи для визначення перетинання й об'єднання множин.

9) Product: id, Найменування, Виробник, Ціна, Строк зберігання, Кількість. Створити масив об'єктів. Вивести:

а) список товарів для заданого найменування;

б) список товарів для заданого найменування, ціна яких не перевершує задану;

в) список товарів, строк зберігання яких більше заданого.

Списки (стеки, черги) $I(1..n)$ і $U(1..n)$ містять результати n -вимірів струму й напруги на невідомому опорі R . Знайти наближене число R методом найменших квадратів.

10) Train: Пункт призначення, Номер поїзда, Час відправлення, Число місць (загальних, купе, плацкарт, люкс). Створити масив об'єктів. Вивести:

а) список поїздів, що йдуть до заданого пункту призначення;

б) список поїздів, що йдуть до заданого пункту призначення й відправляються після заданої години;

в) список поїздів, що відправляються до заданого пункту призначення й кількість загальних місць.

З використанням множин виконати попарне підсумовування довільного кінцевого ряду чисел за наступними правилами: на першому етапі підсумовуються попарно поруч стоячі числа, на другому етапі підсумовуються результати першого етапу й т.д. доти, поки не залишиться одне число.

11) Bus: Прізвище й ініціали водія, Номер автобуса, Номер маршруту, Марка, Рік початку експлуатації, Пробіг. Створити масив об'єктів. Вивести:

- а) список автобусів для заданого номера маршруту;
- б) список автобусів, які експлуатуються більше 10 років;
- в) список автобусів, пробіг у яких більше 100000 км. Не використовуючи допоміжних об'єктів, переставити від'ємні елементи списку в кінець, а позитивні - у початок списку.

Не використовуючи допоміжних об'єктів, переставити негативні елементи списку в кінець, а позитивні - у початок списку.

12) Airlines: Пункт призначення, Номер рейсу, Тип літака, Час вильоту, Дні тижня. Створити масив об'єктів. Вивести:

- а) список рейсів для заданого пункту призначення;
- б) список рейсів для заданого дня тижня;
- в) список рейсів для заданого дня тижня, час вильоту для яких більше заданого.

Задано рядок, що складається із символів '(', ')', '[', ']', '{', '}'. Перевірити правильність розміщення дужок. Використати стек.

7.3.4 Оформити звіт з роботи.

7.3.5 Відповісти на контрольні питання.

7.4 Зміст звіту

7.4.1 Тема та мета роботи.

7.4.2 Завдання до роботи.

7.4.3 Короткі теоретичні відомості.

7.4.4 Текст розробленої програми.

7.4.5 Результати виконання роботи.

7.4.6 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

7.5 Контрольні запитання

7.5.1 Основне призначення класу Vector у мові Java?

7.5.2 Які основні методи і яке їх застосування, класу Vector Ви знаєте?

7.5.3 Основне призначення класу Stack у мові Java?

7.5.4 Які основні методи і яке їх застосування, класу Stack Ви знаєте?

7.5.5 Основне призначення класу Hashtable у мові Java?

7.5.6 Які основні методи і яке їх застосування, класу Hashtable Ви знаєте?

7.5.7 Основне призначення класу Properties у мові Java?

7.5.8 Які основні методи і яке їх застосування, класу Properties Ви знаєте?

7.5.9 Основне призначення інтерфейсу Collection у мові Java?

7.5.10 Які основні методи і яке їх застосування, інтерфейсу Collection Ви знаєте?

8 ЛАБОРАТОРНА РОБОТА № 8 КЛАСИ-УТИЛІТИ

8.1 Мета роботи

Навчитися працювати із класами-утилітами в Java.

8.2 Основні теоретичні відомості

8.2.1 Робота з масивами

Робота з масивами

У класі `Arrays` з пакета `java.util` зібрана безліч методів для роботи з масивами.

Вісімнадцять статичних методів сортують масиви з різними типами числових елементів у порядку зростання чисел або просто об'єкти в їхньому природному порядку.

Вісім з них мають простий вигляд

```
static void sort(type[] a)
```

де `type` може бути один із семи примітивних типів `byte`, `short`, `int`, `long`, `char`, `float`, `double` або тип `Object`.

Вісім методів з тими ж типами сортують частину масиву від індексу `from` включно до індексу `to` виключно:

```
static void sort(type[] a, int from, int to)
```

Два методи впорядковують масив або його частину з елементами типу `Object` за правилом, заданому об'єктом `c`, що реалізує інтерфейс `Comparator`:

```
static void sort(Object[] a, Comparator c)
```

```
static void sort(Object[] a, int from, int to, Comparator c)
```

Після сортування можна організувати бінарний пошук у масиві одним з дев'яти статичних методів пошуку. Вісім методів мають вигляд

```
static int binarySearch(type[] a, type element)
```

де `type` - один з тих же восьми типів. Дев'ятий метод пошуку має вигляд

```
static int binarySearch(Object[] a, Object element, Comparator c)
```

Він відшукує елемент `element` у масиві, відсортованому в порядку, заданому об'єктом `c`.

Методи пошуку повертають індекс знайденого елемента масиву. Якщо елемент не знайдений, то повертається від'ємне число, що означає індекс.

Вісімнадцять статичних методів заповнюють масив або частину масиву значенням value:

```
static void fill(type[], type value)
static void fill(type[], int from, int to, type value)
```

де type - один з восьми примітивних типів або тип object.

Дев'ять статичних логічних методів порівнюють масиви:

```
static boolean equals(type[] a1, type[] a2)
```

де type - один з восьми примітивних типів або тип Object.

Масиви вважаються рівними, якщо вони мають однакову довжину й рівні елементи масивів з однаковими індексами, в такому випадку повертається true.

Програма 8.1 - простий приклад роботи із цими методами.

Програма 8.1. Застосування методів класу Arrays:

```
import java.util.*;
class ArraysTest{
public static void main(String[] args){
int[] a = {34, -45, 12, 67, -24, 45, 36, -56};
Arrays.sort(a) ;
for(int i = 0; i < a.length; i++){
System.out.print(a[i]. + " ");
System.out.println();
Arrays.fill(a, Arrays.binarySearch(a, 12), a.length, 0);
for(int i = 6; i < a.length; i++){
System.out.print(a[i] + " ");
System.out.println();
} }
}
```

8.2.2 Локальні установки

Сукупність різних форматів місцевості - "локаль", зберігається в об'єкті класу Locale з пакета java.util. Для створення такого об'єкта досить знати мову language і місцевість country. Іноді потрібно третя характеристика - варіант variant, що визначає програмний продукт, наприклад, "WIN", "MAC", "POSIX".

За замовчуванням місцеві установки визначаються операційною системою й читаються із системних властивостей.

```
user.language = ua           // Мова - росіянин
user.region = UA             // Місцевість - Росія
file.encoding = Cp1251       // Байтове кодування - CP1251
```

Вони визначають українську локаль і локальне кодування байтових символів. Локаль, встановлену за замовчуванням на тій

машині, де виконується програма, можна з'ясувати статичним методом `Locale.getDefault()`.

В класі `Locale` є два конструктори:

```
Locale(String language, String country)
```

```
Locale(String language, String country, String variant)
```

Параметр `language` - це рядок із двох малих літер, визначених стандартом ISO639, наприклад, "ua", "fr", "en". Параметр `country` - рядок із двох прописних букв, визначених стандартом ISO3166, наприклад, "UA", "US", "EN". Параметр `variant` не визначається стандартом, наприклад, рядок "Traditional".

Локаль часто вказують одним рядком "ru_RU", "en_EN", "en_US", "en_CA" і т.д.

Після створення локалі можна зробити її локалью за замовчуванням статичним методом:

```
Locale.setDefault(Locale newLocale);
```

Деякі статичні методи класу `Locale` дозволяють одержати параметри локалі за замовчуванням:

- `String getCountry()` – стандартний код країни із двох букв;
- `String getDisplayCountry()` – країна записується словом;
- `String getDisplayCountry(Locale locale)` – те ж для зазначеної локалі.

Такі ж методи є для мови й варіанта.

Можна переглянути список всіх локалей, визначених для даної JVM, і їх параметрів:

```
Locale[] getAvailableLocales()
```

```
String[] getISOCountries()
```

```
String[] getISOLanguages()
```

Установлена локаль надалі використовується при виводі даних у місцевому форматі.

8.2.3 Робота з датами й часом

Методи роботи з датами й часом зібрані у два класи: `Calendar` і `Date` з пакета `java.util`.

Об'єкт класу `Date` зберігає число мілісекунд, що пройшли з 1 січня 1970 р. 00:00:00 за Гринвічем.

Клас `Date` зручно використовувати для відліку проміжків часу в мілісекундах.

Одержати поточне число мілісекунд, можна статичним методом `System.currentTimeMillis()`

У класі Date два конструктори. Конструктор Date() заносить у створюваний об'єкт поточний час машини, на якій виконується програма, по системних годинниках, а конструктор Date(long millisec) - зазначене число.

Одержати значення, що зберігається в об'єкті, можна методом long getTime(), встановити нове значення - методом setTime(long newTime).

Три логічних методи порівнюють час:

– boolean after(long when) – повертає true, якщо час when більше даного;

– boolean before(long when) – повертає true, якщо час when менше даного;

– boolean after(Object when) – повертає true, якщо when - об'єкт класу Date і часи збігаються.

Ще два методи, порівнюючи час, повертають від'ємне число типу int, якщо даний час менше аргументу when; нуль, якщо часи збігаються; позитивне число, якщо даний час більше аргументу when:

– int compareTo(Date when);

– int compareTo(object when) – якщо when не відноситься до об'єктів класу Date, відбувається виняткова ситуація.

Перетворення мілісекунд, що зберігаються в об'єктах класу Date, у поточний час і дату робиться методами класу calendar.

Клас Calendar

Клас Calendar – абстрактний, у ньому зібрані загальні властивості календарів: юліанського, григоріанського, місячного.

Оскільки calendar – абстрактний клас, його екземпляри створюються чотирма статичними методами до заданої локалі й/або часовому поясу:

```
Calendar getInstance()
Calendar getInstance(Locale loc)
Calendar getInstance(TimeZone tz)
Calendar getInstance(TimeZone tz, Locale loc)
```

Для роботи з місяцями визначені цілочисельні константи від JANUARY до DECEMBER, для роботи із днями - константи від MONDAY до SUNDAY.

Перший день тижня можна дізнатися методом int getFirstDayOfWeek(), а встановити – методом setFirstDayOfWeek(int day), наприклад:

```
setFirstDayOfWeek(Calendar.MONDAY)
```

Інші методи дозволяють переглянути час і часовий пояс або встановити їх.

Підклас `GregorianCalendar`

У григоріанському календарі дві цілочисельні константи визначають ери: BC(before Christ) і AD(Anno Domini).

Сім конструкторів визначають календар за часом, годинному поясу й/або локалі:

```
GregorianCalendar()
GregorianCalendar(int year, int month, int date)
GregorianCalendar(int year, int month, int date, int hour, int minute)
GregorianCalendar(int year, int month, int date, int hour, int minute, int
second)
GregorianCalendar(Locale loc)
GregorianCalendar(TimeZone tz)
GregorianCalendar(TimeZone tz, Locale loc)
```

Після створення об'єкта варто визначити дату переходу з юліанського календаря на григоріанський календар методом `setGregorianChange(Date date)`. За замовчуванням це 15 жовтня 1582 р.:

```
GregorianCalendar greg = new GregorianCalendar(); greg.setGregorianChange(new
GregorianCalendar(200+, Calendar.FEBRUARY, 14).getTime());
```

Довідатися, чи є рік високосним у григоріанському календарі, можна логічним методом `isLeapYear()`.

Метод `get(int field)` повертає елемент календаря, заданий аргументом `field`. Для цього аргументу в класі `Calendar` визначені наступні статичні цілочисельні константи:

ERA	WEEK_OF_YEAR	DAY_OF_WEEK	SECOND
YEAR	WEEK_OF_MONTH	DAY_OF_WEEK_IN_MONTH	MILLISECOND
MONTH	DAY_OF_YEAR	HOUR_OF_DAY	ZONE_OFFSET
DATE	DAY_OF_MONTH	MINUTE	DST_OFFSET

Кілька методів `set()`, що використовують ці константи, встановлюють відповідні значення.

Подання дати й часу

Різні засоби подання дат і показань часу можна здійснити методами, зібраними в абстрактний клас `DateFormat` і його підклас `SimpleDateFormat` з пакету `Java.text`.

Клас `DateFormat` пропонує чотири стилі подання дати й часу:

– стиль `SHORT` представляє дату й час у короткому числовому виді: 27.04.01 17:32;

– стиль `MEDIUM` задає рік чотирма цифрами й показує секунди: 27.04.2001 17:32:45;

– стиль `LONG` представляє місяць словом і додає годинний пояс: 27 квітень 2001 р. 17:32:45 GMT+02.00;

– стиль FULL такий же, як і стиль LONG; за виключенням локалі США – додається ще день тижня.

Є ще стиль DEFAULT, що збігається зі стилем MEDIUM.

Визначення інший формату шаблону, наприклад:

```
SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy
hh.mm"); System.out.println(sdf.format(new Date()));
```

Одержимо вивід у такому виді: 27-04-2001 17.32.

Не у всіх локалях можна створити об'єкт класу SimpleDateFormat. У таких випадках використовуються статичні методи getInstance() класу DateFormat, що повертають об'єкт класу DateFormat.

Після створення об'єкту метод format() класу DateFormat повертає рядок з датою й часом, відповідно до заданого стилю. Як аргумент задається об'єкт класу Date.

Наприклад:

```
System.out.println("LONG: " + DateFormat.getDateInstance(DateFormat.LONG,
DateFormat.LONG).format(new Date()));
```

або

```
System.out.println("FULL: " + DateFormat.getDateInstance(DateFormat.FULL,
DateFormat.FULL, Locale.US).format(new Date()));
```

8.2.4 Одержання випадкових чисел

Одержати випадкове невід'ємне число, строго менше одиниці, у вигляді типу double можна статичним методом random() із класу java.lang.Math.

При першому звертанні до цього методу створюється генератор псевдовипадкових чисел, що використовується потім при одержанні наступних випадкових чисел.

Більш серйозні дії з випадковими числами можна організувати за допомогою методів класу Random з пакета java.util. У класі два конструктори:

Random (long seed) – створює генератор псевдовипадкових чисел, який використовує для початку роботи число seed; Random() - вибирає за початкове значення поточний час.

Створивши генератор, можна одержувати випадкові числа відповідного типу методами nextBoolean(), nextDouble(), nextFloat(), nextGaussian(), nextInt(), nextLong(), nextInt(int max) або записати

відразу послідовність випадкових чисел у заздалегідь певний масив байтів `bytes` методом `nextBytes(byte[] bytes)`.

Дійсні випадкові числа рівномірно розташовуються в діапазоні від 0,0 включно до 1,0 виключно. Цілі випадкові числа рівномірно розподіляються по всьому діапазоні відповідного типу за, одним виключенням: якщо в аргументі зазначене ціле число `max`, діапазон випадкових чисел буде від нуля включно до `max` виключно.

8.2.5 Взаємодія із системою

Клас `System` дозволяє здійснити взаємодію із системою під час виконання програми (`run time`). Але крім нього для цього є спеціальний клас `Runtime`.

Клас `Runtime` містить деякі методи взаємодії з JVM під час виконання програми. Кожна програма може одержати тільки один екземпляр даного класу статичним методом `getRuntime()`. Всі виклики цього методу повертають посилання на той самий об'єкт.

Методи `freeMemory()` і `totalMemory()` повертають кількість вільної й всієї пам'яті, що є в розпорядженні JVM для розміщення об'єктів, у байтах, типу `long`. Не варто твердо опиратися на ці числа, оскільки кількість пам'яті змінюється динамічно.

Метод `exit(int status)` запускає процес зупинки JVM і передає операційній системі статус завершення `status`. За згодою, ненульовий статус означає ненормальне завершення. Зручніше використовувати аналогічний статичний метод класу `system`.

Метод `halt(int status)` здійснює негайний зупинку JVM. Він не завершує запущені процеси нормально й повинен використовуватися тільки в аварійних ситуаціях.

Метод `loadLibrary(string libName)` дозволяє довантажити динамічну бібліотеку під час виконання по її імені `libName`.

Метод `load(string fileName)` довантажує динамічну бібліотеку по імені файлу `fileName`.

Втім, замість цих методів зручніше використовувати статичні методи класу `system` з тими ж іменами й аргументами.

Метод `gc()` запускає процес звільнення непотрібної оперативної пам'яті (`garbage collection`). Цей процес періодично запускається самою віртуальною машиною Java і виконується з невеликим пріоритетом,

але можна його запустити й із програми. Знов-таки зручніше використовувати статичний Метод `System.gc()`.

Кілька методів `exec()` запускають в окремих процесах виконавчі файли. Аргументом цих методів служить командний рядок виконавчого файлу.

Наприклад, `Runtime.getRuntime().exec("notepad")` запускає Програму блокнот на платформі MS Windows.

Методи `exec()` повертають екземпляр класу `process`, що дозволяє керувати запущеним процесом. Методом `destroy()` можна зупинити процес, методом `exitValue()` одержати його код завершення. Метод `waitFor()` припиняє основний підпроцес доти, поки не закінчиться запущений процес. Три методи `getInputStream()`, `getOutputStream()` И `getErrorStream()` повертають вхідний, вихідний потік і потік помилок запущеного процесу.

8.3 Завдання до роботи

8.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

8.3.2 Вивчити основні принципи роботи Java класами-утилітами.

8.3.3 Виконати наступні завдання:

Загальні завдання:

1. Реалізувати програму яка виводить значення поточної дати й часу з певним інтервалом з обраними параметрами локалізації;
2. Реалізувати клас, що дозволяє виконувати функції описані у вправі 1 і зберігати результати у файл;

Індивідуальні завдання:

1. Реалізувати клас що дозволяє зберігати результати роботи програми із загального завдання 1 в об'єкт класу `SortedSet` і впорядковувати їх у наступному порядку - мілісекунди, секунди, години, дні, місяці, роки.
2. Доповнити програму із загального завдання 2 можливістю виводу значень години, хвилин, секунд, дня, місяця, року в різних системах числення.

8.3.4 Оформити звіт з роботи.

8.3.5 Відповісти на контрольні питання.

8.4 Зміст звіту

8.4.1 Тема та мета роботи.

8.4.2 Завдання до роботи.

8.4.3 Короткі теоретичні відомості.

8.4.4 Текст розробленого програмного забезпечення.

8.4.5 Результати виконання роботи.

8.4.6 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

8.5 Контрольні запитання

8.5.1 Що таке класи-утиліти в мові Java?

8.5.2 Основне застосування й методи класу Arrays?

8.5.3 Що таке локальні установки й для чого вони потрібні в мові Java?

8.5.4 Які методи для роботи з локалью в мові Java Ви знаєте?

8.5.5 Яке основне застосування класу Date?

8.5.6 Які основні методи класу Date Ви знаєте?

8.5.7 Основне застосування й методи класу DateFormats?

8.5.8 Які особливості випадкових чисел у мові Java?

8.5.9 Які особливості копіювання масивів у мові Java?

8.5.10 Основне застосування й методи класу System?

8.5.11 Яким образом в Java можна одержати значення поточного часу й дати?

8.5.12 Для чого призначений клас Calendar?

8.5.13 Який клас дозволяє одержати поточну дату відповідно до регіональних параметрів?

ЛИТЕРАТУРА

1. Фленаган Д.. Java. Справочник / Д. Фленаган. – СПб. : Символ-Плюс, 2004. – 1040 с.
2. Шилдт Г. Искусство программирования на Java / Г. Шилдт. – СПб. : Вильямс, 2005. – 336 с.
3. Хабибуллин И. Ш. Разработка Web-служб средствами Java / И. Ш. Хабибуллин. – СПб. : БХВ-Петербург, 2003. – 400 с.
4. Вязовик Н. А. Программирование на Java. Курс лекций / Н. А. Вязовик – М. : Интернет-университет информационных технологий, 2003 – 592 с.
5. Ноутон П. Java 2. Наиболее полное руководство / П. Ноутон, Г. Шилдт. – СПб. : ВHV-Санкт-Петербург, 2007 – 1072 с.
6. Хорстманн К. С. Java 2. Основы / К. С. Хорстманн, Г. Корнелл. – СПб. : Вильямс, 2003 – Т. I. – 848 с.