Лабораторна робота №2

Основи об'єктно-орієнтованого програмування засобами мови Java.

Класи в Java

Мета лабораторної роботи:	1. Вивчити основні концепції ООП Java. 2. Вивчити прийоми програмування за допомогою класів на мові Java.

Зміст роботи.

1. Теоретичні відомості

Короткі теоретичні відомості про об'єктно-орієнтоване програмування можна переглянути тут.

Короткі теоретичні відомості про використання класів і об'єктів Java можна прочитати в цьому документі.

Дивимося тут приклад реалізації класу з прикладу комплексних чисел .

2. Завдання на лабораторну роботу

Завдання згідно з варіантом можна подивитися тут.

3. Звіт

Повинен містити: постановку завдання, програмний код розв'язання, результат роботи написаної програми

4. **приклад**

Основні поняття ООП

Можливості програмування завжди були обмежені можливостями комп'ютера, або можливостями людини. У минулому столітті основним обмеженням були низькі продуктивні можливості комп'ютера. Нині фізичні обмеження відійшли другого план. З дедалі більш глибоким проникненням комп'ютерів у всі сфери людської діяльності, програмні системи стають все більш простими для користувача та складними з внутрішньої архітектури. Програмування стало справою команди і зміну алгоритмічним ідеологіям програмування прийшли евристичні, дозволяють досягти позитивного результату різними шляхами.

Базовим способом боротьби зі складністю програмних продуктів стало об'єктно-орієнтоване програмування (ООП), що є найпопулярнішою парадигмою. ООП пропонує способи мислення та структурування коду.

ООП – методологія програмування, заснована на представленні програмного продукту як сукупності об'єктів, кожен із яких є екземпляром конкретного класу. ООП використовує як базові елементи евристичну взаємодію об'єктів.

Об'єкт- Справжня іменована сутність, що володіє властивостями і проявляє свою поведінку.

У застосуванні до об'єктно-орієнтованих мов програмування поняття об'єкта та класу конкретизується, а саме:

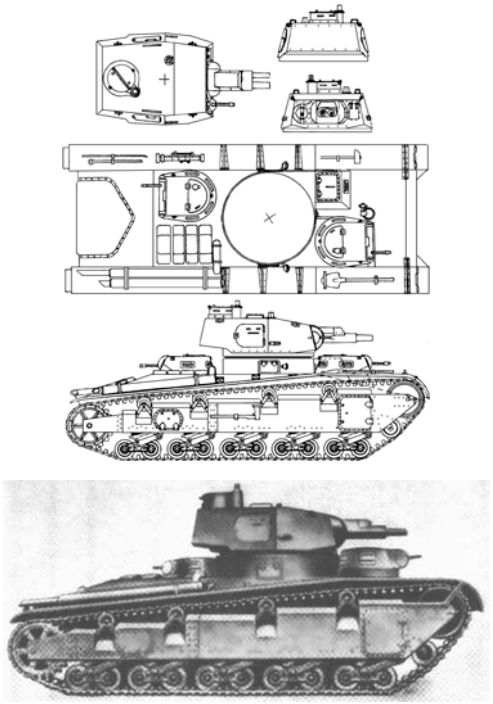
Об'єкт – набір даних (полів об'єкта), що володіють ім'ям, фізично що знаходяться в пам'яті комп'ютера, і методів, що мають доступ до них. Ім'я використовується для доступу до полів та методів, що становлять об'єкт. У граничних випадках об'єкт може містити полів чи методів, і навіть мати імені.

Будь-який об'єкт відноситься до певного класу.

Клас містить опис даних та операцій з них.

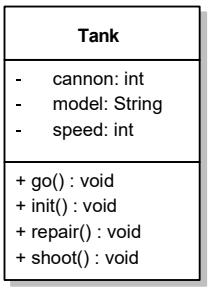
У класі дається узагальнене опис деякого набору родинних, реально існуючих об'єктів. Об'єкт – конкретний екземпляр класу.

Як приклад можна навести креслення танка або його опис (клас) та реальний танк (примірник класу, або об'єкт).



Мал. 1.Опис класу (креслення) та реальний об'єкт

Клас прийнято позначати у вигляді прямокутника, розділеного на три частини:



Мал. 2.Графічне зображення класу

Об'єктно-орієнтоване програмування ґрунтується на принципах:

- абстрагування даних;
- інкапсуляції;

- успадкування;
- поліморфізму;
- "Пізнього зв'язування".

Інкапсуляція (encapsulation) – принцип, що поєднує дані та код, що маніпулює цими даними, а також захищає насамперед дані від прямого зовнішнього доступу та неправильного використання. Іншими словами, доступ до даних класу можливий лише за допомогою методів цього класу.

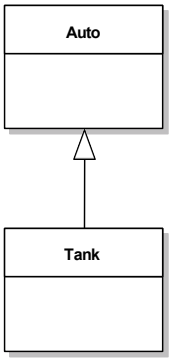
Спадкування (inheritance) – це процес, з якого один об'єкт може набувати властивості іншого. Точніше, об'єкт може успадковувати основні властивості іншого об'єкта і додавати до них властивості та методи, характерні лише йому.

Спадкування буває двох видів:

одиночне - клас (він же підклас) має один і лише один суперклас (предок);

множина – клас може мати будь-яку кількість предків (в Java заборонено).

Графічно успадкування часто зображується у вигляді діаграм UML:



Мал. 1.3.Графічне зображення наслідування

Клас Auto називається суперкласом, а Tank - підкласом.

Поліморфізм (polymorphism) – механізм, що використовує одне й те саме ім'я методу для вирішення двох або більше схожих, але дещо відмінних завдань.

Метою поліморфізму стосовно ООП є використання одного імені завдання загальних для класу дій. У загальному сенсі концепцією поліморфізму є ідея «один інтерфейс, безліч методів».

Механізм «пізнього зв'язування» у процесі виконання програми визначає належність об'єкта конкретному класу та здійснює виклик методу, що відноситься до класу, об'єкт якого був використаний.

Механізм "пізнього зв'язування" дозволяє визначати версію поліморфного методу під час виконання програми. Іншими словами, іноді неможливо на етапі компіляції визначити, яку версію перевизначеного методу буде викликано на тому чи іншому кроці програми.

Наріжним каменем наслідування та поліморфізму постає наступна парадигма: «об'єкт підкласу може використовуватися усюди, де використовується об'єкт суперкласу».

При виклик методу класу він шукається у самому класі. Якщо метод існує, він викликається. Якщо ж метод у поточному класі відсутня, то звернення відбувається до батьківського класу і метод шукається у нього. Якщо пошук невдалий, він триває вгору ієрархічним деревом до кореня (верхнього класу) ієрархії.

Класи та об'єкти

Класи в мові Java поєднують поля класу, методи, конструктори та логічні блоки. Основні відмінності від класів C++: усі функції визначаються усередині класів та називаються методами; неможливо створити метод, який не є методом класу, або оголосити метод поза класом; ключове слово inline, як і C++ , не підтримується; специфікатори доступу public, private, protected впливають лише ті оголошення полів, методів і класів, яких вони стоять, а чи не на ділянку від однієї до іншого специфікатора, як і C++; За замовчуванням елементи не встановлюються в private, а доступні для класів з цього пакета. Оголошення класу має вигляд:

```
[специфікатори] class Ім'яКласу [extends СуперКлас]
   [implements список_інтерфейсів]{/*визначення
   класу*/}
```

Специфікатор доступу до класу може бути public (клас доступний у даному пакеті та поза пакетом), final (клас не може мати підкласів), abstract (клас може містити абстрактні методи, об'єкт такого класу створити не можна). За замовчуванням специфікатор встановлюється у дружній (friendly). Такий клас доступний лише у поточному пакеті. Специфікатор friendly при оголошенні взагалі не використовується і не є ключовим словом мови. Це слово використовується, щоб визначити значення за замовчуванням.

Будь-який клас може успадковувати властивості та методи суперкласу, вказаного після ключового слова extends, і включати безліч інтерфейсів, перерахованих через кому після ключового слова implements. Інтерфейси є абстрактні класи, що містять тільки нереалізовані методи.

// Приклад # 1: простий приклад Java Beans класу: User.java

```
package lab1;
```

```
public classUser {
      public intnumericCode;//порушення інкапсуляції
      privateString password;

      public voidsetNumericCode(int value) {
            if(value > 0) numericCode = value;
            else numericCode = 1;
      }
      public intgetNumericCode() {
            returnnumericCode;
      }
      public voidsetPassword(String pass) {
            if (pass! = null) password = pass;
            else password = "11111";
      }
      publicString getPassword() {
      //public String getPass() {//некоректно – неповне ім'я методу
            returnpassword;
      }
}
```

Клас User містить два поля numericCode і password, позначені як public і private. Значення поля **password**можна змінювати лише за допомогою методів, наприклад, setPassword(). Поле numericCode доступне безпосередньо через об'єкт класу User. Доступ до методів та public-полів даного класу здійснюється лише після створення об'єкта даного класу.

```
/*Приклад # 2 : створення об'єкта, доступ до полів
 та методів об'єкта: UserView.java : Runner.java */
packagelab1;

classUserView {
      public static voidwelcome(User obj) {
  System.out.printf("Привіт! Введено код: %d, пароль: %s",
        obj.getNumericCode(), obj.getPassword());
      }
}
public classRunner {
       public static voidmain(String[] args) {
            User user = new User();
            user.numericCode = 71; // Некоректно - прямий доступ
            // user.password = null; // поле
            недоступнеuser.setPassword("pass"); //коректно
            UserView.welcome(user);
     }
}
```

Компіляція та виконання цього коду приведуть до виведення на консоль наступної інформації:

```
Привіт! Введено код: 71, пароль: pass
```

Створення об'єктів

Об'єкт класу створюється за два кроки. Спочатку оголошується посилання об'єкт класу. Потім за допомогою оператора **new**створюється екземпляр об'єкта, наприклад:

```
User user; //Оголошення посилання
user=new User(); //Створення об'єкта
```

Однак ці дві дії зазвичай поєднують в одну:

```
User user = new User();/*оголошення посилання та створення об'єкта*/
```

Оператор new викликає конструктор, тому у круглих дужках можуть стояти аргументи, що передаються конструктору. Операція присвоювання для об'єктів означає, що два посилання будуть вказувати на ту саму ділянку пам'яті.

Порівняння об'єктів

Операції порівняння посилань на об'єкти немає особливого сенсу, оскільки у своїй порівнюються адреси. Для порівняння значень об'єктів необхідно використати відповідні методи, наприклад equals(). Цей метод успадковується у кожний клас із суперкласу Object, що лежить у корені дерева ієрархії всіх класів і перевизначається у довільному класі визначення еквівалентності вмісту двох об'єктів цього класу.

/ приклад # 7: порівняння рядків та об'єктів: ComparingStrings.java */*

```
packagelab1;

public classComparingStrings {
     public static voidmain(String[] args) {
          String s1, s2;
          s1 = "Java";
          s2 = s1; // змінна посилається на той самий рядок
          System.out.println("порівняння посилань"
             + (s1 = = s2)); // результат true

          // Створення нового об'єкта додаванням символу
          s1 + = '2';
          // s1-="a"; //помилка, віднімати рядки не можна
          // Створення нового об'єкта копіюванням
          s2 = новий String(s1);
          System.out.println("порівняння посилань"
             + (s1 = = s2)); // результат false

          System.out.println("порівняння значень"
             + s1.equals(s2)); // результат true
     }
}
```

КЛАСИ

Клас представляє опис сукупності об'єктів із загальними атрибутами, методами, відносинами та семантикою.

Класи – основний елемент абстракції мови Java, основне призначення якого, крім реалізації призначеного йому договору, це приховування реалізації. Класи завжди взаємодіють один з одним і поєднуються в пакети. З пакетів створюються модулі, які взаємодіють один з одним лише через обмежену кількість методів та класів, не маючи жодного уявлення про процеси, що відбуваються всередині інших модулів.

Ім'я класу в пакеті має бути унікальним. Фізично пакет є каталогом, у якому поміщаються програмні файли, містять реалізацію класів.

Класи дозволяють провести декомпозицію поведінки складної системи до множини елементарних взаємодій пов'язаних об'єктів. Клас визначає структуру та/або поведінку деякого елемента предметної області, для якої розробляється програмна модель.

Визначення найпростішого класу без наслідування має вигляд:

```
class Ім'яКласа {
    {}//логічні блоки
    // дружні дані та методиprivate// закриті
    дані та методиprotected// захищені дані та
    методиpublic// Відкриті дані та методи
}
```

Змінні класу та константи

Класи інкапсулюють змінні та методи – члени класу. Змінні класу оголошуються у ньому так:

```
специфікатор тип ім'я;
```

У мові Java можуть використовуватися статичні змінні класу, оголошені один раз для класу зі специфікатором static і однакові всім екземплярів (об'єктів) класу, або змінні екземпляра класу, створювані кожному за об'єкта класу. Поля класу оголошуються зі специфікаторами доступу public, private, protected або за промовчанням без специфікатора. Крім даних – членів класу, у методах класу використовуються локальні змінні та параметри методів. На відміну від змінних класу, що інкапсулюються нульовими елементами, змінні методи не ініціалізуються за умовчанням.

Змінні із специфікатором final є константами. Специфікатор final можна використовувати для змінної, оголошеної методом, і навіть параметра методу.

У наступному прикладі розглядаються оголошення та ініціалізація значень полів класу та локальних змінних методу, а також використання параметрів методу:

```
/* приклад # 1: типи атрибутів та змінних: Second.java */
packagelab2;
importjava.util.*;

classSecond {
     private intx; // змінна екземпляра класу
     private inty = 71; // Змінна екземпляра класу
     public final int CURRENT_YEAR = 2007; //
     константа protected static int bonus; // змінна
     класу
     staticString version = "Java SE 6"; // змінна класу
     protectedCalendar now;
```

```
    public intmethod(int z) { // параметр методу
        z++;
        inta; // локальна змінна методу
        //a++; // помилка компіляції, значення не задано
        a = 4; //ініціалізація
        a++;
        now = Calendar.getInstance();//ініціалізація
        returna + x + y + z;
    }
}
```

У розглянутому прикладі як змінні екземпляри класу, змінних класу та локальних змінних методу використані дані базових типів, які є посиланнями на об'єкти (крім String). Дані можуть бути посиланнями, призначити яким реальні об'єкти можна за допомогою оператора new.

Обмеження доступу

Мова Java надає кілька рівнів захисту, які забезпечують можливість налаштування області видимості даних та методів. Через наявність пакетів Java працює з чотирма категоріями видимості між елементами класів:

- за умовчанням – дружні члени класу доступні класам, які у тому пакеті;
- **private**- Члени класу доступні тільки членам даного класу;
- **protected**- Члени класу доступні класам, що знаходяться в тому ж пакеті, і підкласам - в інших пакетах;
- **public**– члени класу доступні для всіх класів у цьому та інших пакетах.

Член класу (поле або метод), оголошений public, доступний із будь-якого місця поза класом. Все, що оголошено private, є лише методами всередині класу і ніде більше. Якщо у елемента взагалі не вказаний модифікатор рівня доступу, такий елемент буде видно і доступний з підкласів і класів того ж пакета. Саме такий рівень доступу використовується за умовчанням. Якщо ж необхідно, щоб елемент був доступний з іншого пакета, але тільки підклас того класу, якому він належить, потрібно оголосити такий елемент зі специфікатором protected. Дія специфікаторів доступу поширюється лише на той елемент класу, перед яким вони стоять.

Специфікатор доступу public може також стояти перед визначенням зовнішнього класу. Якщо цей специфікатор відсутній, клас недоступний з інших пакетів.

Конструктори

Конструктор - метод, який автоматично викликається при створенні об'єкта класу та виконує дії з ініціалізації об'єкта. Конструктор має те саме ім'я, що й клас; викликається не по імені, а лише разом із ключовим словом new при створенні екземпляра класу. Конструктор не повертає значення, але може мати параметри і бути перевантаженим.

Деструктори в мові Java не використовуються, об'єкти знищуються збирачем сміття після припинення використання (втрати посилання). Аналогом деструктора є метод finalize(). Виконуючий середовище мови Java буде викликати його щоразу, коли збирач сміття знищуватиме об'єкт класу, якому відповідає жодне посилання.

```
/* приклад # 2: перевантаження конструктора: Quest.java */
packagelab2;

public classQuest {private
      int id; private
      String text;
      // конструктор без параметрів (за замовчуванням)
      publicQuest() {
         super();/* якщо клас буде оголошено без конструктора,
                  то компілятор надасть його саме у такому
                  вигляді*/
      }
      // конструктор із параметрами
      publicQuest(int idc, String txt) {
            super();/*виклик конструктора суперкласу явно
            необов'язковий, компілятор вставить його
            автоматично*/
            id = idc;
            text = txt;
      }
}
```

Об'єкт класу Quest може бути створений двома способами, що викликають один із конструкторів:

```
Quest a = new Quest();//ініціалізація полів значеннями за умовчанням
Quest b = new Quest (71, "Скільки біт займає boolean?");
```

Оператор new викликає конструктор, тому в круглих дужках можуть стояти аргументи, що передаються конструктору.

Якщо конструктор класу не визначено, Java надає конструктор за замовчуванням без параметрів, який ініціалізує поля класу значеннями за промовчанням, наприклад, 0, false, null. Якщо конструктор з параметрами визначено, то конструктор за замовчуванням стає недоступним і його виклику необхідне явне оголошення такого конструктора. Конструктор підклас завжди викликає конструктор суперкласу. Цей виклик може бути явним або неявним і завжди

знаходиться в першому рядку коду конструктора. Якщо конструктору суперкласу потрібно передати параметри, то потрібний явний виклик:

```
super(параметри);
```

У наступному прикладі оголошено клас Point з двома полями (атрибутами), конструктором та методами для ініціалізації та вилучення значень атрибутів.

```
/* приклад # 3: обчислення відстані між точками: Point.java: LocateLogic.java: Runner.java */
package lab2;

public class Point {
/* об'єкт ініціалізується при створенні та не змінюється */
	private final double x;
	private final double y;

	public Point(final double xx, final double yy) {
		super();
		x = xx;
		y = yy;
	}
	public double getX() {
		return x;
	}
	public double getY() {
		return y;
	}
}
package lab2;

public class LocateLogic {
	public double calculateDistance(
					Point t1, Point t2) {
		/* обчислення відстані */
		double dx = t1.getX() - t2.getX();
		double dy = t1.getY() - t2.getY();
		return Math.hypot(dx, dy);
	}
}
package lab2;
public class Runner {
	public static void main(String[] args) {
	// локальні змінні є членами класу Point t1 =
		новий Point(5, 10); Point t2 =
		New Point (2, 6);
		System.out.print("відстань дорівнює:"
		+ New LocateLogic().calculateDistance(t1, t2));
	}
}
```

В результаті буде виведено:

```
відстань дорівнює: 5.0
```

Конструктор оголошується зі специфікатором public, щоб була можливість викликати його під час створення об'єкта у будь-якому пакеті програми. Специфікатор private не дозволяє створювати об'єкти поза класом, а специфікатор «за замовчуванням» – поза пакетом. Специфікатор protected дозволяє створювати об'єкти у поточному пакеті та для підкласів в інших пакетах.

Методи

Винахід методів є другим за важливістю відкриттям після створення комп'ютера. Метод – основний елемент структурування ходу.

Усі функції Java оголошуються лише всередині класів та називаються методами. Найпростіше визначення методу має вигляд:

```
returnType methodName(список_параметрів) {
    // Тіло методу
    returnvalue; // якщо потрібне повернення значення (returnType не void)
}
```

Якщо метод не повертає значення, ключове слово return може бути відсутнім, тип значення, що повертається в цьомуУ разі буде void. Замість порожнього списку параметрів методу тип void не вказується, лише порожні дужки. Виклик методів здійснюється з об'єкта чи класу (для статичних методів):

```
objectName.methodName();
```

Методи-конструктори на ім'я викликаються автоматично тільки при створенні об'єкта класу за допомогою оператора new.

Щоб створити метод, потрібно всередині оголошення класу написати оголошення методу і потім реалізувати його тіло. Оголошення методу як мінімум має містити тип значення, що повертається (можливий void) та ім'я методу. У наведеному нижче оголошенні методу елементи, укладені квадратні дужки, є необов'язковими.

```
[доступ] [static] [abstract] [final] [native]
[synchronized] returnType methodName(список_параметрів)
                                    [throws список_виключень]
```

Як і для полів класу, специфікатор доступу до методів може бути public, private, protected і за промовчанням. При цьому методи суперкласу можна перевантажувати чи перевизначати у породженому підкласі.

Оголошені методі змінні є локальними змінними методу, а чи не членами класів, і ініціалізуються значеннями за умовчанням під час створення об'єкта класу чи виклик методу.

Статичні методи та поля

Поля даних, оголошені у класі як static, є спільними всім об'єктів класу і називаються змінними класу. Якщо один об'єкт змінить значення такого поля, то цю зміну побачать усі об'єкти. p align="justify"> Для роботи зі статичними атрибутами використовуються статичні методи, оголошені зі специфікатором static. Такі методи є методами класу, не прив'язані до жодного об'єкта і не містять покажчика цього на конкретний об'єкт, що викликав метод. Статичні методи реалізують парадигму «раннього зв'язування», яка жорстко визначає версію методу на етапі компіляції. Унаслідок недоступності покажчика ці статичні поля і методи не можуть звертатися до нестатичних полів і методів безпосередньо, тому що для звернення до статичних полів і методів достатньо імені класу, в якому вони визначені.

```
// Приклад # 4: статичні метод і поле: Mark.java
package lab2;

public class Mark {
	private int mark = 3;
	public static int coeff= 5;

	public double getResult() {
		return (double)coeff*mark/100;
	}
	public static void setCoeffFloat(float c) {
		coeff= (int) coeff * c;;
	}
	public void setMark(int mark) {
		this. mark = mark;
	}
//зі статичного методу не можна звернутися до нестатичних полів та методів
	/*public static int getResult() {
		setMark(5);//помилка
		return coeff*mark/100;//помилка
	}*/
}
```

Під час створення двох об'єктів

Mark ob1 = new Mark(); Mark ob2

= New Mark ();

Значення ob1.coeff і ob2.coeff дорівнює 5, оскільки розташовується в одній і тій же області пам'яті. Змінити значення статичного члена можна прямо через ім'я класу:

```
Mark.coeff = 7;
```

Виклик статичного методу також слід здійснювати за допомогою вказівки:
ClassName.methodName(), А саме:

```
Mark.setCoeffFloat();
float z = Math.max (x, y);// Визначення максимуму з двох значень
System.exit(1);// Екстрене завершення роботи програми
```

Статичний метод можна викликати також з використанням імені об'єкта, але такий виклик знижує якість коду і не буде логічно коректним, хоча не призведе до помилки компіляції.

Перевизначення статичних методів класу немає практичного сенсу, оскільки звернення до статичного атрибуту чи методу здійснюється здебільшого у вигляді завдання імені класу, якому вони належать.

Модифікатор final

Модифікатор final використовується визначення констант як члена класу, локальної змінної чи параметра методу. Методи, оголошені як final, не можна заміщати у підкласах, для класів – створювати підкласи. Наприклад:

```
/* приклад # 5: final-поля та методи: Rector.java: ProRector.java */
package lab2;

public class Rector {
```

```
        // ініціалізована константа
        final intID = (int) (Math.random () * 10);
        // Неініціалізована константа
        finalString NAME_RECTOR;

        publicRector() {
        // ініціалізація у конструкторі
              NAME_RECTOR = "Старий"; / / тільки один раз!
        }
        // {NAME_RECTOR = "Новий";} // тільки один раз!

        public final voidjobRector() {
              // Реалізація
              // ID = 100; //помилка!
        }
        public booleancheckRights(final int num) {
              // id = 1; //помилка!
              final intCODE = 72173394;
              if(CODE == num) return true;
              else return false;
        }
        public static voidmain(String[] args) {
              System.out.println(new Rector().ID);
        }
}
packagelab2;

public classProRector extends Rector {
        // public void jobRector(){} //заборонено!
}
```

Константа може бути оголошена як поле класу, але не проініціалізована. У цьому випадку вона має бути проініціалізована в логічному блоці класу, укладеному в {}, або конструкторі, але тільки в одному із зазначених місць. Значення за умовчанням константа отримати не може, на відміну від змінних класу. Константи можуть бути оголошені в методах як локальні або параметри методу. В обох випадках значення таких констант не можна змінювати.

Абстрактні методи

Абстрактні методи розміщуються в абстрактних класах або інтерфейсах, тіла таких методів відсутні і повинні бути реалізовані в підкласах.

```
/* приклад # 6: абстрактний клас і метод: AbstractCourse.java */
public abstract classAbstractCourse {
     privateString name;
     publicAbstractCourse() { }
     public abstract voidchangeTeacher(int id);/*визначення
                                          методу відсутня*/
     publicsetName(String n)
              {name = n;
     }
}
```

На відміну від інтерфейсів, абстрактний клас може містити і абстрактні, і неабстрактні методи, а може і не містити жодного абстрактного методу.

Докладніше абстрактні класи та інтерфейси вивчаються у розділі «Абстрактні класи. Інтерфейси. Пакети».

Логічні блоки

При описі класу можна використовувати логічні блоки. Логічним блоком називається код, укладений у фігурні дужки і який не належить жодному методу поточного класу, наприклад:

```
{ /* код */ }
static{ /* код */ }
```

Логічні блоки найчастіше використовуються як ініціалізатори полів, але можуть містити виклики методів та звернення до полів поточного класу. При створенні об'єкта класу вони викликаються послідовно, як розміщення, разом із ініціалізацією полів як проста послідовність операторів, і лише після виконання останнього блоку буде викликаний конструктор класу. Операції з полями класу всередині логічного блоку до явного оголошення цього поля можливі тільки при використанні посилання this, що є посиланням на поточний об'єкт.

Логічний блок може бути оголошений із специфікатором static. У цьому випадку він викликається лише один раз у життєвому циклі додатка при створенні об'єкта або зверненні до статичного методу (поля) даного класу.

*/*Приклад # 7 : використання логічних блоків при оголошенні класу:*

*Department.java: DemoLogic.java */*

```
package lab2;

public class Department {
	{
		System.out.println("logic(1) id=" + this.id);
	}
	static{
		System.out.println("static logic");
	}
	private int id = 7;

	public Department(int d) {
		id = d;
		System.out.println("конструктор id="+id);
	}
	int getId() {
		return id;
	}
	{
		id = 10;
		System.out.println("logic(2) id=" + id);
	}
}
package lab2;

public class DemoLogic {
	public static void main(String[] args) {
	    Department obj = new Department(71);
	    System.out.println("значення id=" + obj.getId());
	}
}
```

В результаті виконання цієї програми буде виведено:

```
static logic
logic (1) id=0
logic (2) id=10
конструктор id=71
значення id=71
```

У першому рядку виведення поле id отримає значення за промовчанням, оскільки пам'ять для нього виділено під час створення об'єкта, а значення ще проініціалізовано. У другому рядку виводиться значення id, що дорівнює 10, оскільки після ініціалізації атрибута класу було викликано логічний блок, який змінив його значення.

Перевантаження методів

Метод називається перевантаженим, якщо існує кілька його версій з тим самим ім'ям, але з різним списком параметрів. Перевантаження реалізує раннє зв'язування. Перевантаження може обмежуватись одним класом. Методи з однаковими іменами, але з різними списком параметрів і значеннями, що повертаються, можуть знаходитися в різних класах одного ланцюжка успадкування і також будуть перевантаженими. Якщо останньому випадку списки параметрів збігаються, має місце інший механізм – перевизначення методу.

Статичні методи можуть перевантажуватись нестатичними і навпаки – без обмежень.

При виклику перевантажених методів слід уникати ситуацій, коли компілятор не зможе вибрати той чи інший метод.

/ приклад # 8: виклик перевантажених методів: NumberInfo.java */*

```
package lab2;

public class NumberInfo {
	public static void viewNum(Integer i) {//1
		System.out.printf("Integer=%d%n", i);
	}
	public static void viewNum(int i) {//2
		System.out.printf("int=%d%n", i);
	}
	public static void viewNum(Float f) {//3
		System.out.printf("Float=%.4f%n", f);
	}
	public static void viewNum(Number n) {//4
		System.out.println("Number=" + n);

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

```
            Number[] num =
                  {new Integer(7), 71, 3.14f, 7.2};
            for(Number n: num)
                  viewNum(n);

            viewNum(New Integer(8));
            viewNum(81);
            viewNum(4.14f); viewNum
            (8.2);
      }
}
```

Може здатися, що в результаті компіляції та виконання цього коду будуть послідовно викликані всі чотири методи, однак у консоль буде виведено:

```
Number=7
Number=71
Number=3.14
Number=7.2
Integer=8
int=81
Float=4,1400
Number=8.2
```

Тобто у всіх випадках при передачі до методу елементів масиву був викликаний четвертий метод. Це сталося внаслідок того, що вибір варіанта перевантаженого методу відбувається на етапі компіляції та залежить від типу масиву num. Те, що на етапі виконання метод передається інший тип (для перших трьох елементів масиву), не має ніякого значення, так як вибір вже був здійснений заздалегідь.

При безпосередній передачі об'єкта метод вибір проводиться залежно від типу посилання на етапі компіляції.

З одного боку, цей механізм знижує гнучкість, з іншого – усі можливі помилки при зверненні до перевантажених методів відстежуються на етапі компіляції, на відміну від перевизначених методів, коли їхній некоректний виклик призводить до виникнення винятків на етапі виконання.

При перевантаженні завжди слід дотримуватися таких правил:

- не використовувати складні варіанти перевантаження;
- не використовувати навантаження з однаковим числом параметрів;
- замінювати за можливості перевантажені методи на кілька різних методів.

Методи зі змінною кількістю параметрів

Можливість передачі метод нефіксованого числа параметрів дозволяє відмовитися від попереднього створення масиву об'єктів для його подальшої передачі метод.

/ Приклад # 14: визначення кількості параметрів методу: DemoVarargs.java */*

```
package lab2;

public class DemoVarargs {
      public static int getArgCount(Integer... args) {
            if(args.length == 0)
                  System.out.print("No arg=");
            for(int i: args)
                  System.out.print("arg:" + i + " ");
            return args.length;
      }
      public static void main(String args[]) {
        System.out.println("N=" + getArgCount(7, 71, 555));
        Integer[] i = {1, 2, 3, 4, 5, 6, 7};
        System.out.println("N=" + getArgCount(i));
        System.out.println(getArgCount());
      }
}
```

В результаті виконання цієї програми буде виведено:

```
arg:7 arg:71 arg:555 N=3
arg:1 arg:2 arg:3 arg:4 arg:5 arg:6 arg:7 N=7 No
arg=0
```

У прикладі наведено найпростіший метод із змінним числом параметрів. Метод getArgCount() виводить усі передані йому аргументи та повертає їх кількість. При передачі параметрів метод з них автоматично створюється масив. Другий виклик методу в прикладі дозволяє передати метод масив. Метод може бути викликаний без аргументів.

Щоб передати кілька масивів у метод посилання, слід використовувати таке оголошення:

```
void methodName(Тип[]... args){}
```

Методи із змінним числом аргументів можуть бути перевантажені:

```
void methodName(Integer...args) {}
void methodName(int x1, int x2) {}
void methodName(String...args) {}
```

У наступному прикладі наведено три перевантажені методи та кілька варіантів їх виклику. Відмінною рисою є можливість методу з аргументом Object... args приймати як об'єкти, а й масиви:

```
/* приклад # 15: передача масивів: DemoOverload.java */
package lab2;

public class DemoOverload {
	public static void printArgCount(Object... args) {//1
	System.out.println("Object args:" + args.length);
	}
	public static void printArgCount(Integer[]...args){//2
	System.out.println("Integer[] args: " + args.length);
	}
	public static void printArgCount(int... args) {//3
		System.out.print("int args: " + +args.length);
	}
	public static void main(String[] args) {
		Integer[] i = { 1, 2, 3, 4, 5 };

		printArgCount(7, "No", true, null);
		printArgCount(i, i, i);
		printArgCount(i, 4, 71);
		printArgCount(i);//буде викликаний
		метод 1 printArgCount(5, 7);
		// printArgCount();//невизначеність!
	}
}
```

В результаті буде виведено:

```
Object args: 4
Integer[] args: 3
Object args: 3
Object args: 5
int args: 2
```

При передачі метод printArgCount() одиничного масиву і компілятор віддає перевагу методу з параметром Object... args, оскільки ім'я масиву є об'єктною посиланням і тому зазначений параметр буде найближчим. Метод із параметром Integer[]...args не викликається, оскільки найближчим об'єктним посиланням йому буде Object[]...args. Метод з параметром Integer[]...args буде викликаний для одиничного масиву лише за відсутності методу з параметром Object...args.

При виклик методу без параметрів виникає невизначеність через неможливість однозначного вибору.

Немає також обмежень і на перевизначення подібних методів.

Єдиним обмеженням є те, що параметр виду **Тип...args** має бути останнім в оголошенні методу, наприклад:

```
void methodName(Тип1 obj, Тип2... args) {}
```

```
/*
Клас Complex
Комплексні числа широко використовуються у математиці. Вони часто
застосовуються в графічних перетвореннях,
у побудові фракталів, не кажучи вже про фізику та технічні дисципліни.
*/
class Complex {
private static final double EPS = 1e-12; // Точність обчислень

private double re, im; // Дійсна та уявна 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 getIm() { return im;}
public Complex getZ() { return new Complex(re, im); }
public void setRe(double re) {this.re = re; }
public void setIm(double im) {this.im = im; }
public void setZ(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; }

// Виведення на екран ******************************************
public void pr() {
    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, 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
     zl = new Complex(),
                z2 = новий Complex(1.5),
                z3 = новий Complex (3.6, -
                2.2), z4 = новий Complex
                (z3);
```

```
    System.out.println(); // Залишаємо порожній рядок
    System.out.print("z1="); zl.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(zl);
   System.out.print("z2/z1 = "); z3.pr();
  }
}
```

Варіанти завдання

Створити класи, специфікації яких наведені нижче. Визначити конструктори та методи setТип(), getТип(), toString(). Визначити додатково методи у класі, що створює масив об'єктів. Задати критерій вибору даних та вивести ці дані на консоль.

1. **Student**: id, Прізвище, Ім'я, По-батькові, Дата народження, Адреса, Телефон, Факультет, Курс, Група.
   Створити масив об'єктів. Вивести:
   a) список студентів заданого факультету;
   b) списки студентів для кожного факультету та курсу;
   c) список студентів, які народилися після заданого року;
   d) список навчальної групи.
2. **Customer**: id, Прізвище, Ім'я, По батькові, Адреса, Номер кредитної картки, Номер банківського рахунку.
   Створити масив об'єктів. Вивести:
   a) список покупців у алфавітному порядку;
   b) список покупців, у яких номер кредитної картки перебуває у заданому інтервалі.
3. **Patient**: id, Прізвище, Ім'я, По-батькові, Адреса, Телефон, Номер медичної карти, Діагноз.
   Створити масив об'єктів. Вивести:
   a) список пацієнтів, які мають цей діагноз;
   b) список пацієнтів, номер медичної картки яких перебуває у заданому інтервалі.
4. **Abiturient**: id, Прізвище, Ім'я, По-батькові, Адреса, Телефон, Оцінки.Створити масив об'єктів. Вивести:
   a) список абітурієнтів, які мають незадовільні оцінки;
   b) список абітурієнтів, середній бал у яких вищий за заданий;
   c) вибрати задану кількість n абітурієнтів, що мають найвищий середній бал (вивести також повний список абітурієнтів, які мають напівпрохідний бал).
5. **Book**: id, Назва, Автор(и), Видавництво, Рік видання, Кількість сторінок, Ціна, Обкладинка.
   Створити масив об'єктів. Вивести:
   a) список книг заданого автора;
   b) перелік книжок, випущених заданим видавництвом;
   c) Список книг, випущених після заданого року.
6. **House**: id, Номер квартири, Площа, Поверх, Кількість кімнат, Вулиця, Тип будівлі, Термін експлуатації.
   Створити масив об'єктів. Вивести:
   a) список квартир, що мають задану кількість кімнат;
   b) список квартир, що мають задану кількість кімнат і розташовані на поверсі, що знаходиться в заданому проміжку;
   c) список квартир, що мають площу, що перевищує задану.
7. **Phone**: id, Прізвище, Ім'я, По батькові, Адреса, Номер кредитної картки, Дебет, Кредит, Час міських та міжміських розмов.

Створити масив об'єктів. Вивести:
a) відомості про абонентів, у яких час внутрішньоміських розмов перевищує заданий;
b) відомості про абонентів, які користувалися міжміським зв'язком;
c) відомості про абонентів у алфавітному порядку.

8. **Car**: id, Марка, Модель, Рік випуску, Колір, Ціна, Реєстраційний номер.
Створити масив об'єктів. Вивести:
a) список автомобілів заданої марки;
b) список автомобілів заданої моделі, що експлуатуються більше n років;
c) список автомобілів заданого року випуску, ціна яких більше вказаної.

9. **Product**: id, Найменування, UPC, Виробник, Ціна, Термін зберігання, Кількість.
Створити масив об'єктів. Вивести:
a) список товарів для заданого найменування;
b) список товарів для заданого найменування, вартість яких не перевищує задану;
c) список товарів, термін зберігання яких більший за заданий.

10. **Train**: Пункт призначення, Номер поїзда, Час відправлення, Кількість місць (загальних, купе, плацкарт, люкс).
Створити масив об'єктів. Вивести:
a) список поїздів, що прямують до заданого пункту призначення;
b) список поїздів, наступних до заданого пункту призначення іщо відправляються після заданої години;
c) список поїздів, що вирушають до заданого пункту призначення імають спільні місця.

11. **Bus**: Прізвище та ініціали водія, Номер автобуса, Номер маршруту, Марка, Рік початку експлуатації, Пробіг.
Створити масив об'єктів. Вивести:
a) список автобусів для заданого номера маршруту;
b) список автобусів, які експлуатуються понад 10 років;
c) список автобусів, пробіг яких більше 100000 км.

12. **Airlines**: Пункт призначення, Номер рейсу, Тип літака, Час вильоту, Дні тижня.
Створити масив об'єктів. Вивести:
a) список рейсів для заданого пункту призначення;
b) список рейсів для заданого дня тижня;
c) список рейсів для заданого дня тижня, час вильоту для яких більший за заданий.

13. **House**: id, Номер квартири, Площа, Поверх, Кількість кімнат, Вулиця, Тип будівлі, Термін експлуатації.
Створити масив об'єктів. Вивести:
a) список квартир, що мають задану кількість кімнат;
b) список квартир, що мають задану кількість кімнат і розташовані на поверсі, що знаходиться в заданому проміжку;
c) список квартир, що мають площу, що перевищує задану.

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

a) відомості про абонентів, у яких час внутрішньо міських розмов перевищує заданий;
b) відомості про абонентів, які користувалися міжміським зв'язком;
c) відомості про абонентів у алфавітному порядку.

15. **Customer**: id, Прізвище, Ім'я, По батькові, Адреса, Номер кредитної картки, Номер банківського рахунку.
Створити масив об'єктів. Вивести:
a) список покупців у алфавітному порядку;
b) список покупців, у яких номер кредитної картки перебуває у заданому інтервалі.

16. **Patient**: id, Прізвище, Ім'я, По-батькові, Адреса, Телефон, Номер медичної карти, Діагноз.
Створити масив об'єктів. Вивести:
a) список пацієнтів, які мають цей діагноз;
b) список пацієнтів, номер медичної картки яких перебуває у заданому інтервалі.

17. **Customer**: id, Прізвище, Ім'я, По батькові, Стать людини, Адреса, Номер кредитної картки, Номер банківського рахунку.
Створити масив об'єктів. Вивести:
a) список покупців у алфавітному порядку;
b) список покупців, у яких номер кредитної картки перебуває у заданому інтервалі.

18. **Patient**: id, Прізвище, Ім'я, По-батькові, Стать людини, Адреса, Телефон, Номер медичної карти, Діагноз.
Створити масив об'єктів. Вивести:
a) список пацієнтів, які мають цей діагноз;
b) список пацієнтів, номер медичної картки яких перебуває у заданому інтервалі.

19. **Abiturient**: id, Прізвище, Ім'я, По-батькові, Стать людини, Адреса, Телефон, Оцінки.Створити масив об'єктів. Вивести:
a) список абітурієнтів, які мають незадовільні оцінки;
b) список абітурієнтів, середній бал у яких вищий за заданий;
c) вибрати задану кількість n абітурієнтів, що мають найвищий середній бал (вивести також повний список абітурієнтів, які мають напів прохідний бал).

20. **Phone**: id, Прізвище, Ім'я, По батькові, Стать людини, Адреса, Номер кредитної картки, Дебет, Кредит, Час міських та міжміських розмов.
Створити масив об'єктів. Вивести:
a) відомості про абонентів, у яких час усіх розмов перевищує заданий;
b) відомості про абонентів, які користувалися міжміським зв'язком;
c) відомості про абонентів у алфавітному порядку.

Приклад:

```
/*
* Для зміни цього license header, choose License Headers in Project Properties.
* Для зміни цього template file, choose Tools | Templates
* and open the template in the editor.
*/
package lab2;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
*
* @author bsu
*/
public class Lab2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
//// try {
//// RF.test("D: Test java \lab2\testRFtest.txt", "Cp866");
//// } catch (UnsupportedEncodingException ex) {
//// System.out.println("** UnsupportedEncodingException RF ex = "+ex);
//// System.exit(0);
//// }
//// catch (IOException ex) {
//// System.out.println("** IOException ex = "+ex);
//// System.exit(0);
//// }
//// System.exit(0);
        WF.test ("D: Test java lab2 test WF test.txt, Cp866");
        System.exit(0);
        stud [] GR;
        int Kstud = 5;
        GR = new stud [Kstud];
```

```
      /// String _fam, String _nam, String _snam, int _god, String _tel,
      /// String _fak, String _spech, String _grup, int _kurs) {
      GR[0]=new stud(1, "Баршак", "Владислав", "Валерійович", 'М',2001, "050****",
                  "математичний", "комп_науки", "6.1226", 2);
      GR[1]=new stud(2, "Вербицька", "Вікторія", "Володимирівна",'Ж', 2001, "066****",
                  "математичний", "комп_науки", "6.1226", 2);
      GR[2]=new stud(3, "Гемський", "Антон", "Володимирович", 'М', 2001, "066****",
                  "математичний", "комп_науки", "6.1226", 2);
      GR[3]=new stud(4, "Заржевська", "Ольга", "Ігрівна", 'Ж', 2001, "066****",
                  "математичний", "комп_науки", "6.1226", 2);
      GR[4]=new stud(5, "Кара", "Ігор", "Олександрович", 'М', 2001, "066****",
                  "математичний", "комп_науки", "6.1226", 2);
      for(int i=0; i <Kstud; i++ ){
        System.out.println(GR[i].toString());
      }
      /**************************************/
      System.out.println("**************************************");
      Grup G;
      G=new Grup(40);
      for(int i=0; i < Kstud; i++ ){
            G.add_stud(GR[i]);
      }

      for(int i=0; i < G.get_Kstud(); i++ ){
        System.out.println( G.get_stud(i).get_fam() + " "+
                            G.get_stud(i).get_nam() + " "+
                            G.get_stud(i).get_grup() + " " +
                            G.get_stud(i).get_kurs()
                           );

            }

   }
}

class mem {
   //private
   protected int id; // id menA
   //private
   protected String fam;
   //private
   protected String nam;
   //private
   protected String snam;
   //private
   protected char sex;
   //private
   protected int god;
   //private
```

```
    protected String tel;
    public mem(int _id, String _fam, String _nam, String _snam, char _sex, int _god, String
_tel ) {
        id=_id;
        fam = _fam;
        nam = _nam;
        snam = _ snam;
        sex=_sex;
        god=_god;
        tel = _tel;
    }
    @Override
    public String toString(){
        return Integer.toString(id)+ " "+fam+" "+ nam+" "+ snam + " " + sex+ " "+
                Integer.toString(god) + " " + tel;
    }

};
class stud extends mem {
    private String fak;
    private String spech;
    private String grup;
    private int kurs;
    public stud(int _id, String _fam, String _nam, String _snam, char _sex, int _god, String
_tel,
                String _fak, String _spech, String _grup, int _kurs) {
        super(_id, _fam, _nam, _snam, _sex, _god, _tel );
        fak = _fak;
        spech = _spech;
        grup = _grup;
        kurs = _kurs;
    }
    @Override
    public String toString(){
        return fak+" "+spech+" "+grup+ Integer.toString(kurs)+ " " +
                super.toString();
    }
    public stud copy_stud(){
        stud st=new stud(id, fam, nam, snam, sex, god, tel,
                    fak, spech, grup, kurs);
        return st;
    }
    public int get_id() { return id; };
    public String get_fam() { return fam; };
    public String get_nam() { return nam; };
    public String get_snam() { return snam; };
    public char get_sex() { return sex; };
    public int get_god() { return god; };
    public String get_tel() { return tel; };
```

```
    public String get_fak() { return fak; };
    public String get_spech() { return spech; };
    public String get_grup() { return grup; };
    public int get_kurs() { return kurs; };
}

class Grup {
    private stud [] gruppa;
    private int KstudMax;
    private int Kstud; // ptr to free in mass grupp
    public Grup(int _KstudMax){
        KstudMax=_KstudMax;
        Kstud = 0;
        gruppa = новий stud [KstudMax];
    }
    public boolean add_stud(stud st){
        if (Kstud >= KstudMax) return false;
        gruppa[Kstud++]=st.copy_stud();
        return true;
    }
    public int get_Kstud() { return Kstud; }
    public stud [] get_stud() {
        return gruppa;
    }
    public stud get_stud(int k) {
        if ( k >= KstudMax & k < 0) {
            System.out.println("Grup.get_Kstud: номер студент k="
                    +Integer.toString(k)+
                    "поза допустимим діапазоном");
            throw new IndexOutOfBoundsException("Grup.get_Kstud: номер студент k=" +
                                    Integer.toString(k) +
                                    "поза допустимим діапазоном");
        }
        return gruppa [k];
    }
}
//////////////////////////////////////////
class WF {
    String filename;
    File file;
    //FileWriter
    OutputStreamWriter fw;
    BufferedWriter writer;
// BufferedReader reader;
    public WF(String fm, String cp_code) throws IOException{
        filename=fm;
        try {
            file = new File(fm);
            //fw = New FileWriter(file);
```

```
            fw = new OutputStreamWriter(new FileOutputStream(file) , cp_code); //"Cp1251");
            //reader = New BufferedReader(fr);
            writer = new BufferedWriter(fw);
        }
        catch (UnsupportedEncodingException e) {
             System.out.println("*** file" + file + " not support decode = "+
                          cp_code+ "...\nex="+e);
            throw new UnsupportedEncodingException();
        }
        catch (IOException e) {
            System.out.println("*** file" + file + "not writers...");
            throw new IOException();
        }
    }
    public void wl(String txt) throws IOException {
        try {
             writer.write(txt+'\n');
        } catch (IOException e) {
            System.out.println("*** file" + file + " write error...");
            throw new IOException();
        }
    }
    public void close() throws IOException{
        writer.close();
    }

    static void test(String fw, String cp) {
        //String fn="d:/bsu/baze1.txt";
        WF on = null;
        String line;
        //String line_cp866;
        int kstr = 10;
        try {
            // TODO code application logic here
            on = new WF(fw, cp);
            for ( int i=0;i<kstr;i++){
                line=Integer.toString(i)+ " рядок.....";
                //line_cp866 = новий String(line.getBytes("UTF-8"), "Cp866");
                System.out.println(line);

                on.wl(line);
            }
            on.close();
        } catch (IOException ex) {
             System.exit(4);
        }
    }
}
//////////////////
```

```
class RF {
   String filename;
   File file;
   InputStreamReader fr;
   BufferedReader reader;
   public RF(String fm, String cp_code) throws FileNotFoundException,
UnsupportedEncodingException{
      filename=fm;
      try {
         file = new File(fm);
         fr = New InputStreamReader(New FileInputStream(file), cp_code); //"Cp1251");
         reader = новий BufferedReader(fr);
      }
      catch (FileNotFoundException e) {
         System.out.println("*** file" + file + " not found...\n ex="+e);
         throw new FileNotFoundException();
      }
      catch (UnsupportedEncodingException e) {
         System.out.println("*** file" + file + " not support decode = "+
                        cp_code+ "...\nex="+e);
         throw new UnsupportedEncodingException();
      }
   }
   public String rl() throws IOException {
      try {
         return reader.readLine();
      } catch (IOException e) {
         System.out.println("*** file" + file + " read error...");
         throw new IOException();
      }
   }
   public void close() throws IOException{ reader.close(); };
   static void test(String fn, String cp) throws UnsupportedEncodingException{
      //String fn="d:/bsu/baze1.txt";
      RF inp=null;
      try {
         // TODO code application logic here
         inp = new RF(fn, cp); //"Cp866");
      } catch (FileNotFoundException ex) {
         System.exit(4);
      }
      catch (UnsupportedEncodingException e) {
         System.exit(4);
      }
      String line;
      // String line1;
      int kstr = 0;
      try {
         line = inp.rl();
```

```
        while (line! = null) { // "Cp1251"
            // line1 = new String(line.getBytes(), "UTF-8");
            System.out.println(kstr+" "+line);
            kstr++;
            //TODO:
            line = inp.rl();
        }
    } catch (IOException ex) {
        System.exit(4);
    }
  }
}
```