

Програмування-1

Лекція 8
Основи ООП

Конструктори (1)

- Конструктор – спеціальний «метод», який викликається при створенні об'єкта для його ініціалізації
- Конструктори можна перевантажувати (**overload**)
- Відмінності конструкторів від методів:
 - ім'я конструктора співпадає з ім'ям класу (з врахуванням реєстру)
 - відсутній тип повернення. Зовсім. Навіть слово **void** не вказується
 - не може викликатись після створення об'єкта
 - не може бути перевантажений (**overload**) або перевизначений (**override**) у нащадку
 - не може бути **abstract**, **final**, **static**, ...
 - допускаються лише модифікатори області видимості (**public**, **protected**, **private**)
 - якщо конструктор не скритий – неможливо створити об'єкт (приклад – клас Math)
- Якщо жодного конструктора немає, створюється **конструктор за-замовчуванням**:
 - **область видимості – як для класу**
 - без параметрів
 - нічого не робить крім викликання конструктора базового класу без параметрів;
 - якщо в базовому класі такого немає – **помилка компіляції**

Конструктори (2)

- В першому рядку конструктора **можна**:
 - викликати **інший** конструктор **даного** класу: **this(параметри)**
 - викликати конструктор **базового** класу: **super(параметри)**
 - не викликати явно **this** або **super**
 - компілятор автоматично вставить виклик **super()** без параметрів
 - якщо в базовому класі **немає** конструктора **без параметрів**, то **помилка компіляції**
- **Не рекомендується** викликати із конструктора **нестатичні методи** **даного** класу
 - якщо у нащадку їх **перевизначити**, то можуть виникнути **проблеми**
 - **статичні** методи перевизначити неможливо, тобто з ними **проблем не буде**
- **Не рекомендується** із конструктора передавати комусь **this**
 - якщо хтось почне використовувати об'єкт до закінчення ініціалізації, можуть виникнути **проблеми**

Створення об'єктів

- В Java об'єкти створюються та зберігаються **лише у купі (heap)**
 - в Java не буває об'єктів у стеку
 - в Java **об'єкти не передаються за значенням**
 - в Java **об'єкти не передаються за посиланням**
 - в Java **об'єкти взагалі ніяк не передаються** (вони знаходяться у купі)
 - в Java передаються лише **посилання на об'єкти**
 - як і примітиви – за значенням
- В результаті створення об'єкта ми отримуємо **посилання на об'єкт**
 - можна використати лише для доступу до цього об'єкта
 - відсутня адресна арифметика, довільний доступ до пам'яті, перетворення посилання у примітив та навпаки
 - копіювання посилання не призводить до створення копії об'єкта
- Способи створення об'єктів:
 - оператор **new**
 - метод **clone()**
 - serialization
 - reflection

трошки пізніше

Створення об'єктів - оператор new

1. Якщо клас не був завантажений у пам'ять він завантажується
 - ініціалізація статичних полів, виконуються статичні блоки ініціалізації
2. Виділяється місце у пам'яті для об'єкта
3. Ініціалізація нестатичних полів та виконання нестатичних блоків ініціалізації
 - Поля ініціалізуються значеннями, вказаними в описі класу
 - Якщо значень вказано не було, то поля значеннями за замовчуванням:
 - числові поля = 0, логічні поля = false, посилання = null
4. Викликається **конструктор**, який був застосований в операторі **new**
 - Перед виконанням даного конструктора відбувається автоматичний виклик вказаного конструктора суперкласу – **super(параметри)**
 - якщо явного виклику такого конструктора немає, неявно викликається **super()**
5. Коли конструктор відпрацював, **new** повертає посилання на новий об'єкт

this

- **this** посилається на поточний об'єкт
- **Неможна** використовувати у **статичних** методах
- **Можна** використовувати в **нестатичних** методах та **конструкторах** для:
 - отримання посилання на даний об'єкт:
 - **this**
 - доступу до поля, ім'я якого перекрито областтю видимості параметра або локальної змінної:
 - **this.имяполя**
 - виклику перевантаженого конструктора даного класу:
 - **this(параметры)**

this – приклад

```
public class Student {  
    String name;  
  
    public Student(String name) {  
        this.name = name;           // область видимости  
    }  
  
    public Student() {  
        this("No Name");          // виклик іншого конструктора  
    }  
  
    public void addToGroup(Group group) {  
        group.addStudent(this);    // передача комусь посилання на себе  
    }  
}
```

super

- **super** використовується для викликів перевизначених методів та конструкторів базового класу (суперкласу)
- Виклик перевизначеного метода базового класу

super.имяМетода(параметры)

- Виклик конструктора базового класу

super(параметры)

- виклик конструктора базового класу може виконуватись лише на самому початку конструктора
- якщо параметри конструктора потребують розрахунку за громіздким алгоритмом, його можна включити у **статичний** метод :

super (myStaticMethod (параметри)) ;
або
this (myStaticMethod (параметри)) ;

super – приклад

```
class Employee {  
    String name;  
    public Employee(String name) {  
        this.name = name;  
    }  
    public String toString() {  
        return "Name:" + name;  
    }  
}  
  
class Manager extends Employee {  
    String department;  
  
    public Manager(String name, String departament) {  
        super(name);  
        this.department = departament;  
    }  
  
    public String toString() {  
        return super.toString() + " is manager of " +  
            department;  
    }  
}
```

Пакети

- Пакети призначені для організації ієрархічного простору імен класів та інтерфейсів
- Мета – уникнути конфліктів імен
- Пакети можуть включати в себе класи та інші
- На рівні файлової системи: **пакет = каталог, клас = файл**
- Щоб уникнути конфліктів імен пакетів в якості пакета верхнього рівня використовують доменне ім'я своєї компанії у зворотному порядку:
 - com.netcracker, ru.edunc, ua.kpi.nc, ...
- ім'я пакета записують маленькими буквами:
 - package.mycompany.example.com
- java-файл має починатись з:
 - **package packagename;**
- Якщо назва пакету не вказана, клас розміщується в пакеті «**за замовчуванням**». **Не робіть так!**

Імпортування класів

- Повне ім'я класу (fully qualified class name) –

назва пакету. Назва Класу

```
java.util.Date date = new java.util.Date();
```

- Щоб використовувати коротке ім'я, його потрібно імпортувати:

```
package mypackage;

import java.util.Date;

public class MyClass {
    Date date = new Date(); // Просто Date вместо java.util.Date
}
```

Імпортування класів

- Якщо потрібно декілька класів з одного пакету, можна використовувати «*»

```
import java.util.*;
```

- включаються **всі** класи **з даного** пакета
 - класи **з пакетів в даному пакеті** автоматично **не включаються**
 - це ніяк не впливає ні на розмір class-файлу, ні на швидкодію
 - import** працює на этапі компіляції
 - якщо заглянути в середину class-файлу, там завжди записані повні імена класів
- Автоматично імпортуються всі класи із пакета **java.lang**

Статичний імпорт

- Імпортуються не самі класи, а лише їх статичні методи
- JDK 5 +

```
public class Test {  
    public static void main(String[] args) {  
        double x = 0.5;  
        double y = Math.sin(x)*Math.sin(x) + Math.cos(x)*Math.cos(x);  
        System.out.println(y);  
    }  
}
```

```
import static java.lang.Math.*;  
public class Test {  
    public static void main(String[] args) {  
        double x = 0.5;  
        double y = sin(x)*sin(x) + cos(x)*cos(x);  
        System.out.println(y);  
    }  
}
```

Клас Object

Доступ	Тип	Имя
public	boolean	equals(Object obj)
public	int	hashCode()
public	String	toString()
protected	Object	clone()
protected	void	finalize()
public	Class<?>	getClass()
public	void	notify()
public	void	notifyAll()
public	void	wait()
public	void	wait(long timeout)
public	void	wait(long timeout, int nanos)

Класс Class

- Об'єкти класу `Class` містять інформацію о класах, завантажених у віртуальну машину.
- Забезпечують інформацію про класи під час виконання програми - `Reflection`
- Деякі методи:
 - `getName()`
 - `getSuperclass()`
 - `forName(String className)`
 - `newInstance()`
 - `getFields()`
 - `getMethods()`

Класи-обгортки

- Представляють значення примітивного типу у вигляді об'єкту
 - Integer
 - Byte
 - Character
 - Short
 - Long
 - Double
 - Float
 - Boolean
- Додаткові функції
 - перетворення в/із String
 - містять спеціальні константи
(MIN_VALUE, MAX_VALUE, NaN, POSITIVE_INFINITY, ...)
 - інші функції (в залежності від типу)
- Неможна змінити значення (**immutable**)

Клас String

- Представляє собою рядок символів
- Об'єкти String – **immutable!**
 - Після створення неможливо змінити стан об'єкта
- Клас String є **final**
 - Неможливо створити нащадків
- Має цілу купу корисних методів (дивись JavaDoc)
 - жоден з цих методів не змінює стан значення (**immutable**)
 - деякі методи створюють **нові** об'єкти та повертають посилання на них
- String вміє працювати з **юнікодом, локалями, регулярними виразами тощо**
- Для String оператор «+» виконує конкатенацію
- Якщо потрібно дуже багато маніпулювати рядками символів, рекомендується використовувати **StringBuffer** або **StringBuilder**

StringBuffer, StringBuilder

- **StringBuffer** и **StringBuilder** представляють собою **mutable** рядок **символів**, над яким дуже ефективно виконуються операції додавання, вставки та видалення символів
 - на відміну від **String** не створюються нові об'єкти при кожній операції
- **StringBuffer**
 - з'явився у Java 1.0
 - синхронізований (thread-safe)
 - тому працює трохи **повільніше** ніж **StringBuilder**
- **StringBuilder**
 - з'явився у Java 5
 - не синхронізований (non thread-safe)
 - тому працює трохи **швидше** ніж **StringBuffer**

Класи System, Runtime

- **System** – Багато корисних **статичних** полів та методів
 - `static` `in`, `out`, `err`
 - `static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
 - `static Console console()`
 - `static long currentTimeMillis()`
 - `static long nanoTime()`
 - `static void exit(int status)`
 - `static String lineSeparator()`
 - методи для роботи з властивостями(properties) та змінними середи (environment variables)
- **Runtime** – Багато корисних **нестатичних** полів та методів
 - для отримання посилання на об'єкт: `Runtime.getRuntime()`

Робота з датами/часом

- Клас **Date** (`java.util.Date`)
 - Частково застарілий клас (містить багато **Deprecated** методів)
 - Для JDK 8+ рекомендується використовувати новий `java.time API`
 - Коректно працює з часом у форматі «кількість мілісекунд після January 1, 1970, 00:00:00 GMT»
 - Для інших цілей краще не використовувати
 - Містить методи для перетворення в **Instant** (JDK 8+)
- Клас **Calendar**
 - Більш багатий функціонал ніж **Date**
 - Містить методи для перетворення в **Date**
 - `getTime()`, `setTime()`
 - Частково застарілий клас
 - Для JDK 8+ рекомендується використовувати новий `java.time API`
 - Містить методи для перетворення в **Instant** (JDK 8+)
 - Є підтримка локальних налаштувань часу
 - Є **абстрактним** класом (неможливо створити через `new`)
 - статичний метод `Calendar.getInstance()` повертає посилання на екземпляр конкретного класу
 - у нас – **GregorianCalendar**

Інші корисні класи

- **Math** – містить статичні методи для математичних розрахунків
 - sin, cos, log, exp, pow, random,
- **Random** – генератор псевдовипадкових величин
 - більше функцій ніж у Math.random
- **Arrays** – містить статичні утилітні методи для роботи з масивами
 - копіювання, пошук, сортування, заповнення
- **Pattern, Matcher** – регулярні вирази
 - дивись
<http://docs.oracle.com/javase/tutorial/essential/regex/index.html>
- **Locale** – робота с параметрами локалізації