Технически университет – София
Факултет по Компютърни системи и управление

ПИК 3 - Java

Семинарни упражнения

гл. ас. д-р Антония Ташева

Класове и пакети,
наследяване,
абстрактни класове и интерфейси,
полиморфизъм,
капсулация на данни

ТЕМА 3. КЛАСОВЕ И ОБЕКТИ

Класове и обекти

- ***Клас (class)*** *- наричаме набор от ограничения за достъп и описания на възможни действия, които могат да се извършват върху дефинирани променливи*
- ***Обект*** *- наричаме практически представител на даден клас, в който са „попълнени“ конкретни данни в дефинираните променливи и така вече е възможно да бъдат извършвани конкретните действия*

```
public class kidsplay{
  public static void main(String[] args){
    // Създаване на обект чрез конструктор по
подразбиране
    Kid ivancho = new Kid();
    // Създаване на обект чрез втори конструктор
    Kid maria = new Kid("Maria");
    // Извикване на методи на обектите
    ivancho.introduce();
    maria.introduce();
  }
}
```

```
class Kid{
  // Полета
  public String name;
  // Конструктор по подразбиране
  public Kid(){
    java.util.Scanner s = new java.util.Scanner(System.in
    System.out.print("Enter the kid name: ");
    this.name = s.next();
  }
  // Втори конструктор
  public Kid(String name){
    this.name = name;
  }
  // Метод на класа
  public void introduce(){
    System.out.println("Hi, my name is "+this.name);
  }
}
```

Импортиране

```
java.util.Scanner s = new java.util.Scanner(System.in);
```

```
import java.util.*
import java.util.Scanner;
...
Scanner s = new Scanner(System.in);
```

Пакети

- ***Пакет*** *наричаме множество от логически свързани класове*.

- package kids;

Статични методи

- ***Статични*** *са методите, които могат да се извикат без да е нужно да създадем инстанция на клас.*

```
public static void yell(){
   System.out.println("Hooray!!!");
 }

Kid.yell();
```

Статични полета

- Общи за всички инстанции на класа
- Статичните полета трябва да се инициализират

- public static int value = 1;

- статичен метод НЕ може да достъпва не-статично поле – компилаторът ще даде грешка (non-static variable value cannot be referenced from a static context)

Наследяване

- Възможността един клас да наследи цялата функционалност на друг клас и да я разшири.
- Класът, който е наследяван се нарича „родителски клас“, а класа който наследява се нарича „наследник“.
- Освен да добавят нова функционалност, наследниците също могат да изменят съществуваща.

Наследяване

```
package kids;
class Boy extends Kid{
  public String sex = "male";
  public Boy(){
    // извикване на конструктора на базовия клас
    super();
  }
  public Boy(String name){
    // извикване на конструктора на базовия клас
    super(name);
  }
  public void proposeGame(){
    System.out.println(this.name+" said: Let's go play football!");
  }
}
```

Абстрактни методи и класове

- Абстрактни са тези методи в клас, чиято реализация не е имплементирана. С други думи ние сме декларирали, че такъв метод трябва да има, но той все още не е описан как ще работи.
- Класове, които съдържат в себе си абстрактни методи наричаме „абстрактни класове“.
- „Шаблони“
- От тях не могат да се правят обекти!

Абстрактни класове

```
abstract class Figure{
  double dim1;
  double dim2;
  public Figure(double dim1, double dim2){
    this.dim1 = dim1;
    this.dim2 = dim2;
  }
  abstract double getArea();
}
```

Абстрактни класове

```
class Triangle extends Figure{
 public Triangle(double a, double b) {
  super(a,b);
 }
 public double getArea(){
  return (this.dim1 * this.dim2)/2;
 }
}
```

```
class Rectangle extends Figure{
 public Rectangle(double a, double b) {
  super(a,b);
 }
 public double getArea(){
  return this.dim1 * this.dim2;
 }
}
```

Интерфейс

- Интерфейсът съдържа само и единствено абстрактни методи и евентуално константи (за константи ще говорим по-късно).
- Чрез интерфейсите ние даваме само и единствено обща насоченост как трябва да изглежда даден клас, който го „имплементира“.

Интерфейс

```
interface Figure{
  public double getArea();
}

class Rectangle implements Figure{
  double dim1;
  double dim2;
  public Rectangle(double a, double b) {
    this.dim1 = a;
    this.dim2 = b;
  }
  public double getArea(){
    return this.dim1 * this.dim2;
  }
}
```

```
class Triangle implements Figure{
  double dim1;
  double dim2;
  public Triangle(double a, double b) {
    this.dim1 = a;
    this.dim2 = b;
  }
  public double getArea(){
    return (this.dim1 * this.dim2)/2;
  }
}
```

Полиморфизъм

- Понятието „полиморфизъм“ идва от биологията. То описва „съществуване на морфологично различни индивиди в границите на един вид“ (например пчели и търтеи в 1 кошер)
- В ООП под „полиморфизъм“ се разбира същото – различни обекти (индивиди) от един и същи вид (базов клас) извършват различни дейности.

Полиморфизъм

- понятието „**статичен полиморфизъм**“ като синоним на „предефиниране на метод“
- **Динамичен полиморфизъм** се получава тогава, когато предефинираме методи на базов клас в негов наследник. Казваме също, че по този начин променяме съществуващата реализация на метод с нова.
- Ето един пример – създаваме наследник и предефинираме метод print()

```
public class polyexample {
  public static void main(String[] args){
    ClassExample c = new ClassExample(5);
    c.print();
    SubClassExample c2 = new SubClassExample(5);
    c2.print();
  } }
```

```
class ClassExample{
  public int x;
  public ClassExample(){
    x = 0;
  }
  public ClassExample(int x){
    this.x = x;
  }
  public void print(){
    System.out.println(this.x);
  }
}
```

```
SubClassExample extends ClassExample{
  public int y;
  public SubClassExample(){
    super();
    y = 0;
  }
  public SubClassExample(int x){
    super(x);
    y = 0;
  }
  public void print(){
    System.out.println(this.x+" "+this.y);
  }
}
```

Динамично свързване на метод

- Това е свойството на метод да върши различни неща в зависимост от обекта с който работи.
- Динамичното свързване на метод всъщност се свързва и с общото понятие „полиморфизъм“ в обектно-ориентираното програмиране.
- В известен смисъл то „надгражда“ динамичния полиморфизъм.

```
public class polyexample{
  public static void main(String[] args){
    Animal a;
    Dog kuche = new Dog("Sharo");
    Cat kotka = new Cat("Pisana");
    a = kuche;
    a.speak();
    a = kotka;
    a.speak();
  }
}

abstract class Animal{
  public String name;
  public Animal(String name){
    this.name = name;
  }
  abstract void speak();
}
```

```
class Dog extends Animal{
  public Dog(String name){
    super(name);
  }
  public void speak(){
    System.out.println("Бау, бау!");
  }
}

class Cat extends Animal{
  public Cat(String name){
    super(name);
  }
  public void speak(){
    System.out.println("Мяу!");
  }
}
```

Капсулация на данни

- Тенденцията за „скриване на алгоритъма“ от „външния свят“ естествено се пренася и при обектно-ориентираното програмиране.
- Ключовата дума „public“ се превежда като „публичен“, а в програмирането има значение на „достъпен от всеки“. Такива данни не са „капсулирани“, т.е. „не са скрити/защитени“.
- модификатори за достъп - „private“ или „protected“
- „get“ и „set“ методи

Капсулация при наследяване

- Наследникът на класа няма достъп до private полетата и методите на супер класа (родителския клас).
- По този начин казваме, че супер класа е капсулирал своите данни – не предоставя възможност да бъдат променяни
- Ако направите едно поле „protected“, то ще бъде видимо не само от наследниците на класа (както е в C++), но и от всички други класове в текущия пакет (package).
- Ако нашата програма се разпростира само в един пакет (каквито програми сме писали досега), то public и protected ще имат едно и също действие.
- Идентификатор „default“ ~ „protected“

Модификатори на класове

- default (без модификатор) и public.
- Когато един клас е публичен (public), то той е видим от чужди пакети.
- Когато един клас се декларира без модификатор, то само и единствено класове в текущия пакет могат да правят негови инстанции или да го наследяват.
- Важно за Java правило е, че ако искате да направите един клас публичен, то той трябва да стои в самостоятелен .java файл (тоест за него ще се компилира отделен .class файл).

Константи,
под-пакети и не-статични вложени член класове,
локални класове в Java,
анонимни класове в Java,
статични вложени член класове,
Garbage Collection и метод finalize()

ТЕМА 4. ОЩЕ ЗА ООП

Константи

- Константите в Java се създават чрез ключова дума „final“.
- Както е по дефиниция – константите задължително се инициализират (може и по-късно) и след това не могат да бъдат променяни:

```
final int x = 5;
// x = 6; <<< Не е възможно
```

```
final int x;
java.util.Scanner s = new java.util.Scanner(System.in);
x = s.nextInt();
```

Константи

- Ключовата дума final важи за всякакви променливи (обекти).

```
final A obj = new A(3);
obj.x = 5; // Няма проблем - x не е константа
// obj = null;  <<< невъзможно - obj е константа!
```

- В случая когато final се ползва за дефиниця на обект, полетата му не са непроменяеми. Не може да бъде променен само адреса на променливата, която „сочи“ към този обект.

Константни методи

- **Константните методи** не могат да бъдат предефинирани в клас-наследник.

```
 final void func(){
    System.out.println("Final function!");
  }
}
```

Константни класове

- **Константните класове** пък са такива класове, които не могат да бъдат наследявани.

```
final class A{
  public A(){}
}

// cannot inherit from final A
//class B extends A{
//  public B(){ super(); }
//}
```

Под-пакети

- готови пакети с класове - **java.io** (съдържат класове работещи с входно-изходни устройства), **java.net** (класове свързани с мрежови потоци) и т.н.
- йерархия – системния пакет **java** има свой под-пакет **io**, който пък притежава клас FileInputStream.
- За да достигнем до този клас ние указваме пълният път до него – java.io.FileInputStream

Под-пакети

- променливата CLASSPATH
- **C:\myprogram>set CLASSPATH=.;C:\;**

- Под-пакетите са под-директории в директориите на някой пакет.

Важни принципи:

- Давайте уникални имена на пакетите, които пишете;
- Давайте уникални имена на класовете, които пишете .
- Кръщавайте всичко със смислени имена!

Вложени класове

- Йерархията на пакети и под-пакети всъщност много наподобява тази на класове и „**вложени класове**„.
- В езика Java ни е дадена възможност да правим следните конструкции:

```
class ВъншенКлас {
   ...
  (private)  class ВложенКлас {
      ...
    }
}
```

НЕ правим инстанции на вложени класове, а ги използваме само вътре в техните външни класове. Поради тази причина често се казва, че такива вложени класове са „помагащи класове“.

Видове вложени класове

- „вложен член клас“ – предния пример
- „статичен вложен член клас“,
- „локален клас“
- „анонимен клас“.

Локални класове

- „Локални“ класове са тези класове, които са дефинирани вътре в програмен блок.
- Досега знаем как да влагаме класове като членове на даден клас. Когато говорим за локални класове се има предвид клас, който е създаден в тялото на метод.
- Локалните класове са валидни само и единствено в програмния блок, в който са дефинирани.

Локален клас

- Пример localClassExample.java

- Не можем да правим инстанция на клас „PassGenerator“ никъде другаде освен вътре в метод „setPass“, където този локален клас е деклариран. По този начин казваме, че „PassGenerator е локален клас за метод setPass“.

Анонимни класове

- „Анонимните“ класове са специален вид локални класове. Специалното при тях е, че те нямат специални имена и на тях винаги правим само една инстанция.

```
double a = 3;
double b = 4;
double c = Math.sqrt(a*a + b*b);
System.out.println(c);
```

Защо обаче трябва да пазим променливата „c“ ако не я използваме повече в програмата след това?

```
double a = 3;
double b = 4;
System.out.println(Math.sqrt(a*a + b*b));
```

```
BufferedReader in = new BufferedReader(new FileReader("text.txt"));
```

Анонимни класове

- Пример AnonymousClassExample.java

- Виждате, че при декларирането на второто куче ние предефинирахме метод „speak“.
- Чисто функционално това е все едно, че сме наследили клас Dog в нов локален клас и сме предефинирали негов метод.
- Тъй като на този нов локален клас не сме дали формално ново име, то той се нарича „анонимен клас“.

Статични вложени член класове

Когато един вложен клас е статичен то той:

- Е общовалиден за всички инстанции на външния клас;
- Има достъп само до статичните полета и методи на своя външен клас;
- Може да бъде инициализиран без да е нужна инстанция на външния клас;
- Няма достъп до не-статичните полета и методи на своя външен клас.

Статични вложени член класове

- Пример innerClassesExample.java

- Вложените класове имат достъп до private променливите на външните им класове. Вложените не-статични класове не могат да имат статични методи (за разлика от вложените статични класове).
- Най-често в практиката ако създаваме не-статични вложени член класове, те ще са private, т.е. за вътрешно ползване от външния им клас. Обратно – по-често ако пишем вложени статични класове, то те ще са публични.

Garbage Collection

- В Java програмистът няма контрол над унищожението на обектите.
- Цялата тежест над това е прехвърлена върху т.нар. Garbage Collector.
- Съществува аналогия на „почистващ метод“ (деструктор) – това е метод „finalize()“. Именно той се извиква в момента, в който даден обект е изтрит от Garbage Collector.

Garbage Collection

- Пример GCExample.java

- Честото викане на System.gc() и System.runFinalization()“ е лоша практика, понеже „събирането на боклука“ всъщност е доста тежка процедура, която отнема много ресурси – повече от процедурата на създаване на обект

Домашни задания №2

- Варианти за работа в екип от 2-ма души
- Самостоятелни задачи

- Следващата седмица (5-та) практика
- Предаване и защита 6-та седмица!

- Контролна работа №1 – 7-ма седмица!