



Java

# Класове и обекти в Java

Лектор:  
инж. Божидар Бацов

# Контакти

- <http://devcraft.wordpress.com> – блога
- [lordbad@gmail.com](mailto:lordbad@gmail.com) - една от пощите
- [www.fosscourse.org](http://www.fosscourse.org) - проекта
- Чата на проекта в [kenai.com](http://kenai.com)

# Абонирахте ли се за пощенския списък?

- Кликнете тук - <http://fosscourse.org/static/list-subscribe.html>
- Или изпратете празно писмо тук - [fosscourse-subscribe@googlegroups.com](mailto:fosscourse-subscribe@googlegroups.com)

# Кратък преглед на домашната работа

- Решенията са качени в [kenai.com](http://kenai.com)
- Като цяло момците се представиха по-добре – пратиха доста повече решения :)

# Кратко въведение в обектно ориентираното програмиране

- Какво е ООП?
- Основни понятия в ООП
  - Класове – шаблони
  - Обекти – инстанции от шаблоните
    - Състояние (State)
    - Поведение (Behaviour)

# Пример - човек

- Клас Човек

- състояние

- име
    - възраст
    - адрес

- Поведение

- Говори
    - Ходи

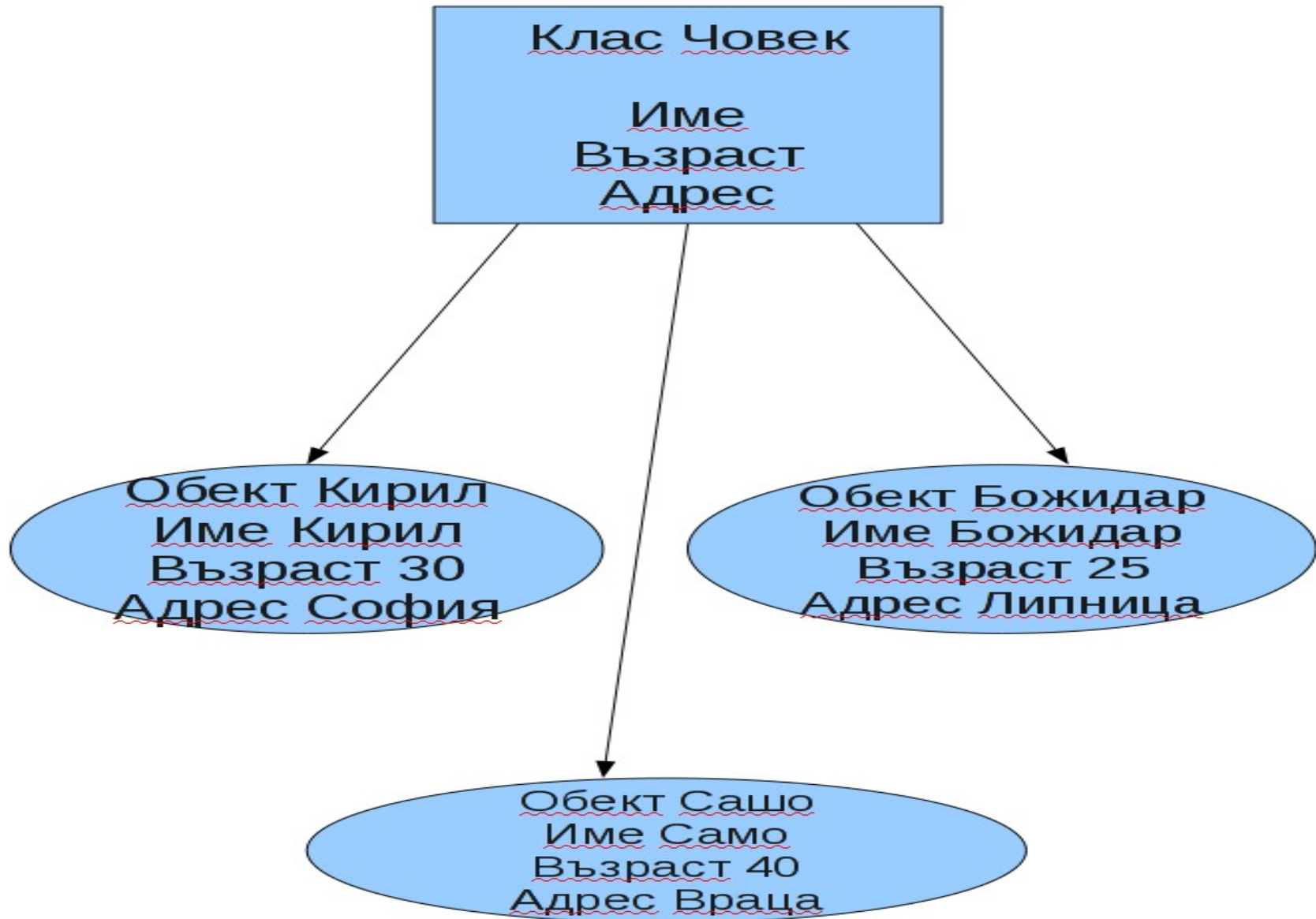
- Обект Божидар

- състояние

- Божидар
    - 33
    - София

- Поведение

- Кажи “aaa”
    - Иди някъде



# Основни понятия при класовете

- Реализация на състоянието – полета(instance fields)
- Реализация на поведението – методи(instance methods)
- Капсулиране на състоянието
- Класови състояние/поведение – статични полета и методи
- Наследяване

# Основни понятия при обектите

- Конкретно състояние – стойностите на полетата
- Поведение – операции над състоянието
- Идентичност – каква информация определя два обекта като еднакви

# Отношения между класовете

- Зависимост (coupling, dependency, uses-a) – класове използвани от класа
- Агрегация (has-a) – индикира че един обект съдържа други обекти
- Наследяване (is-a) – индикира че един тип подтип (по-специализиран тип) на друг тип

# UML

- Unified modeling language
- Показва зависимости между класовете(и много други неща) във вид на диаграми
- NetBeans има вградена поддръжка на UML

# Създаване на обекти

- Конструктор – специален метод, който инициализира състоянието на обект
- Конструктора има същото име като името на класа
- Обръщения към конструктора трябва да се предшествани от ключовата дума “new”

# Създаване на обекти

```
Date currentDate; //not initialized
currentDate = new Date(); //create a new date
//object
System.out.println(currentDate.toString());
anotherDate = currentDate; // a second
reference to the first date
```

# Обености при променливите, когато работим с обекти

- Променливите НЕ са обекти
- Променливите НЕ съдържат обект
- Променливите съдържат само препратка(референция, адрес) на обект
- Няколко променливи могат да сочат към един обект
- Променливите не са инициализирани с null по подразбиране

Пример в NetBeans

# Класове от стандартната библиотека

- `Date` – представя дата
- `Random` – генератор на случайни числа
- `Scanner` – вход от поток
- `System` – агрегатор на системни операции
- `GregorianCalendar` – представя дата по грегорианския(нашия) календар

Пример в NetBeans

# Класът String

- Представя низ от символи, кодирани в Unicode-16
- Непроменими инстанции
- специален опростен синтаксис
  - Създаване на обект
    - `String someString = "string";`
  - Конкатениране(сливане) на низове
    - `String sum = "first " + "second";`

Пример в NetBeans

# Дефиниране на класове

```
class ClassName {  
    field1;  
    field2;  
    ...  
    constructor1;  
    constructor2;  
    ...  
    Method1;  
}
```

# Разработка на класа Person

- Полета
  - Първо име
  - Фамилия
  - Дата на раждане
  - Адрес
- Методи – конструктори, методи за манипулиране на полетата(setter/getter), говори, отиди някъде

Пример в NetBeans

# Работа с няколко сорс файла в Java

- Java компилатора разбира зависимостите между класовете
- Ако клас А зависи(използва) от клас Б – командата:
  - `javac A.java`ще компилира и класа Б, ако не е вече компилиран
- Вградена функционалност тип “make”

# Изследване на класа Person

- Всички полета са маркирани с модификатора за достъп “private”
- Private ограничава достъпа до полетата само до рамките на класа, в който те са дефинирани
- Осигурява се отлична капсулация на данните и достъпа до тях се контролира надеждно чрез методи

# Изследване на класа Person

- Всички методи са маркирани с модификатора за достъп `public`
- `Public` маркира програмен елемент като достъпен за ВСИЧКИ класове
- Почти никога не е добра идея полета да бъдат маркирани като `public` – това нарушава капсулацията на данните – основен принцип на ООП

# Представяне на обект като String

- Полезна техника за търсене на грешки
- Метод toString()
- Генериране на метода toString() с NetBeans

# Конструктор

- има същото име като класа
- Не връща резултат
- Клас може да има повече от един конструктор
- Конструктора може да приема различен брой аргументи
- Конструктора може да бъде извикан само с ключовата дума “new”

# Видове параметри при конструкторите

- Експлицитни – стойността им се използва директно за инициализиране на поле
- Имплицитни – стойността им се използва като база за изчисляването на стойността, която да се зададе на поле

# Предимства на енкапсулацията

- Валидиране на данни
- Защитаване на данни от промени
- Гъвкавост – възможно е да бъде променена вътрешна имплементация без промени на публичното API

## Допълнение за private

- Private членовете от всеки обект на класа са достъпни в неговите методи. Например:

```
class Person {  
    public int compare(Person another) {  
        if (this.name.equals(another.name)) ...  
    }  
}
```

## Добрият стил диктува...

- Само методите, които ще бъдат използвани от клиентски код да бъдат декларирани публични – допълнителните (utility) методи е желателно да бъдат private

# Модификатора `final`

- Приложен към примитивен тип – има смисъл на константа
- Приложен към обект – има смисъл на константна референция, самия обект може да бъде променен
- Приложен към клас – ненаследим клас
- Приложен към метод – метод, който не може да бъде `overridden`

# Статични(класови) полета и методи

- Асоциирани са със самия клас, а не обектите инстанцирани от него
- Достъпни са както посредством класа, така и посредством обектите от него
- Статични полета често се използват за съхранение на константи
- Статични методи често се използват за реализация на фабрични методи

Пример в NetBeans

# Параметри на методите

- Винаги се придават по стойност(pass by value)
  - При примитивните типове се предава самата стойност на аргумента
  - При референтни типове(обекти) се предава препратка към обекта. Самата препратка е препроменяема, но обекта указан от нея е.
- Прототип на метод

Пример в NetBeans

# Конструиране на обекти

- Винаги посредство конструктор
- Ако не декларирате конструктор, компилатора ще създаде такъв
- Всички полета се инициализират по подразбиране – числените типове на 0, обектите на null, boolean на false
- Може да имате повече от един конструктор

# Презареждане на методи

- В един клас може да имате повече от един метод с едно и също име, стига сигнатурата на методите да е различна
- Сигнатурата включва само името на метода и неговите параметри, но не и типа на връщаната стойност
  - `void method(int x, int y);`
  - `void method(int x);`

# Детайли за конструкторите

- Извикване на един конструктор от друг
- Конструктора по подразбиране
- Конвенции за именуване на параметрите на конструкторите
- Private конструктори

# Пакети в Java

- Позволяват групирането на свързани класове
- Предовратяват конфликти в имената на класовете – приблизителен еквивалент на namespaces в C++/C#
- Желателно е да са уникални – например да започват с интернет домейн
- Запазени пакети - java.\*

# Импортиране на класове и пакети

- Един клас може да ползва всички класове в своя пакет и всички публични класове
- Публичните класове трябва да бъдат импортирани или квалифицирани с пълните си имена(пакет + клас) за да бъдат достъпни
- За да бъдат използвани класовете от текущия пакет не е нужно импортиране

# Статично импортиране

- Импортиране само на определени статични членове от клас – полета или методи
- Подходящо за “статични” класове като Math
- Трябва да използва много внимателно – в противен случай нарушава четимостта на програмата

# Добавяне на клас към пакет

```
package com.drowltd.project;  
class Person {  
    ...  
}
```

# Някои особености за пакетите

- Всяка част от името на пакета, разделена с точно, съответства на поддиректория в кода на програмата
  - `com.fosscourse.java`
  - `com/fosscourse/java`
- Не съществува йерархична организация на пакетите – независимо от сродство в имената всеки пакет е независим от другите

# Пакетиране на Java код

- Jar(java archive) – bytecode + мета информация, компресирани с zip
- Добавяне на jar в клас пътя
- Изпълнение на код от jar

```
java -jar jarfile.jar
```

# Клас пътя в Java(classpath)

- Пътят в който виртуалната машина търси класове при изпълнение на програмата
- Може да се задава като аргумент на виртуалната машина или като променлива на средата
- Източник на много проблеми :)

# Пример за classpath

В UNIX(Linux, Solaris, BSD)

```
java -cp .:dir2:dir3 MyClass
```

```
export CLASSPATH=.:dir2:dir3
```

В Windows:

```
Java -cp .;dir2;dir3; MyClass
```

```
set CLASSPATH=.;dir2;dir3
```

# Документиране на класове с javadoc

- Разглеждане на javadoc документация
- Документират се публични и protected елементи – пакети, класове, методи и полета
- Стандартни javadoc анотации - @author, @version, @since, @parameter, @return, @throws, @deprecated, @see
- Работа с инструмента javadoc

# Работа с javadoc

- `javadoc -d docdir package` – създава html документация за пакета `package` в директорията `docdir`
- `javadoc -d docdir package1, package2` – създава документация за няколко пакета

# Седем прости правила за създаването на добри класове

1. Винаги дръжте полетата private
2. Винаги инициализирайте полетата
3. Не използвайте много основни типове
4. Не слагайте винаги setter/getter методи
5. Дефинирайте класовете по стандартен начин

# Седем правила за създаването на добри класове

6. Раздробявайте класовете, които имат много отговорности

7. Избирайте смислени имена на класове,  
Методи и полета

# Домашна работа

Създайте прост календар с помощта на  
класа `GregorianCalendar`

# October 2009

Su Mo Tu We Th Fr Sa

				1	2	3
4	5	6	7	8	9	10
11	12	13*	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

# Домашна работа

Създайте класа Employee. Нека той има полета име, стаж, позиция и заплата. Нека

Класа има два конструктора – един по подразбиране и един, който инициализира целия обект. Нека класа има метод “increaseSalary”, който има два варианта – с параметър сума и параметър процент.

Създайте и проста програма, която използва класа

# Домашна работа

Разширете класа `Employee` с брояч на броя на инстанциите от него. Създайте 10 служителя и ги съхранете в масив.

Обходете масива с цикъл `foreach` и за служители със стаж повече от 1 година задайте случайно повишение на заплатата между 10 и 20 процента. Разпечайте служители преди и след повишенията.