ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Вовед во Java

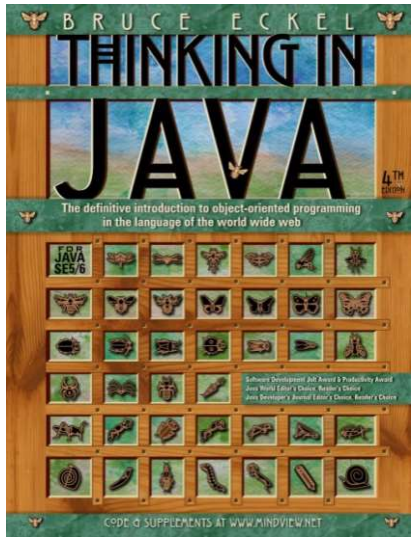
Алгоритми и податочни структури

Аудиториска вежба 0

Зошто Java?

- Развиена од Sun Microsystems, 1995 (James Gosling)
- Објектно-ориентиран јазик за општа намена
- Се базира на C/C++
- Дизајниран за лесни Web/Internet апликации
- Широко распространет
- Нуди независност на платформа (ОС)

Литература



- Thinking in Java, 4th edition
  - by Bruce Eckel
  - http://mindview.net/Books/TIJ4

Web resources…

- http://www.apl.jhu.edu/~hall/java/

Особини на Java

- **Едноставен**
  - Решава дел од проблемите на C++
  - Нема покажувачи
  - Автоматски garbage collection
  - Богати предефинирани класни библиотеки
    http://java.sun.com/j2se/1.4.2/docs/api/
- **Објектно ориентиран**
  - Се фокусира на податоците (објектите) и методите (функциите) кои манипулираат со податоците
  - Сите функции се поврзани со објекти
  - Скоро сите податочни типови се објекти (датотеки, стрингови, итн.)
  - Потенцијално подобра организација на код и реискористливост

Недостатоци на Java

- **Поспор е од компајлираните јазици како C**
  - Експерименти покажуваат дека Java е 3 или 4 пати поспор од C или C++

    чумајме: "Comparing Java vs. C/C++ Efficiency Issues to Interpersonal Issues" (Lutz Prechelt)
  - Погоден е за повеќето апликации, освен за оние кои се временски критични

Развојни околини

- **За да вршите развој на софтвер во Java потребно е да имате инсталирано** Sun Java Development Kit (JDK)
- **Постојат многу околини кои даваат поддршка за развој на софтвер во Јава, вклучувајќи ги:**
  - Sun NetBeans
  - IBM Eclipse, GNU Eclipse
  - IntelliJ
  - Borland JBuilder
  - BlueJ
  - jGRASP

  } Инсталирани се во лабораториите на ФИНКИ
- **Иако овие околини се разликуваат во деталите, основниот процес на компајлирање и извршување во суштина е идентичен**

Eclipse

- Eclipse е графичко IDE развиено од страна на Eclipse Foundation
- Цел код за Eclipse е напишан во Java
- Eclipse е развиен како платформа не само за Java програмскиот јазик, туку за т.н plug-ins (приклучоци) (еден од нив е самата Java)

Лесна инсталација

- Може да го спуштите од www.eclipse.org
- Препорачливо е да се спушти Eclipse IDE for Java EE Developers
- Најновата верзија е Eclipse 4.9 (SimRel)
- Внимание!
- Пред да се инсталира Eclipse потребно е да имате инсталирано JDK (Java Development Kit)

Лесна инсталација

- Нема потреба од инсталација
- Само unzip и го стартувате
- На првиот почеток Eclipse ќе ви побара да ја наведете локацијата за вашиот работен директориум (workspace directory) каде што ќе се чуваат сите ваши проекти и датотеки

Работа со Eclipse

- Како да напишете и извршите Java код во Eclipse во 5 чекори:
  - Направете нов проект (Java Project)
  - Додадете нова класа на проектот (File->New->Class)
  - Напишете го кодот
  - Комајлирањето е автоматски, нема потреба вие тоа да го правите
  - Стартувајте го проектот (Run->Run…)
  - Најдете ги грешките (Run->Debug…)

Прва програма во Java

```
public class Zdravo
   {
      public static void main(String[] args)
      {
         System.out.println("Zdravo studenti!\n");
      } // kraj na metodot main
   } // kraj na klasata Zdravo
```

- public static void main(String[] args) е дел од секоја Јава програма
- Програмата започнува со извршување во main
- Малите загради укажуваат дека main е метод
- Јава апликациите содржат еден или повеќе методи
- Само еден метод може да се вика main

ВНИМАВАЈТЕ:

JAVA е чувствителна на големи и мали букви т.е.
System != system != SyStEM

Важно за еден Java код

- Во една Јава програма МОРА да има **ЕДНА јавна класа** (public class)

- **public static void main (String [ ] a)** е задолжителен метод во **јавната класа ,** во кој се пишува кодот

Пр. public class test **{**
public static void main (String [] a)
{
овде се пишува кодот
} //крај на методот
} //крај на класата

Се е објект

- Во Java, се е објект, па дури и програмите
- C++ е хибриден јазик
  - Компатибилен со C
- Во Java, единствен начин на програмирање е објектно – ориентираното програмирање
- Објект се креира со помош на клучниот збор “new”

```
String s = new String(“Hello!”);
```

Податочни типови

- Основни (примитивни) vs. Референцирачки податочни типови
- Основни, примитивни типови
  - boolean, byte, char, short, int, long, float, double
  - сите тие имаат подразбирливи вредности
- Непримитивните податочни типови во Јава се објектите и низите (array).
  - Тие често се нарекуваат “референцни типови” бидејќи до нив се достапува преку референца (by reference)
  - Подразбирлива вредност на секоја референцна променлива е null

Уништување објекти

- Нема потреба од уништување на објектите !!!!
- Garbage Collection се грижи за тоа
  - JBM означува даден објект како „ѓубре кое треба да се собере“ во моментот кога кон тој објект нема повеќе референци
- Јава програмерот е ослободен од должноста да ја управува меморијата

Правила

- Сите правила што важат за објектно-ориентираната парадигма, важат и во Java
- Потсетете се што сте учеле во курсот ООП!!!
- Правилата се исти, само синтаксата на јазикот е различна.

Пакети во Java

- Java Application Programming Interface (API)
  - Исто така познато и како Јава Библиотека со Класи (анг. Java Class Library)
  - Содржи предефинирани методи и класи
    - Сродните класи се организирани во пакети (анг. packages)
    - Вклучува методи за математички пресметки, манипулација со знаци/стрингови, влез/излез, бази на податоци, мрежно работење, процесирање на датотеки, проверка за грешки, итн.
- Колекција од класи и методи кои што ви ги нуди Јава API (https://docs.oracle.com/javase/8/docs/api/)

Влез/излез во Java

- Наредба за излез на екран:

```
System.out.println("Java programming is interesting.");
```

- Печатење променливи(објекти):

```
Integer number = 10;
System.out.println("Number = " + number);
```
Дјдфх

```
String s = new String("I'm");
System.out.println(s + " FINKI student");
```
Дјдфх

Влез/излез во Java

- Со користење на класата Scanner

```
import java.util.Scanner;
```

- Се креира објект од класата Scanner и истиот се користи за да се земе влезот од корисникот

```
Scanner input = new Scanner(System.in);
int number = input.nextInt();
```

Дјдфх

- По завршување на читањето се повикува метод close() за да се затвори објектот

Влез/излез во Java

- Пример: Да се прочита целобројна вредност внесена од корисник.

```
import java.util.Scanner;
 class Input {
      public static void main(String[] args) {
            Scanner input = new Scanner(System.in);

            System.out.print("Enter an integer: ");
            int number = input.nextInt();
            System.out.println("You entered " + number);

            // closing the scanner object
            input.close();
      }
}
```

Влез/излез во Java

- Пример: Да се прочита влез даден во една линија (сè до знакот за enter)

```
import java.util.Scanner;
 class Input {
      public static void main(String[] args) {
            Scanner input = new Scanner(System.in);
            System.out.print("Enter your name: ");

            // reads the entire line
            String value = input.nextLine();
            System.out.println("Using nextLine(): " + value);

            // closing the scanner object
            input.close();
      }
}
```

Контрола на тек во Java

Пример 1: Да се испечати бројот на парни и непарни броеви како и нивниот просек од дадена влезна низа со броеви.

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

   int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};
   int sum = 0, odd = 0, even = 0;
   Double average;

   // access all elements using for loop
   for (int i=0; i<numbers.length; i++) {
     sum += numbers[i];
     if(numbers[i]%2==0)
        even++;
     else
        odd++;
   }

   // get the total number of elements
   int arrayLength = numbers.length;

   // calculate the average
   // convert the average from int to double
   average =  ((double)sum / (double)arrayLength);

   System.out.println("Number of even numbers is " + even + " and number of odd
       numbers is " + odd);
   System.out.println("Average = " + average);
 }
```

Структурата for-each

```
// print array elements

class Main {
  public static void main(String[] args) {

    // create an array
    int[] numbers = {3, 9, 5, -5};

    // for each loop
    for (int number: numbers) {
      System.out.println(number);
    }
  }
}
```

Структури за повторување

*Структура на **for** циклус:*

```
for( <initialization> ; <condition> ; <statement> ){
        <Block of statements>;
}
```

*Структура на **while** циклус:*

```
while(<boolean condition>){
        <Block of statements>;
}
```

*Структура на **do-while** циклус:*

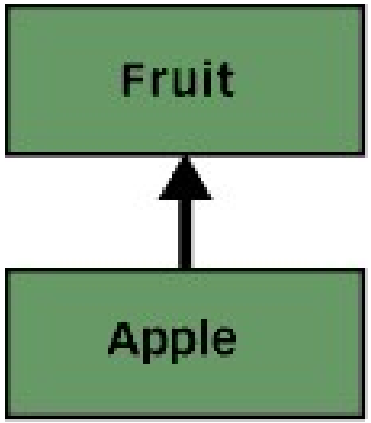
```
do{
        <Block of statements>;
}while(<boolean condition>);
```

Наследување - потсетување

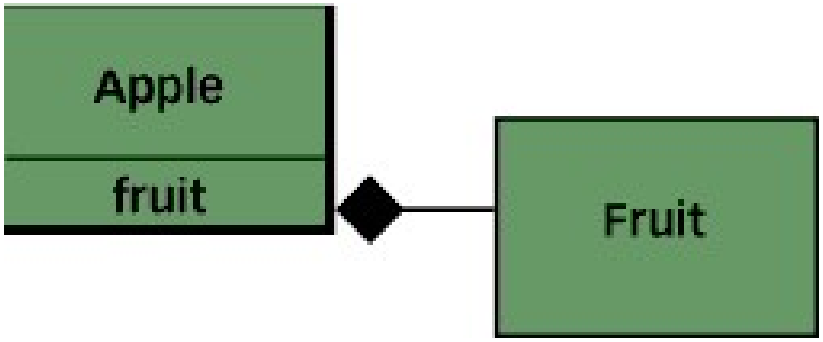
- Класата Б ја наследува класата А ако објектите на класата Б ги имаат сите карактеритики на класата А, дополнети со сопствени карактеристики
- Во Јава се вели дека **поткласата** ја **проширува** суперкласата.
- Наследност: “е”-релација

“is-a” наспроти “has-a”

- Една лоша употреба на наследноста е мешањето на релацијата has-a со релацијата is-a
- has-a значи дека една класа вклучува објекти од друга класа како атрибути



Наследност



Композиција

Разлики во Java

- **Терминологија**
  - **Суперкласа,** - А (основна, родителска класа)
  - **Потклаcа** – Б (изведена класа)

- Во Јава
  - Единечна наследност – дрво на хиерархија
  - НЕМА повеќекратно наследување
    - Кога класа наследува од повеќе различни класи
    - Се решава со Interface

Класна хиерархија

- Поткласата наследува од суперкласата, но додава и свои особености
  - Додава свои променливи
  - Додава свои методи
- Директна суперкласа
  - Поткласите се веднаш под суперкласата
- Индиректна суперкласа
  - Е секоја суперкласа која не е директно поврзана со поткласата во **класна хиерархија**

Основна класа во Java

- Класната хиерархија ги задава релациите помеѓу класите
- Класната хиерархија започнува со класата **Object** (во пакетот java.lang)
  - .. Од која СЕКОЈА класа во Јава е наследена (директно или индиректно)
- За надкласа на дадена класа Јава компајлерот ја поставува класата `Object` во случај кога декларацијата на класата не наведува (експлицитно) дека се наследува некоја друга класа.

Класна хиерархија - пример

```java
// base class
class Bicycle {
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int gear, int speed)
    {
        this.gear = gear;
        this.speed = speed;
    }

    public void applyBrake(int decrement)
    {
        speed -= decrement;
    }

    public void speedUp(int increment)
    {
        speed += increment;
    }

    // toString() method to print info of Bicycle
    public String toString()
    {
        return ("No of gears are " + gear + "\n" + "speed of bicycle is " + speed);
    }
}
```

Класна хиерархија - пример

```
// derived class
class MountainBike extends Bicycle {

    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int gear, int speed, int startHeight)
    {
        // invoking base-class(Bicycle) constructor
        super(gear, speed);
        seatHeight = startHeight;
    }

    public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }

    // overriding toString() method of Bicycle to print more info
    @Override
    public String toString()
    {
        return (super.toString() + "\nseat height is " + seatHeight);
    }
}
```

Класна хиерархија - пример

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

        Bicycle b = new Bicycle(1, 60);
        System.out.println(b.toString());
        MountainBike mb = new MountainBike(3, 100, 25);
        System.out.println(mb.toString());
    }
}
```

Што се генерици

- Генериците овозможуваат типовите (класи и интерфејси) да бидат параметризирани во фазата на нивната дефиниција
- Како и формалните параметри кои се користат во декларацијата на методите, параметрите на податочни типови (генериците) овозможуваат реискоритливост на кодот со различни влезни вредности

Генерички типови

- Генерички тип е генеричка класа или интерфејс кои имаат параметри за типовите податоци
- На пример:
  - LinkedList<E> има параметар E кој што ги претставува типовите на елементи кои се наоѓаат во поврзаната листа

Зошто генерици?

- Посилна проверка на типовите податоци за време на компајлирање
  - Поправање грешки при компајлирање е полесно отколку грешки во време на извршување (што е потешко да се најдат)
- Елиминација на кастирање
- Можност за имплементирање на генерички алгоритми
  - Алгоритмите ќе можат да работат над колекција од различни податочни типови и истите ќе бидат полесни за читање

Пример 1

- Кодот без генерици бара кастирање

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0); //casting
```

- Код со генерици е чист код

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);   // no cast
```

Пример 2

- Кодот без генерици не е сигурен на типови податоци

```
Vector v = new Vector();
v.add(new String("hello"));
v.add(new Integer(5));
// ClassCastException occurs during runtime
String s = (String) v.get(1);
```

- Код со генерици

```
Vector<String> vs = new Vector<String>();
vs.add(new Integer(5));  // compile error!
vs.add(new String("hello"));
String s = vs.get(0);
```

Именска конвенција

- По конвенција, имињата на параметарските типови се единечни големи букви
- Најупотребувани имиња на параметарски типови се:
  - E - element
  - K - key
  - N - number
  - T - type
  - V - value
  - S, U, V – 2nd, 3rd, 4th types

Пример

- Едноставна класа `Box`

```
public class Box {
    private Object object;

    public void set(Object object){this.object = object;}
    public Object get() { return object; }
}
```

- Генеричка верзија на класата `Box`

```
public class Box<T> {
  // T stands for "Type"
  private T t;

  public void set(T t) { this.t = t; }
  public T get() { return t; }
}
```

Инстанцирање објект

- Едноставна класа `Box`

```
public class Box {
    private Object object;

    public void set(Object object){this.object = object;}
    public Object get() { return object; }
}
```

```
Box b = new Box();
```

- Генеричка верзија на класата `Box`

```
public class Box<T> {
  // T stands for "Type"
  private T t;

  public void set(T t) { this.t = t; }
  public T get() { return t; }
}
```

```
Box<Integer> intBox = new Box<Integer>();
```

Повеќе параметарски типови

- Генеричка класа може да има повеќе параметарски типови
- На пример, нека имаме генеричка класа `OrderedPair` која што имплементира генерички интерфејс `Pair`

```
public interface Pair<K, V> {
      public K getKey();
      public V getValue();
}
```

Повеќе параметарски типови

```
public class OrderedPair<K, V> implements
Pair<K, V> {
    private K key;
    private V value;

    public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey(){ return key; }
    public V getValue() { return value; }
}
```

- Две инстанци на класата `OrderedPair`

```
OrderedPair<String, Integer> p1 = new OrderedPair<>("Even", 8);
OrderedPair<String, String>  p2 = new OrderedPair<>("hello", "world");
```

Генерици и подтипови

- Вие може да го напишете ова:

```
Number someNumber = new Number();
Integer someInteger = new Integer(10);
someNumber = someInteger;   // OK
```

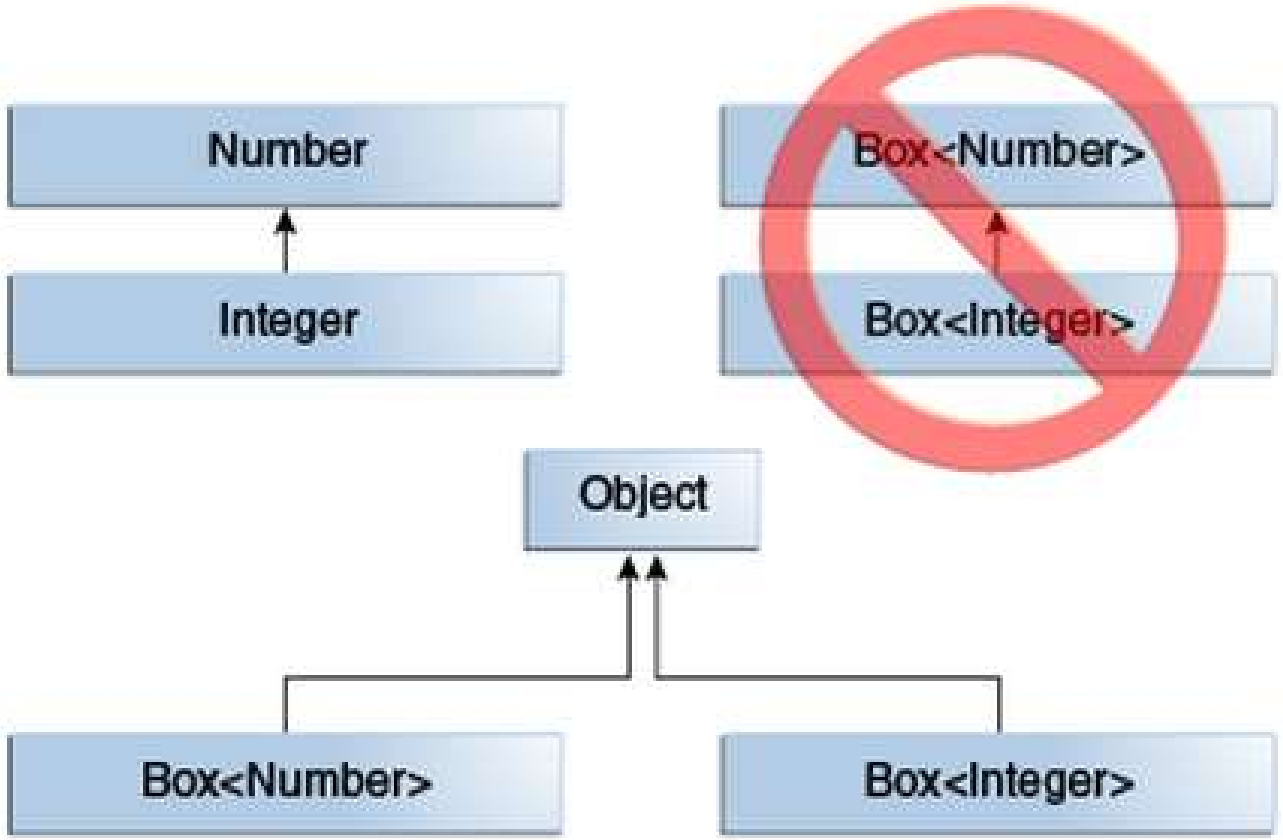
  - Според принципите на ООП, `Integer` е подкласа (подтип) на `Number`
- Според ова, вие ќе очекувате да го напишете истото со генерици

```
Box<Number> box = new Box<Integer>();
```

  - но ова не може вака!

Генерици и подтипови

- Не постои наследување помеѓу типовите аргументи кај генеричките класи



Генерици и подтипови

- вредности на колекциите од елементи познаваат наследување меѓу себе
- Така што сега, овој код е валиден

```
ArrayList<Number> an = new ArrayList<Number>();
an.add(new Integer(5)); // OK
an.add(new Long(1000L)); // OK
```

  - Но, не може

```
an.add(new String("hello")); // compile error
```

Пример со генерици

```
// create a generics class
class GenericsClass<T> {

  // variable of T type
  private T data;

  public GenericsClass(T data) {
    this.data = data;
  }

  // method that return T type variable
  public T getData() {
    return this.data;
  }
}

class Main {
  public static void main(String[] args) {

    GenericsClass<Integer> intObj = new GenericsClass<>(5);
    System.out.println("Generic Class returns: " + intObj.getData());

    GenericsClass<String> stringObj = new GenericsClass<>("Java Programming");
    System.out.println("Generic Class returns: " + stringObj.getData());
  }
}
```