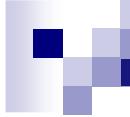


Programming

Basic building blocks 2nd part



Summary – classes and objects

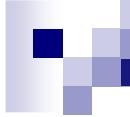
■ Classes

- Declaration – architecture
- Variables – properties
- Functions – methods
- Constructors

■ Objects

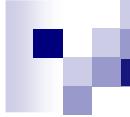
- Life cycle of an object:
 - How to create, use, destroy an object?
 - Garbage collector (JAVA)

■ Examples



Objects - things

- objects vary in their characteristics (properties),
- properties also determine what you can do with objects - we have methods to alter the properties and to inquire about properties,
- objects (predmeti):
 - store **data**: property values; eg. attribute color may have a value: green, blue, red, ...
 - provide methods for working with these data; eg. changeColor()



Classes

- Objects with the same characteristics are grouped into classes: Class is (roughly speaking) a set of objects with the same properties,
 - all objects in the class have the same properties, eg. for all we can talk about color,
 - all the objects do not share the same value for a property; eg. not necessarily have to be all green



Objects

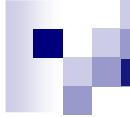
- First we have to define (describe) features of the class of objects – **define a class**:

```
public class Triangle { ... }
```

- Then we create (instantiate) one representative of the class - make one object,
- in Java do so by using **new**:

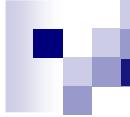
```
object = new class(parameters);
```

```
abc = new Triangle (3, 4, 5);
```



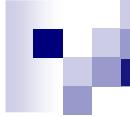
Objects

- The object has its life cycle:
 - It is born (made up)
 - Its properties can be queried or changed (used by)
 - eventually it passes away (we destroy it)



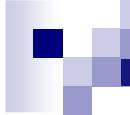
Classes

- In the definition of a class methods are grouped:
 - tvorjenje / uničevanje (*create / destroy*)
 - poizvedovanje (*query*)
 - spreminjanje (*update*)
 - (zasebne) lastnosti in metode (*private*)



Special method `main`

- there may be a method called `main` in the definition of a class;
- method `main` is started as the first method
 - example of use: the main method of our program
- there are other specific methods (Finalize, etc.).



A class example razreda – Triangle

- We will define a triangle class, which will have properties:
 - the constructor will determine the length of triangle sides, which we can later change;
 - we want to calculate the perimeter;
 - we want to know how many triangles were made (exist)



A class example razreda – Triangle

```
public class trikotnik {  
    protected int a, b, c; // sides of (this) trikotnik  
    static protected  
    int stTrikotnikov= 0; // number of. trikotnik
```

- construct / destroy
- query
- set
- (private) properties and methods

A class example razreda – **Triangle**

```
/* ----- */  
/* -----[ construct / destruct ]--- */  
public trikotnik(int an, int bn, int cn) {  
    a= an; b= bn; c= cn; stTrikotnikov++;  
}; // (tvoritelj) trikotnik  
public trikotnik() {  
    a= b= c= 0; stTrikotnikov++;  
}; // trikotnik
```

- The length of the sides will be set at construction time, can be changed later
- perimeter
- Number of “live” triangles (trikotnik)

- construct / destroy
- query
- set
- (private) properties and
- methods

A class example razreda – Triangle

```
/* ----- */  
/* ----- [ query ]--- */  
public int obseg() { return a+b+c; };  
public int narejenihTrikotnikov() {  
    return stTrikotnikov;  
}; // narejenihTrikotnikov
```

- The length of the sides will be set at construction time, can be changed later
- perimeter
- Number of “live” triangles (trikotnik)

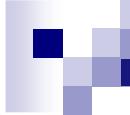
- construct / destroy
- query
- set
- (private) properties and
- methods

A class example razreda – Triangle

```
/* ----- */  
/* ----- [ spremjanje ]--- */  
public void aNajBo(int noviA) { a= noviA; };  
public void bNajBo(int noviB) { b= noviB; };  
public void cNajBo(int noviC) { c= noviC; };  
  
} // trikotnik
```

- The length of the sides will be set at construction time, can be changed later
- perimeter
- Number of “live” triangles (trikotnik)

- construct / destroy
- query
- set
- (private) properties and
- methods



An example – Student (študent)

- An example, the **student** containing your data.
- What are the characteristics of a student?
 - **name**, input at construction time;
 - **year**
 - **registration number**, which is assigned at construction;
 - to be able to display its data.

Class Student

```
public class Student {  
    /* -----[ data ]--- */  
    private String mIme;  
    private String mPriimek;  
    protected int mLetnik= 0;  
    private int mVpisnaStevilka= 0;  
    /* -----[ create / destruct ]--- */  
    public student(String ime, String priimek);  
    /* -----[ update ]--- */  
    public void vpis(int vpisnaStevilka);  
    /* -----[ query ]--- */  
    public String ime(void);  
    public String priimek(void);  
    public int letnik(void);  
    public int vpisnaStevilka(void);  
    public void izpisi(void);  
} // student
```

Class Student

```
/* -----[ create / destruct ]--- */
public student(String ime, String priimek) {
    mIme= ime; mPriimek= priimek;
    mLetnik= 0; mVpisnaStevilka= 0;
}
/* -----[ update ]--- */
public void vpis(int vpisnaStevilka) {
    mVpisnaStevilka= vpisnaStevilka;
    letnik= 1;
}
```

Class Student

```
/* -----[ query ]--- */
public String ime(void) {
    return mIme;
}
public String priimek(void) {
    return mPriimek;
}
public int letnik(void) {
    return mLetnik;
}
public int vpisnaStevilka(void) {
    return mVpisnaStevilka;
}
```

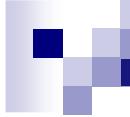


Class Student

```
public void izpisi(void) {  
    System.out.print(mIme + " " + mPriimek);  
    System.out.print(" (" + mVpisnaStevilka + ")");  
    System.out.print(" - " + letnik + ":");  
};
```

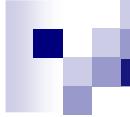
Class Student – usage

```
public class uporaba {  
    public static lepoIzpisi(student kdo) {  
        kdo.izpisi(); System.out.println();  
    }; // lepoIzpisi  
  
    public static void main(String args[]) {  
        student martin= new student("Martin", "Krpan");  
        student peter= new student("Peter", "Klepec");  
        student pehta= new student("Botra", "Pehta");  
  
        lepoIzpisi(martin); lepoIzpisi(peter); lepoIzpisi(pehta);  
        martin.vpis(123); peter.vpis(124); pehta.vpis(125);  
  
        lepoIzpisi(martin); lepoIzpisi(peter); lepoIzpisi(pehta);  
    }; // main  
}; // uporaba
```



A student has an exam

- After enrollment, students in the first year have different exams.
- The objects are dependent on the program: some students have Systems 1 and others Discrete Mathematics.
- The students are automatically enrolled into the second year after passing the exam.

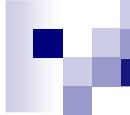


A student has an exam

- Our class has `Student` does not support the defined exams.
- We would like to have a function that would be called when the student will pass an exam. The grade will be passed a parameter.
- Now what?
- Define a new class, which will include both exams and use some internal logic to select the proper exam.

Student with S1 **or** DM

```
public class studentMAaliRIN {  
    /* -----[ data ]--- */  
  
    ...  
  
    private boolean mZS1;  
    /* -----[ create / destruct ]--- */  
    public student(String ime, String priimek, boolean zS1);  
    /* -----[ update ]--- */  
  
    ...  
    public void s1(int ocena);  
    public void dm(int ocena);  
    /* -----[ query ]--- */  
  
    ...  
    public int ocenaS1(void);  
    public int ocenaDM(void);  
} // studentMAaliRIN  
martin= new studentMAaliRIN("Martin", "Krpán", false);  
peter= new studentMAaliRIN("Peter", "Klepč", true);
```



Student has an exam

- We define two separate classes;
 - The first class (students of mathematics - MA) and the other (students of computer science - RIN)
 - Class is distinguished by the functions `d1` and `s1`

Class StudentMA

```
public class StudentMA {  
    /* -----[ data ]--- */  
    private String mIme;           private String mPriimek;  
    protected int mLetnik= 0;      private int mVpisnaStevilka= 0;  
    private int mOcenaDM= 0;  
    /* -----[ create / destruct ]--- */  
    public studentMA(String ime, String priimek);  
    /* -----[ update ]--- */  
    public void vpis(int vpisnaStevilka);  
    public void dm(int ocena);  
    /* -----[ query ]--- */  
    public String ime(void);  
    public String priimek(void);  
    public int letnik(void);  
    public int vpisnaStevilka(void);  
    public int ocenaDM(void);  
    public void izpisi(void);  
} // studentMA
```

Class StudentMA

```
/* -----[ create / destruct ]--- */
public studentMA(String ime, String priimek) {
    mIme= ime; mPriimek= priimek;
    mLetnik= 0; mVpisnaStevilka= 0;
    mOcenaDM= 0;
}; // studentMR
/* -----[ update ]--- */
public void vpis(int vpisnaStevilka) {
    mVpisnaStevilka= vpisnaStevilka;
    letnik= 1;
}; // vpis
public void dm(int ocena) {
    mOcenaDM= ocena;
    letnik= 2;
}; // dm
```

Class StudentMA

```
/* -----[ query ]--- */
public String ime(void) { return mIme; }
public String priimek(void) { return mPriimek; }
public int letnik(void) { return mLetnik; }
public int vpisnaStevilka(void) {
    return mVpisnaStevilka;
}; // vpisnaStevilka
public int ocenaDM(void) { return mOcenaDM; };
```

Class StudentMA

```
public void izpisi(void) {  
    System.out.print(mIme + " " + mPriimek);  
    System.out.print("(" + mVpisnaStevilka + ")");  
    System.out.print(" - " + letnik + ":");  
    System.out.print("dm: ");  
    if (mOcenaDM == 0) System.out.print("-");  
    else                  System.out.print(mOcenaDM);  
}; // izpis
```

Class StudentMA - usage

```
public class uporabaMA {  
    public static lepoIzpisi(student kdo) {  
        kdo.izpisi(); System.out.println();  
    } // lepoIzpisi  
  
    public static void main(String args[]) {  
        studentMA martin= new studentMA("Martin", "Krpan");  
        student peter= new student ("Peter", "Klepec");  
        student pehta= new student ("Botra", "Pehta");  
  
        lepoIzpisi(martin); lepoIzpisi(peter); lepoIzpisi(pehta);  
        martin.vpis(123); peter.vpis(124); pehta.vpis(125);  
        martin.dm(9);  
  
        lepoIzpisi(martin); lepoIzpisi(peter); lepoIzpisi(pehta);  
    }; // main  
} // uporabaMA
```

Class StudentRIN

```
public class studentRIN {  
    /* -----[ data ]--- */  
    private String mIme;           private String mPriimek;  
    protected int mLetnik= 0;      private int mVpisnaStevilka= 0;  
    private int mOcenaS1= 0;  
    /* -----[ create / destruct ]--- */  
    public studentRIN(String ime, String priimek);  
    /* -----[ update ]--- */  
    public void vpis(int vpisnaStevilka);  
    public void s1(int ocena);  
    /* -----[ query ]--- */  
    public String ime(void);  
    public String priimek(void);  
    public int letnik(void);  
    public int vpisnaStevilka(void);  
    public int ocenaS1(void);  
    public void izpisi(void);  
} // studentRIN
```

Class studentRIN

```
/* -----[ create / destruct ]--- */
public studentRIN(String ime, String priimek) {
    mIme= ime; mPriimek= priimek;
    mLetnik= 0; mVpisnaStevilka= 0;
    mOcenaS1= 0;
}; // studentRIN
/* -----[ update ]--- */
public void vpis(int vpisnaStevilka) {
    mVpisnaStevilka= vpisnaStevilka;
    letnik= 1;
}; // vpis
public void s1(int ocena) {
    mOcenaS1= ocena;
    letnik= 2;
}; // s1
```

Class studentRIN

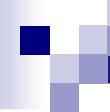
```
/* -----[ query ]--- */
public String ime(void) { return mIme; }
public String priimek(void) { return mPriimek; }
public int letnik(void) { return mLetnik; }
public int vpisnaStevilka(void) {
    return mVpisnaStevilka;
}; // vpisnaStevilka
public int ocenaS1(void) { return mOcenaS1; }
```

Class studentRIN

```
public void izpisi(void) {  
    System.out.print(mIme + " " + mPriimek);  
    System.out.print("(" + mVpisnaStevilka + ")");  
    System.out.print(" - " + letnik + ":");  
    System.out.print("s1: ");  
    if (mOcenaS1 == 0) System.out.print("-");  
    else                  System.out.print(mOcenaS1);  
}; // izpis
```

Student classes - usage

```
public class uporabaMAinRIN {  
    public static lepoIzpisi(student kdo) {  
        kdo.izpisi(); System.out.println();  
    }; // lepoIzpisi  
  
    public static void main(String args[]) {  
        studentMA martin= new studentMA("Martin", "Krpan");  
        studentRIN peter= new studentRIN("Peter", "Klepec");  
        student pehta= new student ("Botra", "Pehta");  
  
        lepoIzpisi(martin); lepoIzpisi(peter); lepoIzpisi(pehta);  
        martin.vpisi(123); peter.vpisi(124); pehta.vpisi(125);  
        martin.dm(9); peter.s1(8);  
  
        lepoIzpisi(martin); lepoIzpisi(peter); lepoIzpisi(pehta);  
    }; // main  
} // uporabaMAinRIN
```



What does an object contain

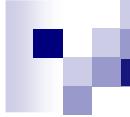
access type name(parameters);

- Object contains the data and methods for dealing with these data (data on the properties);
- We must indicate the accessibility of the data and methods;
- We can use:
 - All: **public**
 - Nobody: **private**
 - Friends (in module - package): **friend**
 - Friends and family: **protected**

Accessibility

■ *public and private:*

```
public class A {  
    public void javno() { zasebno(); };  
    private void zasebno() { ... };  
} // A  
public class B {  
    public void javno(A x) {  
        x.javno(); x.zasebno();  
    }  
} // B
```



What does an object contain

- Methods and data can be characterised with access and to the data and methods also characterized by the membership
- Common to all objects in the class or only in a particular object (`this`)
- If they are common to all living objects from the class, use the code: `static`;
- The value of a property can be changed (normal property), it is possible to define that a property cannot change its value (becomes a constant) : `final`.

static

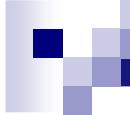
- Methods or data can be shared:

```
public class C {  
    public static int s= 0; // skupno  
    public         int p= 1; // vsak posebej  
} // C  
C c1= new C(); C c2= new C();  
                                (s, p): c1 (0, 1); c2 (0, 1);  
c1.s= 2;  
                                c1 (2, 1); c2 (2, 1);  
c2.p= 3;  
                                c1 (2, 1); c2 (2, 3);
```

final

- Methods or data can be constant (final) :

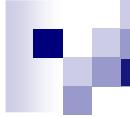
```
public class D {  
    public static int s= 0; // spremenljivo  
    public final int p= 1; // dokoncno  
} // D  
D c1= new D(); D c2= new D();  
  
          (s, p): c1 (0, 1); c2 (0, 1);  
c1.s= 2;  
  
          c1 (2, 1); c2 (2, 1);  
  
c2.p= 3; ni dovoljeno
```



Functions, interfaces and agreements...

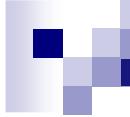
- A function can be seen as a black box that does something:
 - user view
 - user interface - API
 - Example: $x(u, v) \rightarrow z$

We do not care how something is done, but just what is done by the black box!



...interfaces and agreements...

- This view allows us to work independently in a group;
- A useful description of the function must include:
 - description of how the function is called (**signature**),
 - description of what the function actually does (**contract**).

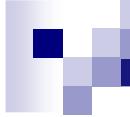


Signature

- A description how to call (invoke) a function:

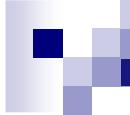
`int sestejDo0(int x)`

- type of the result,
- name of the function
- Parameters.



Contract

- The contract binds the user of the function and the writer (implement),
 - first knows what to expect and demand from a function,
 - second knows what the function does;
 - If the contract is very formal, it can be used to formally prove the behavior of the software.
 - We will use (semi-) formal variant.



Parts of a contract

■ Opis (*description*)

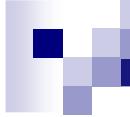
Desc:

compute the sum of integers from
0 to x.

■ Parametri (*parameters*)

Params:

x - operand [in]



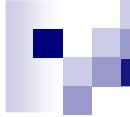
Parts of a contract

- Predpogoj (*precondition*)

Pre: $x > 0$

- Popogoj (*postcondition*)

Post: RESULT > 0



Parts of a contract

- Rezultat (*result*)

Result: $x + (x-1) + (x-2) + \dots + 1$

- Okolje (*environment*)

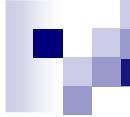
Env: -

Parts of a contract

```
/*
Desc:      compute the sum of integers
              from 0 to x.
Params:    x - operand [in]
Pre:       x > 0
Post:      RESULT > 0
Result:    x + (x-1) + (x-2) + ... + 1
Env:       -
*/

```

```
public int sestejDo0(x) { ... };
```



Argument passing

- How are the parameters passed to the function?
 - By value
 - By reference
- What is this?



Parameters

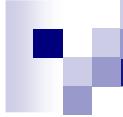
- The VALUE is passed
- Formal / dejanski parameter
- metodaA(12 + 5, a)
- Enako, kot če bi pisalo metodaA(12 + 5, 0 + a)
- Kar se dogaja s parametri v metodi, se na spremenljivkah, uporabljenih ob klicu, NE pozna!
- ```
public static int povecajZa2 (int x){
 return x + 2;
}
```

```
...
int x = 5;
y = povecajZa2(x);
povecajZa2(z);
povecajZa2(x + y);
```



# Arrays – as parameters

- ```
public static void sprTab( int[ ] x ) {  
    . . .  
}
```
- Call: `sprTab(nekaTabela);`
- The value of the parameter is passed
 - The address of the array
- The actual array is changed in the function!



Arrays – as parameters

- The method sprTab changes the values of array x and in the same time the values of the array nekaTabela
 - At the call of the function. The array x references (shows) to the same table as nekaTabela
- “normal“ (primitive typed) variables
 - ```
public static void sprSrem(int x) {
 ...
}
```
  - Call: sprSrem(y);
  - The value of y is not changed!

# Arrays

- ```
int tab1 = new int[100];
int tab2 = new int[20];
double tab3 = new double[17];
...
tab1 = tab2; // tab1 and tab2 are the same one array
            // the "old" tab1 CANNOT BE ACCESSED
```

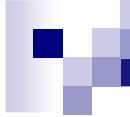

tab3 = tab1; //ERROR, the arrays are not the same type
- The array as a whole, in principle, is not used!
- A copy of the table
 - copy all elements
- Display arrays
 - display individual elements

Parameter passing – example nr. 1

```
public class Parametri1 {  
    public static void main(String[] args) {  
        int x = 10;  
        int y = 20;  
        metodaA(x, y + 2);  
    }  
  
    public static void metodaA(int a, int b) {  
        int c;  
        c = a;  
        a = a * b;           x <-- 10  
        b = c;              y <-- 20  
    }  
}
```

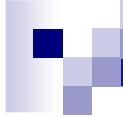
Parameter passing – example nr. 2

```
public class Parametri2 {  
    public static void main(String[] args) {  
        int x = 10;  
        int y = metodaB(20);  
        metodaA(x, y + 2);  
    }  
  
    public static void metodaA(int a, int b) {  
        int c;  
        c = a;  
        a = a * b;  
        b = metodaB(a);  
    }  
    public static int metodaB(int a) {  
        int c;  
        c = a;  
        a = a + c;  
        return a;  
    }  
}
```



Parameter passing – resume:

- All primitive types are passed by value:
 - Int, char, double, float
 - String!
- All compound types are passed by reference:
 - Arrays
 - Student, ...



References

- <http://java.sun.com/docs/books/tutorial/java/javaOO/>
- http://publib.boulder.ibm.com/infocenter/macxhelp/v6v81/index.jsp?topic=/com.ibm.vacpp6m.doc/language/ref/clrc07examples_calling_functions.htm
- <http://www.cs.toronto.edu/~diane/tutorials/params/>