Uvod in predstavitev programskega jezika Java

Splošnonamenski programski jezik **Java** je bil razvit kot **varen jezik za poljubno napravo** (npr. preverjanje tipov, vidljivost konstruktov, virtualni stroj). Sintaksa zahteva daljše programe kot v programskem jeziku Python, ki pa so lahko tudi desetkrat hitrejši. Jezik se prevaja, kar pomeni, da program `Demo.java` v ukazni vrstici najprej prevedemo kot `javac Demo.java`, kar ustvari vmesno datoteko `Demo.class`, katero nato izvedemo kot `java Demo`.

Programming is not science, it is a skill! If you want to run as fast as Usain Bolt, you have to do a lot of running. There is no other way! And it is the same with programming. Just try to run a lot =)

Najkrajši program in izpis na zaslon

V programskem jeziku **Java** mora vsaka **izvorna datoteka** s programsko kodo vsebovati vsaj javni razred katerega ime je enako imenu datoteke (npr. `Demo`). V kolikor želimo program tudi izvajati, mora omenjen razred vsebovati javno statično metodo `main(String[] args)`, kjer se začne izvajanje programa. Pri tem je parameter `args` tabela nizov znakov, ki jih **uporabnik doda klicu programa**. Na primer, če program v ukazni vrstici izvedemo kot

```bash
java Demo 1 -x fast
```

bo tabela `args` vsebovala nize znakov `"1"`, `"-x"` in `"fast"`.

V programskem jeziku **Java bloke kode**, ki naj se izvedejo skupaj oziroma zaporedoma, določimo z zavitimi oklepaji `{...}`. Vsak programski stavek zaključimo s podpičjem `;`, dočim pa je lahko celoten program v eni vrstici. V ukazni vrstici lahko **izpišemo niz znakov** `str` z uporabo metode `println(String str)` objekta `out` v razredu `System`.

```java
public class Demo {

    /** Javadoc komentar */
    public static void main(String[] args) {
        System.out.println("Pozdravljeni pri predmetu PRO2!"); // vrstični komentar
        /* večvrstični ali bločni komentar */
    }

}
```

Programske knjižnice in paketi

V programskem jeziku **Java razrede drugih paketov** ali knjižnic **uvozimo** s stavkom `import`, kot je prikazano spodaj. Uporabljene razrede je potrebno uvoziti izven definicije razreda `Demo` (na samem začetku izvorne datoteke).

```java
import java.util.ArrayList;
import java.util.List;
import java.io.*;
```

Vsak Java razred se nahaja **v nekem paketu** `package` (npr. `java.util`), pri čimer je hierarhija paketov dejansko predstavljena z mapami datotečnega sistema. Na primer, če bi se izvorna datoteka razreda `Demo` nahajala v mapi `./pro2/demo`, bi morali na samem začetku dodati še spodnji stavek. Le-tega lahko izpustimo, če se izvorna datoteka nahaja v korenski mapi `.`

```java
package pro2.demo;
```

Programske spremenljivke in konstante

V programskem jeziku **Java spremenljivke** definiramo in jim določimo začetno vrednost *preden* so prvič uporabljene. Pri definiciji **moramo določiti tip** spremenljivke, ki ga v nadaljevanju ni moč spremeniti! **Primitivni tipi spremenljivk** so logične vrednosti enake `true` ali `false` (tj. `boolean`), cela števila (npr. `int`, `long`), realna števila (tj. `float`, `double`) in posamezni znaki `'.'` (tj. `char`). Med **osnovne tipe spremenljivk** navadno štejemo tudi nize znakov `"..."`, ki so objekti razreda `String`.

```java
int x = 1;
double y;
y = 1.23 * 9;
char ch = 'a'; // enojni narekovaji '.'
String str = "niz znakov"; // dvojni narekovaji "..."
int len = str.length();
boolean b = x > 1;
```

V programskem jeziku **Java konstante** označimo z določilom `final`, ki jih po določitvi začetne vrednosti ni več moč spremeniti. Navadno jih označimo z velikimi črkami.

```java
final float G = 9.81f;
```

Spremenljivke in konstante so **veljavne le znotraj bloka kode**, v katerem so definirane. **Med osnovnimi tipi** spremenljivk in konstant lahko **pretvarjamo** z uporabo javnih statičnih funkcij razredov `Integer`, `Double`, `String` itd., kot je prikazano spodaj. Pri tem operator `+` predstavlja konkatenacijo nizov

znakov, ker je v vseh primerih vsaj en od argumentov niz znakov.

```java
int z = (int)y; // celi del števila
double w = (double)x; // 1.0 * x
z = Integer.parseInt("7");
w = Double.parseDouble("1.23");
str = "Vrednost spremenljivke w je enaka " + w; // konkatenacija nizov
str = String.format("Vrednost spremenljivke w je enaka %.3f", w); // formatiranje nizov
System.out.println(x + " " + y + " " + z + " " + w + " " + str);
```

S spremenljivkami **primitivnih številskih tipov** lahko **računamo** z uporabo standardnih operatorjev in javnih statičnih funkcij razreda `Math`, kot je prikazano spodaj. Pri tem operator `+` predstavlja seštevanje, ker sta oba argumenta števili.

```java
System.out.println(x + y * z / w);
System.out.println(x % z); // ostanek pri deljenju
System.out.println(Math.pow(y, 2.0)); // potenciranje števil
System.out.println(42.0 * Math.random()); // naključno število iz [0, 42)
System.out.println((int)(3.0 * Math.random())); // naključno število iz {0, 1, 2}
```

Pogojni stavki in programske vejitve

Programske vejitve omogočajo **selektivno izvajanje** programske kode glede na določen logičen pogoj. Najpogosteje se uporabljajo pogojni stavki (tj. `if else` stavki), dočim v večini programskih jezikih obstajajo tudi izbirni stavki (tj. `switch` stavki) in drugi. Vse pogojne stavke je moč gnezditi ipd.

V programskem jeziku **Java pogojne stavke** zapišemo kot je prikazano spodaj. Pri tem bloke kode, ki vsebujejo le en programski stavek, ni potrebno posebej označiti z zavitimi oklepaji `{...}`.

```java
if (x < 1) {
    System.out.println("Vrednost spremenljivke x je manjša od 1");
}
else if (x < 2)
    System.out.println("Vrednost spremenljivke x je med 1 in 2");
else
    System.out.println("Vrednost spremenljivke x je večja ali enaka 2");
```

Vgnezdene pogojne stavke `? :` zapišemo kot je prikazano spodaj. Pri tem operator `+` predstavlja konkatenacijo nizov znakov, ker je v obeh primerih vsaj en od argumentov niz znakov.

```java
System.out.println("Vrednost spremenljivke x je " + (x < 1? "manjša od ": "večja ali enaka
```

Logične vrednosti v pogoju lahko združujemo z uporabo negacije `!`, konjunkcije `&&` in disjunkcije `||`, kot je prikazano spodaj.

Java
```
if (x == 1 || x == 2)
    System.out.println("Vrednost spremenljivke x je enaka 1 ali 2");
if (x == 1 && x == 2)
    System.out.println("To ni mogoče!");
if (x != 1 && x != 2) // if (!(x == 1 || x == 2))
    System.out.println("Vrednost spremenljivke x ni enaka 1 ali 2");
```

V programskem jeziku **Java izbirne stavke** zapišemo kot je prikazano spodaj. Pri tem lahko izbiramo le preko vrednosti primitivnih spremenljivk, dočim posamezne vrednosti določimo z uporabo ukaza `case` in privzeto vrednost z uporabo ukaza `default`. Pomembno je, da vsako izbiro zaključimo z ukazom `break`.

Java
```
switch (x) {
    case 1:
        System.out.println("Vrednost spremenljivke x je enaka 1");
        break;
    case 2:
        System.out.println("Vrednost spremenljivke x je enaka 2");
        break;
    default:
        System.out.println("Vrednost spremenljivke x ni enaka 1 ali 2");
        break;
}

switch (ch) {
    case 'a':
        System.out.println("Vrednost spremenljivke ch je enaka 'a'");
        break;
    default:
        System.out.println("Vrednost spremenljivke ch ni enaka 'a'");
        break;
}
```

Iterativno izvajanje in programske zanke

Programske zanke omogočajo **iterativno izvajanje** programske kode dokler velja določen logičen pogoj. Najpogosteje se uporabljajo standardne zanke (tj. `for` in `while` zanke), dočim v večini programskih jezikih obstajajo tudi npr. `do while` zanke in druge. Vse zanke je moč gnezditi ipd.

V programskem jeziku **Java `for` zanko** zapišemo kot je prikazano spodaj. Pri tem se najprej izvede prvi parameter zanke (ločen s podpičjem `;`), ki v spodnjem primeru definira in nastavi začetno vrednost

števca `i`. Pred vsako iteracijo zanke se izvede drugi parameter (ločen s podpičjem `;`), ki določi pogoj dokler se zanka še izvaja. Po vsaki iteraciji zanke se izvede tretji parameter, ki v spodnjem primeru poveča vrednost števca za ena.

```java
for (int i = 0; i < 3; i += 1) {
    System.out.println("Vrednost spremenljivke i je enaka " + i);
}
```

Ekvivalentno lahko v programskem jeziku **Java** **`while`** **zanko** zapišemo kot je prikazano spodaj.

```java
int ind = 0;
while (ind < 3) {
    System.out.println("Vrednost spremenljivke ind je enaka " + ind);
    ind++; // ind += 1;
}
```

Z **Java** **`for`** **zanko** lahko iteriramo tudi **preko podatkovnih zbirk**, kot je prikazano spodaj. Pri tem je podatkovna zbirka lahko tabela (npr. `args` tipa `String[]`), seznam (tj. objekt razreda `List`) ali množica (tj. objekt razreda `Set`), dočim moramo pri uporabi navesti tip elementov zbirke po kateri iteriramo (npr. `arg` tipa `String`).

```java
for (String arg: args)
    System.out.println(arg);
```

Programske **zanke predčasno zaključimo** z uporabo ukaza `break`, dočim naslednjo iteracijo zanke **predčasno pričnemo** z uporabo ukaza `continue`.

Programske metode in funkcije

Programske metode in funkcije omogočajo **ponovljeno izvajanje** enake programske kode upoštevajoč podane argumente. Pri tem metode zgolj izvedejo določeno programsko kodo, funkcije pa vrnejo tudi rezultat z uporabo stavka `return`.

V programskem jeziku **Java metodo** zapišemo kot je prikazano spodaj. Pri tem zaporedoma določimo vidljivost metode (npr. `public`), ali gre za statično metodo razreda (tj. `static`) ali metodo objekta, tip rezultata (tj. `void` v primeru metode) ter na koncu samo ime metode (npr. `method` v spodnjem primeru).

```java
public static void method(int x, double y) {
    System.out.println("Vrednost produkta x*y je enaka " + x * y);
}
```

Argumentom metode moramo obvezno **določiti tip** (npr. `double y`), dočim lahko **privzete vrednosti argumentov** določimo preko metod z enakim imenom in različnim seznamom parametrov (tj. preobteževanje metod).

```java
static void method(int x) {
    method(x, 1.0);
}

static void method(double y) {
    method(42, y);
}

static void method() {
    method(42);
}
```

V programskem jeziku **Java funkcijo** zapišemo kot je prikazano spodaj. Za razliko od metode moramo **določiti tip rezultata** (npr. `int` v spodnjem primeru) in le-tega na koncu funkcije vrniti z uporabo stavka `return`.

```java
public static int function(int i) {
    System.out.println("Vrednost vhodnega argumenta funkcije je enaka " + i);
    i += 13;
    System.out.println("Vrednost rezultata funkcije je enaka " + i);
    return i;
}
```

Vidljivost programskih konstruktov

V programskem jeziku **Java** lahko vsakemu **programskemu konstruktu** (npr. spremenljivki, metodi, funkciji, razredu) pri definiciji **določimo vidljivost** in s tem omejimo dostop. Slednje seveda ne velja za lokalne spremenljivke ter argumente metod in funkcij (npr. vse spremenljivke zgoraj), ki so vedno vidne le lokalno. Določila za omejitev dostopa programskih konstruktov so našteta spodaj.

- `private` — dostopno znotraj razreda
- ` ` — dostopno znotraj razreda ali paketa
- `protected` — … razreda, podrazreda ali paketa
- `public` — dostopno iz vseh razredov in paketov

Poimenovanje programskih konstruktov

V programskem jeziku **Java programske konstrukte** (npr. spremenljivke, metode, funkcije, razrede) **poimenujemo** v skladu s splošno sprejetim stilom, kot je opisano spodaj.

- *spremenljivke, funkcije ipd.* — z malo začetnico kot npr. `myDemoFunction`
- *razredi, vmesniki ipd.* — z veliko začetnico kot npr. `MyDemoClass`
- *konstante* — z velikimi črkami kot npr. `MY_DEMO_CONSTANT`

Programski razredi, objekti in dedovanje

Pri **objektno orientiranem programiranju** skupke programskih konstruktov, s katerimi želimo upravljati kot s celoto, združujemo v objekte, ki so določeni z razredi in vmesniki. Pri tem razredi predstavljajo tip objekta, ki združuje atribute, metode in funkcije objekta.

V programskem jeziku **Java razrede definiramo** z ukazom `class` (npr. `class Demo`), dočim **vmesnike definiramo** z ukazom `interface`. S preobteževanjem metod lahko razredu definiramo več **konstruktorjev**, ki so javne metode z enakim imenom kot je ime razreda (npr. `public Demo()`) in poskrbijo za začetno stanje objekta. Pri tem rezervirana beseda `this` predstavlja sam objekt razreda, rezervirana beseda `super` pa objekt nadrazreda (tj. očeta). Navadno redefiniramo tudi funkcijo `toString()`, ki vrne niz znakov z berljivim opisom objekta razreda, in funkcijo `equals(Object object)`, ki preveri ali je objekt razreda enak podanemu objektu `object`. Pazite, da pri definiciji atributov, metod in funkcij ne uporabite določila `static`!

Definicija razreda `XY`, ki naj predstavlja točko v ravnini, je prikazana spodaj. Zaradi enostavnosti je razred `XY` vključen kar v izvorno datoteko razreda `Demo`, dočim je navadno vsak razred v svoji izvorni datoteki. To hkrati pomeni, da pri definiciji razreda ne smemo uporabiti določila `public`!

Java

```java
class XY {

  private int x;

  private int y;

  public XY() {
    this(0);
  }

  public XY(int x) {
    this(x, 1);
  }

  public XY(int x, int y) {
    super();

    this.x = x;
    this.y = y;
  }

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }

  @Override
  public String toString() {
    return "x = " + getX() + ", y = " + getY();
  }

  @Override
  public boolean equals(Object object) {
    if (!(object instanceof XY)) // preverjanje tipov
      return false;

    XY xy = (XY)object; // pretvarjanje tipov
    return getX() == xy.getX() && getY() == xy.getY();
  }

}
```

Ključen koncept pri objektno orientiranem programiranju je **dedovanje razredov**, pri čimer (pod)razred (tj. sin) prevzame vse atribute, metode in funkcije nadrazreda (tj. očeta). (Pod)razredu lahko definiramo tudi

poljubne druge atribute, metode in funkcije, poleg tega pa lahko redefiniramo (tj. prepišemo) funkcionalnosti nadrazreda. V programskem jeziku **Java nadrazred dedujemo** z uporabo rezervirane besede `extends`, pri čimer ima vsak (pod)razred lahko le en nadrazred!

Definicija (pod)razreda `XYZ` nadrazreda `XY`, ki naj predstavlja točko v prostoru, je prikazana spodaj.

Java

```
class XYZ extends XY {

  private int z;

  public XYZ(int x, int y, int z) {
    super(x, y);

    this.z = z;
  }

  public int getZ() {
    return z;
  }

  @Override
  public String toString() {
    return super.toString() + ", z = " + getZ();
  }

  @Override
  public boolean equals(Object object) {
    if (!super.equals(object) || !(object instanceof XYZ))
      return false;

    return getZ() == ((XYZ)object).getZ();
  }

}
```

Delovanje razredov `XY` in `XYZ` lahko preizkusimo s pomočjo spodnjega programa. Pri tem objekte razredov ustvarimo tako, da pred imenom konstruktorja uporabimo rezervirano besedo `new`.

Java

```
XY xy = new XY(1, 2);
System.out.println(xy);
System.out.println(new XY());
System.out.println(new XY(1).equals(xy));
System.out.println(new XYZ(1, 2, 3));
```

Dočim ima vsak (pod)razred definiran z ukazom `class` lahko le en nadrazred, pa lahko le-ta

implementira več vmesnikov definiranih z ukazom `interface`, ki vsebujejo zgolj definicije metod in funkcij brez implementacije. V programskem jeziku **Java vmesnik implementiramo** z uporabo rezervirane besede `implements`.

Definicija (pod)razreda `Point` nadrazreda `XYZ`, ki implementira vmesnik `Printable`, je prikazana spodaj.

Java
```java
interface Printable {

  public void print();

}

class Point extends XYZ implements Printable {

  public Point(int x, int y, int z) {
    super(x, y, z);
  }

  @Override
  public void print() {
    System.out.println(toString());
  }

}
```

Delovanje razreda `Point` lahko preizkusimo s pomočjo spodnjega programa.

Java
```java
Point point = null; // prazna vrednost
point = new Point(1, 2, 3);
point.print();
XYZ xyz = point;
System.out.println(xyz);
```

V programskem jeziku **Java abstraktni razred** definiramo z ukazom `abstract class` (npr. `abstract class Demo`), ki predstavlja vmesno možnost med razredi in vmesniki.

Programske zbirke podatkov in tabele

Seznami podatkov

V programskem jeziku **Java seznam** predstavimo kot objekt razreda, ki implementira vmesnik `List` v paketu `java.util`. Najpogosteje uporabimo **sezname podprte s tabelo** definirane v razredu `ArrayList`.

```java
import java.util.Collections;
import java.util.ArrayList;
import java.util.List;
```

Seznam je **urejena zbirka podatkov spremenljive velikosti** kar pomeni, da lahko dodajamo, spreminjamo in brišemo elemente ter dostopamo do elementov po indeksu. Pri tem morajo biti elementi seznama objekti istega razreda ali njegovih podrazredov, dočim **tip elementov določimo** s trikotnimi oklepaji `<...>` (npr. `ArrayList<Double>`). Tako ni moč ustvariti seznama elementov primitivnega tipa (npr. `double`)!

```java
List<Double> list = new ArrayList<Double>();
list.add(1.0);
list.add(0, 1.1);
list.set(0, 0.9);
System.out.println(list.size());
System.out.println(list.get(1));
for (double value: list)
    System.out.println(value);
list.remove(0);
```

Seznam lahko **uredimo na mestu** z uporabo javne statične metode `sort(List<?> list)` razreda `Collections` v paketu `java.util`. Podobno lahko seznam **naključno premešamo** z uporabo javne statične metode `shuffle(List<?> list)`.

```java
for (int i = 0; i < 3; i++)
    list.add(Math.random());
Collections.sort(list);
for (double value: list)
    System.out.println(value);
```

Množice podatkov

V programskem jeziku **Java množico** predstavimo kot objekt razreda, ki implementira vmesnik `Set` v paketu `java.util`. Najpogosteje uporabimo **množice implementirane z zgoščevalnimi funkcijami** definirane v razredu `HashSet`.

```java
import java.util.HashSet;
import java.util.Set;
```

Množica je **neurejena zbirka enoličnih podatkov spremenljive velikosti** kar pomeni, da lahko dodajamo in brišemo elemente ter dostopamo do elementov po vrednosti. Pri tem morajo biti elementi množice objekti istega razreda ali njegovih podrazredov, dočim **tip vrednosti določimo** s trikotnimi oklepaji `<...>` (npr.

`HashSet<Double>`). Tako ni moč ustvariti množice vrednosti primitivnega tipa (npr. `double`)!

```java
Set<Double> set = new HashSet<Double>();
set.add(1.0);
set.addAll(list);
System.out.println(set.size());
System.out.println(set.contains(1.0));
for (double value: set)
    System.out.println(value);
set.remove(1.0);
```

Slovarji podatkov

V programskem jeziku **Java slovar** predstavimo kot objekt razreda, ki implementira vmesnik `Map` v paketu `java.util`. Najpogosteje uporabimo **slovarje implementirane z zgoščevalnimi funkcijami** definirane v razredu `HashMap`.

```java
import java.util.HashMap;
import java.util.Map;
```

Slovar je **neurejena zbirka enoličnih preslikav spremenljive velikosti** kar pomeni, da lahko dodajamo, brišemo in dostopamo do vrednosti po ključu. Pri tem morajo biti ključi in vrednosti slovarja objekti istega razreda ali njegovih podrazredov, dočim **tip ključev in vrednosti določimo** s trikotnimi oklepaji `<...>` (npr. `HashMap<String, Integer>`). Tako ni moč ustvariti slovarja vrednosti primitivnega tipa (npr. `int`)!

```java
Map<String, Integer> map = new HashMap<String, Integer>();
map.put("foo", 0);
map.put("bar", 1);
map.put("baz", 1);
System.out.println(map.size());
System.out.println(map.get("foo"));
System.out.println(map.containsKey("baz"));
for (String key: map.keySet())
    System.out.println(key + " " + map.get(key));
map.remove("baz");
```

Tabele podatkov

V programskem jeziku **Java tabelo** določimo z oglatimi oklepaji `[]` za imenom tipa elementov (npr. `int[]`, `Integer[]`). Tabela je **urejena zbirka podatkov nespremenljive velikosti** kar pomeni, da moramo velikost tabele podati ob ustvarjanju tabele (npr. `new int[3]`), dočim lahko spreminjamo in dostopamo do elementov le po indeksu. Pri tem so lahko **elementi tabele primitivnega tipa** ali pa objekti

istega razreda, dočim tip elementov določimo ob definiciji (npr. `int[] array`).

Java

```java
double[] array = new double[3]; // privzete vrednosti
for (int i = 0; i < array.length; i++) {
    System.out.println(array[i]);
    array[i] = Math.random(); // naključna vrednost
    System.out.println(array[i]);
}

array = new double[] {0.0, 1.0, 2.0};
for (double value: array)
    System.out.println(value);

int[][] array2 = new int[4][7];
for (int i = 0; i < array2.length; i++) {
    for (int j = 0; j < array2[i].length; j++) {
        array2[i][j] = i * array2[i].length + j + 1;
        System.out.format("%3d", array2[i][j]);
    }
    System.out.println();
}
```

Branje in pisanje tekstovnih datotek

V programskem jeziku Java lahko **tekstovno datoteko preberete** s pomočjo spodnjega programa…

Java

```java
try {
    BufferedReader reader = new BufferedReader(new FileReader("lorem.txt"));
    int i = 1; String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(i + ". " + line);
        i++;
    }
    reader.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

…dočim lahko **v datoteko zapišete nize znakov** s pomočjo spodnjega programa.

```java
try {
    BufferedWriter writer = new BufferedWriter(new FileWriter("array.txt"));
    for (int i = 0; i < array2.length; i++) {
        for (int j = 0; j < array2[i].length; j++)
            writer.write(String.format("%3d", array2[i][j]));
        writer.write("\n");
    }
    writer.flush();
    writer.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

Nizi znakov in regularni izrazi

Za **delo z nizi znakov** v programskem jeziku Java si lahko ogledate javne metode in funkcije v razredu `String` (npr. `length()`, `charAt(int index)`, `indexOf(String str)`, `substring(int index)`), za **delo z regularnimi izrazi** pa dokumentacijo razreda `Pattern`.