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 `Demo2.java` v ukazni vrstici najprej prevedemo z `javac Demo2.java`, kar ustvari vmesno datoteko `Demo2.class`, katero nato izvedemo z `java Demo2`.

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 `Demo2` (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 `Demo2` 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;
```

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 (t.j. vse spremenljivke doslej), 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 Demo2`), 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 Demo2()`) 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 `Demo2`, 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
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
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
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}; // začetne vrednosti
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();
}
```

Časovne zahtevnosti

V spodnji tabeli so prikazane **časovne zahtevnosti osnovnih operacij** nad standardnimi Java podatkovnimi zbirkami, kjer je n število elementov zbirke in i indeks (tj. zaporedna številka) elementa.

Zbirka	Vmesnik	Razred	Iskanje	Dodajanje	Brisanje	Urejanje	Duplikati?
tabela	/	`[]`	$\mathcal{O}(n)$	/	/	$\mathcal{O}(n \log n)$	ja
seznam	`List`	`ArrayList`	$\mathcal{O}(n)$	$\mathcal{O}(n-i)$	$\mathcal{O}(n-i)$	$\mathcal{O}(n \log n)$	ja
množica	`Set`	`HashSet`	$\approx \mathcal{O}(1)$	$\approx \mathcal{O}(1)$	$\approx \mathcal{O}(1)$	/	ne
slovar	`Map`	`HashMap`	$\approx \mathcal{O}(1)$	$\approx \mathcal{O}(1)$	$\approx \mathcal{O}(1)$	/	ne

Branje in pisanje tekstovnih datotek

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

```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`.