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

```java
static void method(int x) {
    method(x, 2.0);
}

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

static void method() {
    method(1);
}
```

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

```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;
}
```