Vežbe 1

Java programski jezik je jezik visokog nivoa koji se može opisati sledecim epitetma:

- Jednostavan
- Objektno-orijentisan
- Distribuiran
- Multithread - Nit (eng. thread) je osnovna izvršna jedinica u procesu koji omogućava programima izvršavanje niza instrukcija na procesoru. Svaka nit ima svoj stek i registre, što joj omogućava izvršavanje nezavisnih sekvenci instrukcija. Niti u procesu dele isti adresni prostor i resurse, što omogućava jednostavnu razmenu podataka i komunikaciju između njih. Niti omogućavaju paralelno ili istovremeno izvršavanje različitih delova programa, što povećava efikasnost programa, posebno na višejezgarnim računarima. Korišćenje niti može poboljšati odziv aplikacija i iskorišćenje resursa.
- Dinamičan
- Nezavisan od arhitekture računara
- Programski jezik visokih performani
- Siguran

Kompajliranje i pokretanje Java programa

Primer Java programa - HelloWorld.java

```
public class HelloWorldApp
{
  public static void main(String[] args)
  {
    System.out.println("Hello World!"); // Prikazi string.
  }
}
```

1. Svaka Java aplikacija mora sadržati barem jednu klasu s metodom main(String[] args). main metoda je prva metoda koja će se izvršiti nakon što se pokrene Java program (slično kao i u C).
2. Ovako napisan program se prevodi izvrsavajući komandu `javac HelloWorld.java` .
3. Ako nema grešaka prevodilac javac kreira datoteku HelloWorld.class koja sadrži bytecode instrukcije za JVM (Java virtual machine). Java virtuelna mašina je program koji je dostupan na mnogim operativnim sistemima i služi za izvršavanje bytecode-a (mašinski jezik za JVM). Isti .class fajl moguće je pokrenuti na Windows-u, Linux-u, Solaris OS-u ili Mac OS-u. Zbog toga je Java programski jezik koji je nezavisan od platforme. Potrebno ga je kompajlirati samo jednom i moguće da ga je nakon toga izvršiti svuda.
4. JVM se pokreće sa `javac HelloWorld` .

Razlika između proceduralnog i objektno-orijentisanog programiranja

Standardno, **proceduralno programiranje**(npr. C): Program započinje izvršavanjem funkcije main koja izvršava postavljeni zadatak pozivanjem drugih funkcija. Program završava kad se izvrše sve instrukcije funkcije

main. Osnovni građevni blok programa je, dakle, funkcija. Postavljeni zadatak se rešava tako da što se razbije na niz manjih zadataka od kojih se svaka može implementirati u jednoj funkciji, tako da je program niz funkcijskih poziva.

U **objektno-orijentisanom programiranju** osnovnu ulogu imaju objekti koji sadrže i podatke i funkcije (metode). Program se konstruiše kao skup objekata koji međusobno komuniciraju. Podaci koje objekat sadrži predstavljaju njegovo stanje, dok pomoću metoda on to stanje može da menja i komunicira sa drugim objektima.

Objektno orijentisani koncept programiranja

Objekat

Objekati su ključna stvar za razumevanje objektno-orijentisane tehnologije. U prirodi oko nas možemo uočiti mnoge primere objekata: Pas, računar, televizor, automobil...

Objekti u prirodi dele dve karakteristike:

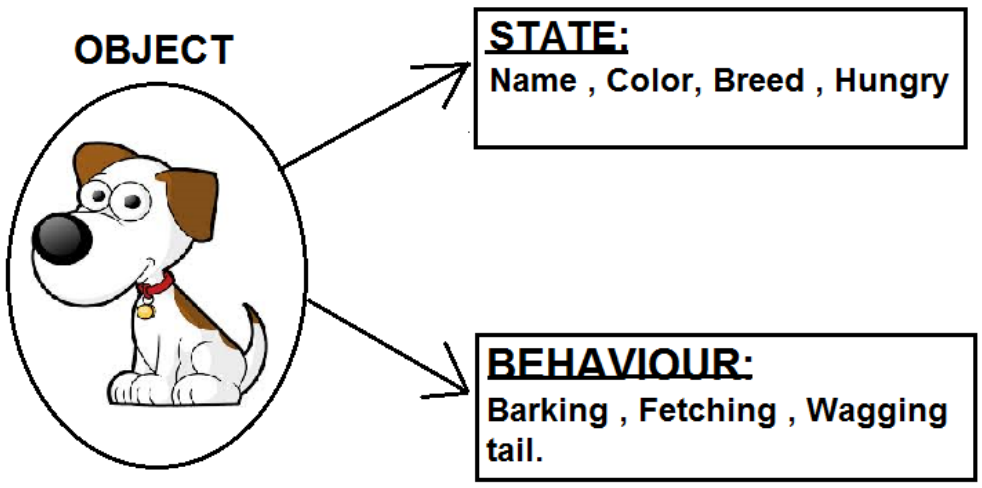
- **Stanje**. Automobil ima nekoliko stanja: marka, klasa, boja, maksimalna brzina, tip menjača...
- **Ponašanje**. Automobil može da se upali, ugasi, može da mu se promeni brzina...

Programski objekti su konceptualno isti kao objekti u prirodi. Oni takođe imaju stanje i ponašanje. Objekti svoja stanja čuvaju u poljima (promenljivama), dok svoje ponašanje ispoljavaju uz pomoć metoda (funkcija).

Prilikom uočavanja objekata oko nas, potrebno je za početak odgovoriti na dva pitanja:

- U kojim stanjima može biti objekat?
- Koja su moguća ponašanja objekta?

Primer 1.1



Pisanje koda u individualne programske objekte nosi sa sobom nekoliko pogodnosti:

1. Modularnost,
2. Skrivanje informacija - promena stanja objekta spolja može se vršiti samo preko metoda,
3. Ponovna upotreba koda,
4. Jednostavnije otkrivanje i otklanjanje grešaka.

Klasa

U prirodi se mogu pronaći mnogi objekti koji su istog tipa. Postoje hiljade automobila koji su od istog proizvođača i istog tipa. Svaki od tih automobila izrađen je prema istom spisku nacrta i ima iste komponente. U objektno-orijentisanoj terminologiji kažemo da je automobil instanca klase objekata automobila. Tačnije klasa je šematski plan za svaki objekat koji se kreira. To ćemo najbolje ilustrovati na primeru 1.2 u kome su kreirane dve klase Automobil i TestKlasa koja u sebi sadrži metodu main():

Primer 1.2

```
class Automobil {
  int stepenPrenosa = 1;
  int brzina = 0;

  void promeniBrzinu(int novaStepenPrenosa) {
    stepenPrenos = noviStepenPrenosa;
  }

  void ubrzaj(int povecanje) {
    brzina += povecanje;
  }

  void uspori(int smanjenje) {
    brzina -= smanjenje;
  }

  void stampajStanje() {
    System.out.println("stepen prenosa:" + stepenPrenosa + " brzina:" + brzina);
  }
}


public class TestnaKlasa {
  public static void main(String[] args) {
    Automobil a1 = new Automobil();
    Automobil a2 = new Automobil();

    a1.promeniBrzinu(2);
    a1.ubrzaj(30);
    a1.stampajStanje();
```

```
    a2.promeniBrzinu(2);
    a2.ubrzaj(30);
    a2.promeniBrzinu(3);
    a2.ubrzaj(20);
    a2.uspori(35);
    a2.promeniBrzinu(2);
    a2.stampajStanje();
  }
}
```

U jednom .java fajlu može postojati beskonačno mnogo klasa. Jedini uslov je da ako postoji public klasa mora se zvati isto kao i fajl i može postojati samo jedna takva klasa.

Overloading

Overloading je koncept u mnogim programskim jezicima koji omogućava definisanje više metoda ili funkcija istog imena, ali s različitim potpisima (parametrima) unutar iste klase ili opsega. Parametri se moraju razlikovati po broju ili tipu.

Glavne karakteristike overloadinga su:

1. Isto ime, različiti potpisi: Overloading dozvoljava isto ime za više metoda ili funkcija, ali uz različite parametre, kao što su različit broj parametara, tipovi parametara ili oba.
2. Lakša upotreba i razumljivost: Omogućava programerima da koriste intuitivna imena za funkcije/metode, čineći kod čitljivijim i olakšavajući upotrebu.

Primer 1.3

```
class OverloadDemo {

  void test() {
    System.out.println("No parameters");
  }

  void test(int a) {
    System.out.println("a: " + a);
  }

  void test(int a, int b) {
    System.out.println("a and b: " + a + " " + b);
  }

  double test(double a) {
    System.out.println("double a: " + a);
    return a*a;
  }
}
```

```
class Overload {
  public static void main(String args[]) {
    OverloadDemo ob = new OverloadDemo();
    double result;

    ob.test();
    ob.test(10);
    ob.test(10, 20);
    result = ob.test(123.2);
    System.out.println("Result of ob.test(123.2): " + result);
  }
}
```

Konstruktor

Konstruktor je metod ima isto ime kao i klasa, poziva se isključivo pri instanciranju objekata (dakle, operatorom **new**) i nema povratne vrednosti. Klasa može imati više konstruktora (overloading na delu). Konstruktor bez parametara je **default-ni konstruktor** i on postoji kada klasa nema posebno implementiran (naveden) ni jedan konstuktor. Ukoliko je implementiran makar jedan konstruktor, onda default-ni konstruktor ne postoji, osim ukoliko ga ne implementiramo.

Primer 1.4

```
public class Cube
{
  int length;
  int breadth;
  int height;

  public int getVolume() {
    return (length * breadth * height);
  }

  Cube() {
    length = 10;
    breadth = 15;
    height = 5;
    System.out.println("Constructor with no parameters");
  }


  Cube(int l, int b) {
    length = l;
    breadth = b;
    height = 10;
    System.out.println("Constructor with two parameters");
```

```
  }

  Cube(int l, int b, int h) {
    length = l;
    breadth = b;
    height = h;
    System.out.println("Constructor with three parameters");
  }

  public static void main(String[] args) {
    Cube cubeObj1, cubeObj2;
    cubeObj1 = new Cube();
    cubeObj2 = new Cube(10, 20, 30);
    System.out.println("Volume of Cube1 is : " + cubeObj1.getVolume());
    System.out.println("Volume of Cube2 is : " + cubeObj2.getVolume());
  }
}
```
