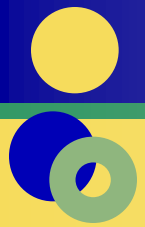
Klase



Objektno-orjentisano programiranje 1

Klasa

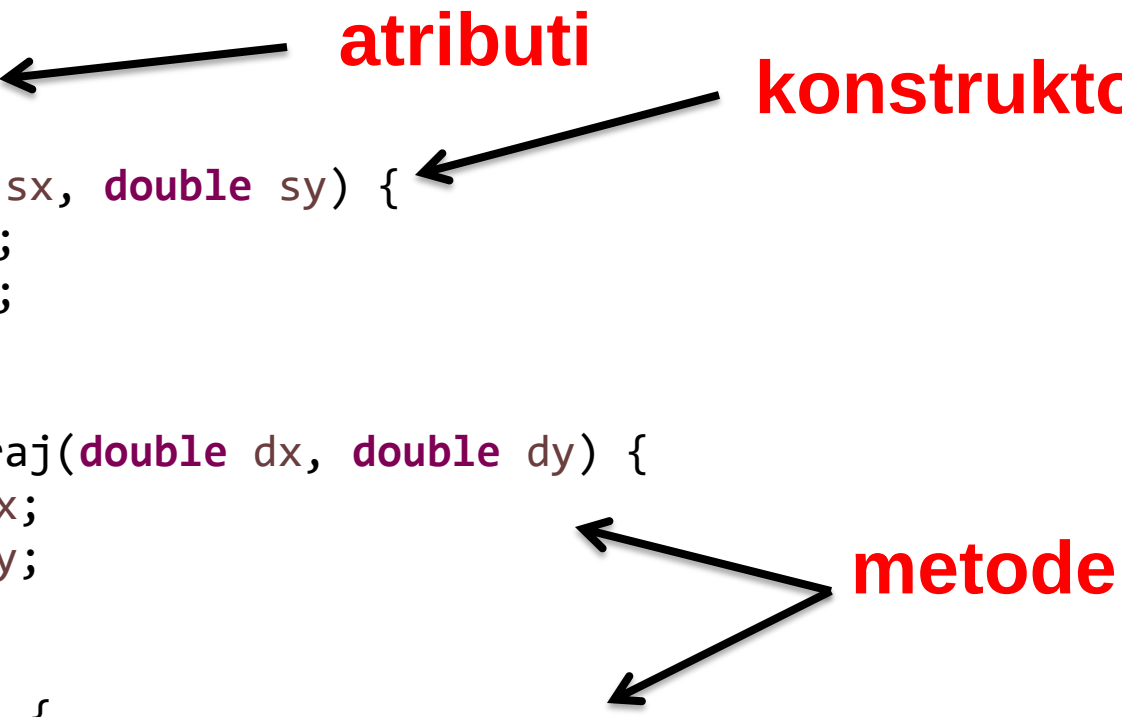
- Klasa – definicija skupa istorodnih objekata
- Klasom definišemo atribute (polja) i metode klase
- Konstruktor – specijalni metod za inicijalizaciju objekta

```
class Tacka {
	double x, y;

	Tacka(double sx, double sy) {
		x = sx;
		y = sy;
	}

	void transliraj(double dx, double dy) {
		x += dx;
		y += dy;
	}

	String opis() {
		return "Tacka (" + x + ", " + y + ")";
	}
}
```



Klase instanciramo operatorom **new**

- **Operator new dinamički alocira memoriju za objekat i poziva konstruktor klase**

```
public class TackeDemo {
	public static void main(String[] args) {
		Tacka t1 = new Tacka(12, 10);
		t1.transliraj(1, 4);
		System.out.println(t1.opis());
		// Tacka (13.0, 14.0)

		Tacka t2 = new Tacka(4.34, 5.63);
		t2.transliraj(4.2, 5.7);
		System.out.println(t2.opis());
		// Tacka (8.54, 11.33)

		Tacka t3 = t1;
		t3.transliraj(1, 1);
		System.out.println(t1.opis());
		// Tacka (14.0, 15.0)
	}
}
```

Dizajn i implementacija klase

- **Sakrivanje informacija**
  - **modifikatori pristupa (vidljivosti) *private* i *public***
  - Privatnim atributima i metodama možemo pristupati samo unutar klase, javni atributi i metodi su vidljivi i van klase
- Dizajn klase – definiše načine interakcije objekta sa drugim objektima
  - Interakcije idu kroz pozive metoda
  - skup zaglavlja javnih metoda koji se mogu pozvati nad objektom klase
- Implementacija klase – realizacija dizajna pri čemu možemo uvoditi pomoćne metode
- Enkapsulacija – sakrivanje detalja implementacije dizajna klase
  - Atributi i pomoćne metode definisane sa modifikatorom *private*
    - Metode objekta kontrolišu stanje objekta
    - Promena atributa klase ne utiče na druge klase (održavanje i modifikovanje programa)

Enkapsulacija

```
public class Tacka {
	private double x, y;

	public Tacka(double sx, double sy) {
		x = sx;
		y = sy;
	}

	public void transliraj(double dx, double dy) {
		x += dx;
		y += dy;
	}

	public String opis() {
		return "Tacka (" + x + ", " + y + ")";
	}
}
```

Enkapsulacija

```
public class TackeDemo {
    public static void main(String[] args) {
        Tacka t1 = new Tacka(12, 10);

        // ne prolazi kompajliranje
        // t1.x = 1;
        // t1.y = 2;

        t1.transliraj(1, 2);
        System.out.println(t1.opis());
    }
}
```

Get i set metode

- **Get metoda** – metoda koja vraća vrednost atributa
- **Set metoda** – metoda koja postavlja atribut na neku novu vrednost

```
public class Tacka {
        private double x, y;

        public Tacka(double sx, double sy) {
                ...
        }

        public void setX(double newX) { x = newX; }

        public void setY(double newY) { y = newY; }

        public double getX() { return x; }

        public double getY() { return y; }

        ...
}
```

Promenljive

- Imamo različite vrste promenljivih: atributi klase, parametri metoda, lokalne promenljive u telu metoda
- **Leksička pravila opsega – za neku pojavu identifikatora u kodu tražimo toj pojavi najbližu definiciju tipa**
  - Prvo u tekućem bloku, onda redom po nadgnježdenim blokovima, potom u listi parametara metoda, i na kraju među atributima klase
- Vrednost promenljive čiji je tip neka klasa je
  - referenca na objekat te klase
  - ***null***
- Ako referencijalna promenljiva ima vrednost ***null*** tada ona ne pokazuje ni na jedan objekat
- Vrednost ***null*** se može dodeliti bilo kojoj promenljivoj referencijalnog tipa

Referenca this

- Svaki objekat ima implicitno definisano polje koje se zove **this**
- **this je referenca na samog sebe**
- this referenca je zgodna kada imamo koliziju na nivou imena

```
public class Tacka {
    private double x, y;

    public Tacka(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public void setX(double x) {
        this.x = x;
    }

    ...
}
```

Klase su referencijalni tipovi

- **Samim tim klase mogu biti tipovi parametara i povratne vrednosti metoda**

```
public class Tacka {
	private double x, y;

	public Tacka(double x, double y) { ... }

	public Tacka transliraj(double dx, double dy) {
		return new Tacka(x + dx, y + dy);
	}

	public double distancaDo(Tacka druga) {
		double dx = x - druga.x;
		double dy = y - druga.y;
		return Math.sqrt(dx * dx + dy * dy);
	}

	...
}
```

Klasa može definisati više konstruktora

- **Moraju se razlikovati ili po broju parametara ili bar jedan parametar mora biti različitog tipa**

```
public class Tacka {
	private double x, y;

	public Tacka() {
		x = y = 0;
	}

	public Tacka(double xy) {
		x = y = xy;
	}

	public Tacka(double x, double y) {
		this.x = x;
		this.y = y;
	}

	...
}
```

Konstruktori može pozvati drugi konstruktor

- Koristimo ključnu reč this iza koje zadajemo vrednost parametara
- Na osnovu tipova parametara se određuje koji konstruktor se poziva

```
public class Tacka {
	private double x, y;

	public Tacka() {
		this(0);
	}

	public Tacka(double xy) {
		this(xy, xy);
	}

	public Tacka(double x, double y) {
		this.x = x;
		this.y = y;
	}
	...
}
```

Konstruktori može pozvati drugi konstruktor

- **Ako konstruktor poziva neki drugi konstruktor tada taj poziv mora biti prva naredba u konstruktoru**

```
public class Tacka {
	private double x, y;

	// ne prolazi kompajliranje!
	public Tacka() {
		x = 2;
		y = 3;
		this(10, 15);
	}

	public Tacka(double x, double y) {
		this.x = x;
		this.y = y;
	}
}
```

Preopterećeni (overloaded) metodi

- Klasa može definisati više metoda sa istim imenom
- Takvi metodi se moraju razlikovati ili po broj parametara ili se bar jedan mora razlikovati po tipu

```
public class Recenica {
	private String sadrzaj;

	public Recenica(String sadrzaj) {
		this.sadrzaj = sadrzaj;
	}

	public void dodaj(String rec) {
		sadrzaj += " " + rec;
	}

	public void dodaj(String prva, String druga) {
		sadrzaj += " " + prva + " " + druga;
	}

	public String getSadrzaj() { return sadrzaj; }
}
```

Promenljive referencijalnog tipa i operatori

- **Operatorom = se kopira vrednost reference, ne objekat**
- **Operatorima == i != se porede vrednosti referenci ne vrednosti objekata**

```
Recenica r1 = new Recenica("Ana voli Milovana");

// r2 pokazuje na isti objekat na koji pokazuje r1
// ne pravi se novi objekat!
Recenica r2 = r1;

Recenica r3 = new Recenica("Ana voli Milovana");
if (r3 == r1)
        System.out.println("Ovo se nece ispisati");
else
        System.out.println("Ovo se hoce ispisati");
```

Promenljive referencijalnog tipa

- Promenljive referencijalnog tipa se implicitno inicijalizuju na ***null***
- ***NullPointerException*** – izuzetak koji se pojavljuje kada preko reference koja je *null* probamo da pozovemo javni metod ili pristupimo javnom polju
- **Izuzetak – greška tokom izvršavanja programa**

```
// NullPointerException
Recenica r;
r.dodaj("Ana ");
r.dodaj("voli Milovana");
```

```
Exception in thread "main" java.lang.NullPointerException
        at ReceniceDemo.main(ReceniceDemo.java:68)
```

Klase i nizovi

- **Možemo kreirati nizove objekata – nizove za koje je neka klasa tip elemenata niza**
- **Nizovi su takođe promenljive referencijalnog tipa**

```
// niz recenica
Recenica[] r;

// u ovom trenutku r je null

// instanciramo niz
r = new Recenica[10];

// u ovom trenutku elementi niza su null

// instanciramo elemente niza
for (int i = 0; i < r.length; i++)
      r[i] = new Recenica("recenica " + i);
```

Statički atributi i metodi

- Atributi i metodi deklarisani koristeći ključnu reč **static**
- Statički atributi i metodi nisu vezani za objekte nego za klasu
  - Pristupamo im preko imena klase, a možemo i preko objekata
- **Statički atributi klase postoje nezavisno od objekata klase**
  - Statički atributi neke klase postoje nezavisno od toga kada i koliko puta je neka klasa intancirana, postoje i kada klasa nije instancirana nijednom
- Svaki objekat ima svoju (nezavisnu) kopiju nestatičkih atributa, svi objekti dele iste statičke atribute (ne kopiraju se)
- Statički metodi ne mogu pristupati nestatičkim atributima, samo statičkim
- Nestatičke metode mogu pristupati i statičkim i nestatičkim atributima

```java
public class Ucenik {
	private int id;
	private String ime;

	// auto-increment ID
	private static int idVal;

	public static void initID(int start) {
		idVal = start;
	}

	public Ucenik(String ime) {
		id = idVal++;
		this.ime = ime;
	}

	public String toString() {
		return "ID = " + id + ", " + ime;
	}
}
```

```java
public class UceniciDemo {
    public static void main(String[] args) {
        Ucenik.initID(101);

        Ucenik a = new Ucenik("Ana");
        System.out.println(a);
        // ID = 101, Ana

        Ucenik b = new Ucenik("Pera");
        System.out.println(b);
        // ID = 102, Pera
    }
}
```

Singleton pattern

```
class Brojac {
	private int val = 0;

	// singleton brojac
	private static Brojac singleton = null;

	// sakrivamo konstruktor
	private Brojac() {}

	public static Brojac getInstance() {
		if (singleton == null)
			singleton = new Brojac();

		return singleton;
	}

	public void inc()   { val++;       }
	public int getVal() { return val; }
}
```

Singleton pattern

```
// ne prolazi kompajliranje!
// Brojac br = new Brojac();

Brojac br = Brojac.getInstance();
for (int i = 0; i < 5; i++)
    br.inc();

Brojac br2 = Brojac.getInstance();
for (int i = 0; i < 5; i++)
    br2.inc();

System.out.println("Brojac = " + br.getVal());
// Brojac = 10
```