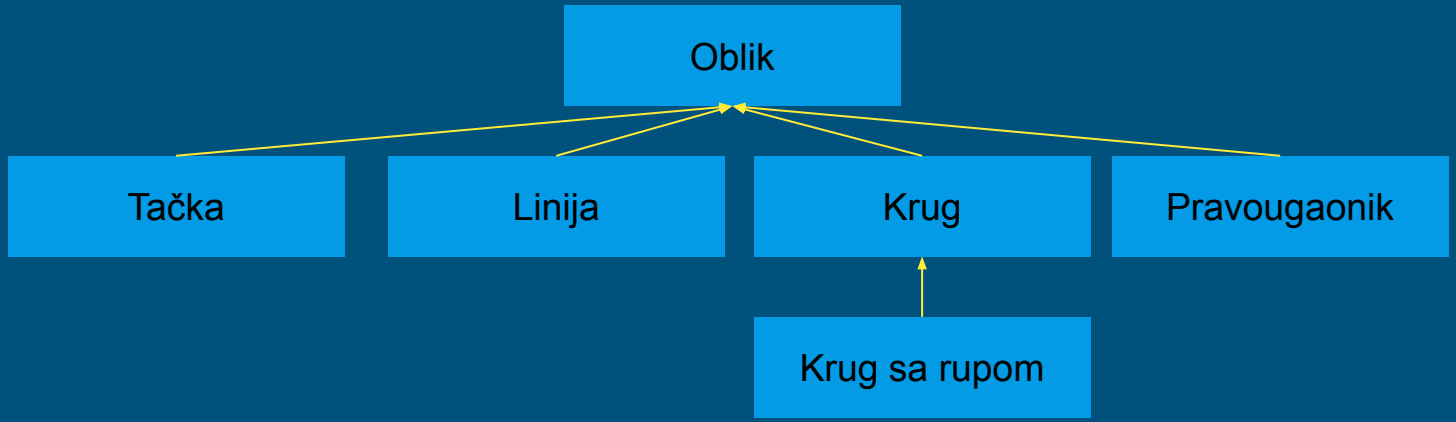
9. Apstraktne klase i interfejsi

Sadržaj

- Polimorfizam
- Apstraktne klase
- Interfejsi
- Dimaničko povezivanje

Hijerarhija proširivanja

- Analizom odnosa različitih klasa (vrsta) geometrijskih oblika možemo definisati klasu Oblik koja je vrh te hijerarhije



Buttom - up pristup

- Generalizacijom na osnovu zajedničkih osobina i operacija različitih klasa možemo definisanti novu klasu koja je osnovna za sve postmatrane
- Sve klase oblika imaju zajedničku osobinu `selected` i odgovarajuće metode pristupa `isSelected` i `setSelected`
- Uvođenjem klase `Shape` i izvođenjem svih klasa konkretnih oblika iz nje omogućava da se eliminiše duplirani kod u izvedenim klasama

Klasa Shape

```
public class Shape {
  protected boolean selected;

  public Shape() {}

  public Shape(boolean selected) {
    this.selected = selected;
  }

  public boolean isSelected() {
    return this.selected;
  }

  public void setSelected(boolean selected) {
    this.selected = selected;
  }
}
```

Modifikacija klase `Circle`

```
public class Circle extends Shape {
  protected Point center;
  protected int radius;

  public Circle(Point center, int radius) {
    super(); // podrazumeva se i ako se ne navede poziv
    this.center = center;
    this.radius = radius;
  }

  public Circle(Point center, int radius, boolean selected) {
    super(selected);
    this.center = center;
    this.radius = radius;
  }
  // metode pristupa, bez setSelected i isSelected
}
```

Polimorfizam

- Značenje “moć poprimanja više oblika”
- Objekti izvedenih klasa na različtite načina izvršavaju istu operaciju
- Implementacija istem metode `toString` u različitim klasama izvedenim iz klase `Object`
  - `Point (<x>, <y>)`
  - `Line (<startX>, <startY>) -> (<endX>, <endY>)`
  - `Circle center(<cx>, <cy>), radius = <r>`

Poziv `toString` kada nije redefinisana

```
Object[] objects = new Object[3];
objects[0] = new Point(5, 7);
objects[1] = new Line(new Point(1, 0), new Point(5, 0));
objects[2] = new Circle(new Point(5, 7), 6);
for (int i = 0; i < objects.length; i++)
  System.out.println(objects[i].toString());
```

Poziv `toString` kada jeste redefinisana

```
Object[] objects = new Object[3];
objects[0] = new Point(5, 7);
objects[1] = new Line(new Point(1, 0), new Point(5, 0));
objects[2] = new Circle(new Point(5, 7), 6);
for (int i = 0; i < objects.length; i++) {
  if (objects[i] instanceof Point)
    System.out.println(((Point)objects[i]).toString());
  else if (objects[i] instanceof Line)
    System.out.println(((Line)objects[i]).toString());
  else if (objects[i] instanceof Circle)
    System.out.println(((Circle)objects[i]).toString());
}
```

Dinamičko (kasno) povezivanje

- Kod sa prethodnog slajda će raditi i bez ispitivanja i konverzije tipa elementa niza
- Razlog za to je što Java podržava dinamičko ili kasno povezivanje
- U vreme izvršenja programa nad objektom će se izvršiti metoda
  - one klase kojom je objekat **inicijaliovan**,
  - a **ne** one klase kojom je referenca **deklarisana**
- Koja će se vrednost površine ispisati?

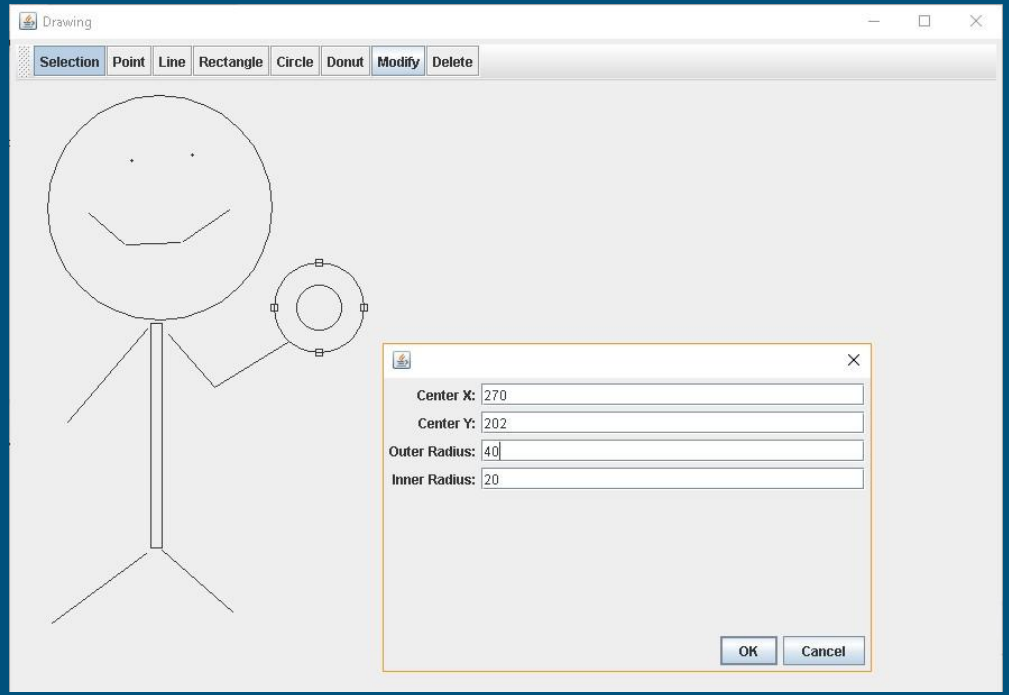
```
Circle c = new Donut(new Point(5, 6), 7, 4);
System.out.println(c.area());
```

Pojam apstraktne klase

- Ako posmatramo sve vrste oblika, oni su ili tačke ili linije ili krugovi ili krugovi sa rupom ili…
- Oblik je apstraktna klasa koja nam je poslužila da u nju smestimo sve što je zajedničko za oblike
- Mogu da se uoče operacije koje ima svaka izvedena klasa ali se te operacije izvršavaju na različite načine
  - svaki oblik se crta na različit način
  - određivanje sadržavanja zadate tačke se za svaki oblik određuje na različit način

Studija slučaja i operacije nad oblicima

- Svaki oblik mora “znati” da se iscrta na crtežu
- Svaki oblik mora da detektuje da li sadrži neko x,y
- Ove metode nije moguće implemenirati u klasi `Shape` jer se različito realizuju u izvedenim klasama



Apstraktna klasa u OO programiranju

- Sadrži apstraktne metode - metode bez implementacije
- Može da sadrži metode koje imaju implementaciju
- Ne može biti instancirana
- Ima konstruktor(e) za inicijalizaciju obeležja definisanih u toj klasi

```
public abstract class Shape {
  protected boolean selected;
  public Shape() {}
  public Shape(boolean selected) {
    this.selected = selected;
  }
  // ...
}
```

Apstraktne metode

- Moraju biti redefinisane u izvedenim klasama
- Ukoliko neka od apstraktnih metoda nije redefinisana, izvedena klasa mora biti deklarisana kao apstraktna

```
public abstract class Shape {
  // ...
  public abstract void draw(Graphics g);
  public abstract boolean contains(int x, int y);
}
```

Iscrtavanje bez apstraktnih metoda

```
public class Drawing … {
  private Object[] shapes = new Object[10];
  // …
  public void paint(Graphics g) {
    for (int i = 0; i < shapes.length; i++) {
      Object obj = shapes[i];
      if (obj instanceof Point)
        ((Point)obj).draw(g);
      else if (obj instanceof Line)
        ((Line)obj).draw(g);
      else if …
      // za svaku vrstu oblika po jedan if
    }
  }
}
```

Iscrtavanje sa apstraktnim metodama

```
public class Drawing … {
  private Shape[] shapes = new Shape[10];
  // …

  public void paint(Graphics g) {
    for (int i = 0; i < shapes.length; i++)
        shapes[i].draw(g);
  }
}
```

Interfejsi

- Komponenta kojoj su sve metode javne i apstraktne
  - izostavljaju se modifikatori `public abstract` jer se podrazumevaju
- Može da ima promenljive koje su javne statičke konstante
- Klase **implementiraju** interfejse
- Jedna klasa može da implementira više interfejsa

```
public class Donut extends Circle  implements Movable, Comparable {...
```

- Klase koje implementiraju interfejs ali ne implementiraju sve metode moraju biti deklarisane kao apstraktne

Primer korišćenja interfejsa

- Pretpostavimo da želimo da sortiramo krugove sa rupom
- Da bi se izvršilo sortiranje potrebno je definisati kriterijum, na primer:
  - po većem spoljnom radijusu
  - po manjem unutrašnjem radijusu
  - po ukupnoj površini…
- Pored definisanja kriterijuma sortiranja, potrebno je isprogramirati algoritam sortiranja (postoji više različih po složenosti i performansama)
  - *bubble sort*
  - *quick sort*
  - *merge sort...*

Definisanje kriterijuma sortiranja

- Interfejs `Comparable` definiše jednu metodu u kojoj treba da bude implementiran kriterijum sortiranja

```
public interface Comparable {
  int compareTo(Object o);
}
```

- Metoda treba da vrati
  - 0 - ako su objekat za koji se poziva metoda i parametar isti (po kriterijumu sortiranja)
  - > 0 - ako je objekat za koji se poziva metoda veći od parametra
  - < 0 - ako je objekat za koji se poziva metoda manji od parametra

Implementacija poređenja za `Donut`

```
public class Donut extends Circle implements Comparable {
  // …
  public int compareTo(Object o) {
    Donut d = (Donut)o;
    return this.area() - d.area();
  }
}
```

Implementacija algoritma sortiranja

- Dobra vest je da klasa `java.util.Arrays` ima statičku metodu
  `public static void sort(Object[])`
  koja sortira prosleđeni niz objekata ukoliko ta klasa implementira `Comparable`
- Specifikacija interfejsa `Comparable` omogućila je da tvorac (programer) klase `Arrays` koji nije unapred mogao da zna da li ćemo sortirati krugove sa rupom ili objekte neke druge klase obezbedi sortiranje u skladu sa specifikacijom metode `compareTo`
- Tvorac klase `Donut` ne mora da zna da isprogramira algoritam sortiranja

Bubble sort koji koristi `Comparable`

```
public class BubbleSort {
  public static void sort(Comparable[] items) {
    for (int i = 1; i < items.length; i++) {
      for (int j = 0; j < items.length - 1; j++) {
        if (items[j].compareTo(items[j+1]) > 0) {
          Comparable tempitem = items[j+1];
          items[j+1] = items[j];
          items[j] = tempitem;
        }
      }
    }
  }
```

Rezime

- Polimorfizam omogućava da se ista operacija u različitim klasama izvršava na različit način
- Dinamičko povezivanje garantuje da se prilikom izvršenja programa poziva metoda klase kojim je objekat inicijalizovan
- Apstraktne klase omogućavaju da se izveden klase obavežu da implementiraju potrebne metode
- Interfejsi predstavljaju vrstu ugovora između različitih komponenti sistema