6. Kompozicija

Sadržaj

- Kompozicija
- Klasa `String`
- Ovojne (engl. *wrapper*) klase

Pojam kompozicije

- Predstavlja odnos “*has a*” između složene klase i komponente
- Od primitivnih tipova možemo komponovati klasu (složeni tip)
  - primer klase `Point`
  - obeležja su tipa `int` i `boolean`
- Od postojećih klasa možemo dalje komponovati nove klase
  - klasa `Bicycle` može biti deklarisana kao:

```
public class Bicycle {
  private Wheel frontWheel;
  private Wheel rearWheel;
  private Point centerOfMass;
  ...
}
```

Kompozicija primitivnih i složenih tipova

- Linija (duž) se može definisati pomoću obeležja
  - `startX`, `startY`, `endX`, `endY` tipa `int`
  - `startPoint`, `endPoint` tipa `Point`
- Implementacija metode za izračunavanje dužine linije u prvom slučaju

```
public double length() {
  int dx = this.startX - this.endX;
  int dy = this.startY - this.endY;
  return Math.sqrt(dx * dx + dy * dy);
}
```

- Implementacija metode za izračunavanje dužine linije u drugom slučaju

```
public double length() {
  return this.startPoint.distance(this.endPoint);
}
```

Definicija klase `Line` (1)

```
public class Line {
  private Point startPoint;
  private Point endPoint;
  private boolean selected;

  public Line(Point startPoint, Point endPoint) {
    this.startPoint = startPoint;
    this.endPoint = endPoint;
  }
  public Line(Point startPoint, Point endPoint, boolean selected) {
    this(startPoint, endPoint);
    this.selected = selected;
  }
  …
```

Definicija klase `Line` (2)

```
  public Point getStartPoint() {
    return this.startPoint;
  }

  public void setStartPoint(Point startPoint) {
    this.startPoint = startPoint;
  }
  // … ostale metode pristupa

  public double length() {
    return this.startPoint.distance(this.endPoint);
  }
}
```

Rad sa obeležjima složenog tipa

- Obeležja složenog tipa tretiraju se kao i svaka druga referenca na objekat

```
public class TestLine {
  public static void main() {
    Point p1 = new Point(4, 5);
    Point p2 = new Point(6, 7);
    Line l = new Line(p1, p2);
    l.getStartPoint().setX(7);
    if (l.getEndPoint().getY() > 4)
      ...
  }
}
```

Definicija klase `Circle`

- Geometrija kruga određena je centralnom tačkom (centrom) i poluprečnikom (radijusom)

```
public class Circle {
  private Point center;
  private int radius;
  private boolean selected;
  // konstruktori i metode pristupa
}
```

Metoda `area` u klasi `Circle`

- Pretpostavimo da je u klasi `Circle` potrebno obezbediti metodu `area` koja vraća površinu kruga
- Metoda ne bi imala parametre jer su sve potrebne informacije sadržane u obeležjima

```
public double area() {
  return this.radius * this.radius * Math.PI;
}
```

Metoda `contains` u klasi `Circle`

- Pretpostavimo da je u klasi `Circle` potrebno obezbediti metodu `contains` koja detektuje da li krug sadrži prosleđenu tačku
- Metoda bi proveravala da li je udaljenost prosleđene tačke od centra manja od poluprečnika ili ne

```
public boolean contains(Point p) {
  return (this.center.distance(p) <= this.radius);
}
```

- Ukoliko bi centar bio definisan pomoću obeležja `x` i `y` tipa `int`, metoda bi morala da sadrži implementaciju Pitagorine teoreme

Klasa `String`

- Pripada paketu `java.lang`
- Ima neke specifičnosti koje nemaju druge klase
  - moguće kreiranje objekta bez poziva konstruktora:
    ```
    String str1 = "abc";
    ```
  - operator "+" omogućava spajanje nizova znakova
    ```
    String str2 = str1 + "def";
    ```
- Objekat klase `String` nije moguće modifikovati - uvek se pravi novi objekat
  - posledica ovoga je pad performansi pri radu sa velikim količinama podataka
  - klasa `StringBuilder` rešava pomenuti problem
- Konverzija vrednosti primitivnog tipa u niz znakova je uvek moguća
- Konverzija niza znakova u vrednost nekog primitivnog tipa nije uvek moguća

Konverzija primitivnog tipa u niz znakova

- Klasi `String` sadrži statičku metodu `valueOf` za svaki primitivni tip podatka

```
int someIntValue = 456;
String strValue1 = String.valueOf(someIntValue);
boolean someBooleanValue = true;
String strValue2 = String.valueOf(someBooleanValue);
String concat = strValue1 + strValue2; // “456true”
```

Konverzija niza znakova u primitivni tip

- Postoje ovojne (engl. *wrapper*) klase za odgovarajuće primitivne tipove:
  - `Integer`
  - `Boolean`
  - `Double…`
- Prilikom konverzije niza znakova u primitivni tip može doći do nepredviđene situacije:

```
int intValue = Integer.parseInt(“123”);
String str = “false”;
boolean booleanValue = Boolean.parseBoolean(str);
double dblValue = Double.parseDouble(str); // greska!
```

Metode klase `String`

- Klasa `String` obezbeđuje metode za rad sa nizom znakova:
  - `public int length()` - vraća broj znakova
  - `public char charAt(int)` - vraća znak na zadatoj poziciji
  - `public int indexOf(String)` - vraća poziciju zadatog stringa
  - `public String replaceAll(String, String)` - vraća novi string u kom je zamenjen zadati podstring novim podstringom
  - `public String[] split(String)` - vraća niz znakova nastao deljenjem stringa po zadatom podstringu
  - `public String substring(int)` - vraća podstring od zadate pozicije do kraja
  - `public String substring(int, int)` - vraća podstring od zadate početne pozicije do zadate krajnje pozicije

Primeri poziva metoda klase `String`

```
String str = “neki niz znakova”;
int length = str.length();                  // 16
String[] strArray = str.split(“ ”);         // {“neki”, “niz”, “znakova”}
char tenthChar = str.charAt(9);             // ‘z’
String rpl = str.replaceAll(“ ”, “-”);      // “neki-niz-znakova”
String sub1 = str.substring(3);             // “i niz znakova”
String sub2 = str.substring(3, 8);          // “i niz”
```

Rezime

- Kompozicija omogućava višestruko korišćenje koda
- Kompozicijom se izbegava dupliranje koda što olakšava održavanje koda
- Obeležja složenog tipa tretiraju se kao i svaka referenca na objekat tog tipa
- Klasa `String` omogućava rad sa nizovima znakova
- Ovojne klase omogućavaju konverziju nizova znakova u vrednosti primitivnog tipa