Java tečaj

3. dio

Apstraktni razredi.
Sučelja. Anonimni razredi.

Apstraktni razred

- Pretpostavimo da na raspolaganju imamo razred Slika:
  - `public int getSirina() {...}`
  - `public int getVisina() {...}`
  - `public void upaliTocku(int x, int y) {...}`
  - `public void ugasiTocku(int x, int y) {...}`

Apstraktni razred

- Kako najlakše postići da se naši geometrijski likovi mogu iscrtavati?
- Proširimo razred `GeometrijskiLik` s dvije metode:
  - `public boolean sadrziTocku(int x, int y) {...}`
  - `public void popuniLik(Slika slika) {...}`

Apstraktni razred

- Zadatak metode `sadrziTocku` jest utvrditi sadrži li trenutni lik predanu točku (x,y)

No kako to utvrditi? `GeometrijskiLik` je bazni razred za svaki lik – informacija koje točke lik sadrži ovdje nije dostupna

Apstraktni razred

- Rješenje 1
  - `sadržiTočku` uvijek vraća `false`
  - Svaki razred koji nasljedi `GeometrijskiLik` **mora** override-ati navedenu metodu kako bi ispravno vraćao informaciju o sadržanim točkama

Apstraktni razred

- Zadatak metode **`popuniLik`** jest nacrtati sliku lika
- Nemoguće bez da znamo koji je to lik?
  - Ne! Možda neefikasno, ali ne i nemoguće!
  - Za svaku točku slike pitaj lik sadrži li tu točku, i ako da, upali je!

Apstraktni razred

```
public class GeometrijskiLik {
   ...
   public boolean sadrziTocku(int x, int y) {
      return false;
   }
   public void popuniLik(Slika slika) {
      ...
   }
}
```

Apstraktni razred

```
public class Linija extends GeometrijskiLik {
   ...
   public boolean sadrziTocku(int x, int y) {
       … nova implementacija …
   }
}
```

Apstraktni razred

- Zahvaljujući polimorfizmu, metoda `popuniLik` će za svaki konkretan lik pozivati njegovu redefiniranu metodu `sadrziTocku(…)`, i uspješno obaviti crtanje

Apstraktni razred – primjer 1

```
public static void main(String[] args) {
  Slika slika = new Slika(40, 40);

  Linija l1 = new Linija(10, 10, 30, 30);
  l1.popuniLik(slika);

  Pravokutnik p = new Pravokutnik(2, 20, 10, 15);
  p.popuniLik(slika);

  Pravokutnik p2 = new Pravokutnik(30, 4, 2, 2);
  p2.popuniLik(slika);

  slika.nacrtajSliku(System.out);
}
```

Apstraktni razred

- Opisana metoda `popuniLik` je spora!
- Konkretni likovi mogu obaviti redefiniranje i te metode, i dodatno je ubrzati.
- Primjerice
  - `Linija`: bresenhamov postupak
  - `Pravokutnik`: dvije ograničene `for` petlje

Apstraktni razred

- Ponovno rješenje 1
  - `sadržiTočku` uvijek vraća `false`
  - Svaki razred koji nasljedi `GeometrijskiLik` **mora** nadjačati navedenu metodu kako bi ispravno vraćao informaciju o sadržanim točkama
  - ----------
  - Kako postići ovaj **mora**?

Apstraktni razred

- Rješenje 2
  - Objektno orijentirana paradigma poznaje pojam apstraktne metode – metoda za koju se definira signatura, ali ne i implementacija:
    public abstract boolean sadrziTocku(int x, int y);
  - Razred koji sadrži barem jednu ovakvu metodu također je apstraktan i mora biti definiran kao:
    public abstract class ImeRazreda {…}

Apstraktni razred

- Rješenje 2
  - Nije moguće stvarati primjerke apstraktnih razreda jer nisu potpuno definirani (nedostaju neke metode)
  - Apstraktni razred nužno je naslijediti!
  - Razred koji ga nasljeđuje može definirati sve njegove apstraktne metode, ali i ne mora – tada je i on apstraktan i ne možemo stvarati njegove primjerke

Apstraktni razred

- Uporaba apstraktnih razreda
  - Ponuditi osnovu za definiranje drugih razreda
  - Nudi implementaciju svih dijeljenih algoritama
  - Metode u kojima se konkretni razredi razlikuju definira apstraktnima, čime ih prisiljava da ih definiraju

Sučelja

- Sučelje možemo poistovjetiti sa potpuno apstraktnim razredom – razredom koji definira niz apstraktnih metoda
- Java ipak razlikuje razred od sučelja
- Sučelje se definira ključnom riječi **interface** (a ne **class** kao kod razreda)

Sučelja

- Sučelje na elementarnoj razini možemo poistovijetiti s popisom metoda koje svaki razred koji ga implementira mora imati
- Terminološki, razredi se nasljeđuju:
  class A extends B {}
  a sučelja implementiraju:
  class A implements I {}

Sučelja

- Za razliku od modela nasljeđivanja gdje razred može imati samo jednog roditelja, Java razredima dozvoljava da implementiraju proizvoljan broj sučelja:

```
class A
  implements I1, I2, I3 {...}
```

Sučelja

- Sučelja i nasljeđivanje se međusobno ne isključuju! Primjerice, ispravno je napisati:

```
class B
  extends A
  implements I1, I2, I3 {...}
```

Sučelja

- Navedemo li samo signaturu metode:
  - Metoda je po definiciji javna i apstraktna
  - Nije moguće smanjiti vidljivost metode
  - Sljedeći primjer definira dvije istovrsne metode (preferiramo kraće – čitljivije!)

```
interface X {
   int getNumber1();
   public abstract int getNumber2();
}
```

Sučelja

- Od Jave 8, sučelja mogu ponuditi i pretpostavljene implementacije metoda
  - Ispred metode navodi se ključna riječ **`default`**
  - Ta se implementacija koristi ako razred ne ponudi svoju

```
interface X {
  default int getNumber() { return 42; }
}
```

Sučelja

- Od Jave 8, sučelja mogu ponuditi i pretpostavljene implementacije metoda
  - Pretpostavljene metode mogu pozivati druge metode sučelja; npr.

```
interface Collection {
  int size();
  default boolean isEmpty() {
    return size()==0;
  }
}
```

Sučelja

- Sučelje može definirati konstante
  - One se tretiraju kao javne, statičke i finalne
  - Nije moguće smanjiti vidljivost
  - Sljedeći primjer definira dvije istovrsne konstante

```
interface MathConstants {
  double PI = 3.14;
  public static final E = 2.71;
}
```

Sučelja

- Od Jave 8, sučelja mogu ponuditi i statičke metode
  - Ispred metode navodi se ključna riječ **`static`**
  - Po definiciji metoda je javna i nije moguće smanjiti vidljivost

```
interface X {
   static int getNumber() { return 42; }
}
```

Sučelja vs apstraktni razredi

- Apstraktni razred je – razred
  - Dobro rješenje za definirati osnovu za izvođenje novih razreda
  - Može ponuditi implementaciju zajedničkih algoritama

Sučelja vs apstraktni razredi

- Sučelje je – “popis”!
  - Dobro rješenje za dodavanje “karakteristika” postojećim razredima
  - Funkcionira neovisno o strukturi nasljeđivanja: razred koji već ima definiranu strukturu nasljeđivanja može implementirati proizvoljna sučelja

Sučelja vs apstraktni razredi

- Sučelje je – “popis”!
  - Izuzetno često korišteni u Javi
  - Susrest ćemo se s njima uskoro kod Collection Frameworka i Swinga
  - Omogućava razdvajanje implementacije i “obećane” funkcionalnosti
  - Može se shvatiti i kao “pogled” kroz koji se vidi neki objekt

Primjer uporabe sučelja

- Objekti koji jedan broj transformiraju u neki drugi

```
interface Transformer {
  double transform(double value);
}

class Add3Transformer implements Transformer {
  public double transform(double value) {
    return value + 3;
  }
}

Transformer t = new Add3Transformer();
double x = t.transform(2.5);
```

Primjer uporabe sučelja

- "Anonimni razred" je konstrukt koji omogućava stvaranje primjerka razreda koji definiramo na mjestu na kojem stvaramo i sam objekt

```
Transformer t = new Transformer() {
  public double transform(double value) {
    return value + 3;
  }
}
double x = t.transform(2.5);
```

Primjer uporabe sučelja

- Metode anonimnog razreda vide iz vanjskog konteksta samo lokalne varijable koje su konstante ili efektivno konstante

```
int k = 7; k++;
int j = 8;
final int OFFSET = 3;
Transformer t = new Transformer() {
  public double transform(double value) {
    return value + OFFSET;
  }
}
double x = t.transform(2.5);
```

Ne vidi se u anonimnom razredu

Vidi se u anonimnom razredu

Primjer uporabe sučelja

- Konceptualno, lambda izraz je sintaksna pokrata za stvaranje primjerka anonimnog razreda

```
Transformer t = new Transformer() {
  public double transform(double value) {
    return value + OFFSET;
  }
}
Transformer t2 = (double value) -> {
    return value + OFFSET;
  };
```

Primjer uporabe sučelja

- Konceptualno, lambda izraz je sintaksna pokrata za stvaranje primjerka anonimnog razreda

```
Transformer t = new Transformer() {
   public double transform(double value) {
      return value + OFFSET;
   }
}
Transformer t2 = value -> {
      return value + OFFSET;
   };
```

Primjer uporabe sučelja

- Konceptualno, lambda izraz je sintaksna pokrata za stvaranje primjerka anonimnog razreda

```
Transformer t = new Transformer() {
  public double transform(double value) {
    return value + OFFSET;
  }
}
Transformer t2 = value -> value + OFFSET;
```

Primjer uporabe sučelja

- Konceptualno, lambda izraz je sintaksna pokrata za stvaranje primjerka anonimnog razreda

```
Transformer t = new Transformer() {
  public double transform(double value) {
    return Math.sin(value);
  }
}
Transformer t2 = Math::sin;
```

Primjer uporabe sučelja

- Bitna razlika između lambde i anonimnog razreda
  - Na razini jezika: lambda ne unosi novi doseg; anonimni razred to čini pa u njemu this je referenca na njegov objekt
  - Lambdu možemo definirati samo nad sučeljem koje ima **jednu** apstraktnu metodu
    - Takva sučelja nazivamo funkcijskim sučeljima (@FunctionalInterface)

Primjer uporabe sučelja

- Bitna razlika između lambde i anonimnog razreda
  - Na razini JVM-a: lambda ne generira zasebnu class datoteku (što radi definicija anonimnog razreda) već svoj kod obično "donira" u class datoteku matičnog razreda; brže za “učitavanje”

Primjer uporabe sučelja

- Bitna razlika između lambde i anonimnog razreda
  - Objekti definirani kao lambde koje ne hvataju lokalne varijable izvana (tzv. *non-capturing lambdas*) se keširaju na mjestu stvaranja (ušteda memorije); sljedeći kod alocira samo jedan objekt iako je u petlji

```
double[] polje = {1,2,3};
for(int i = 0; i < polje.length; i++) {
  Transformer t2 = Math::sin;
  polje[i] = t2.transform(polje[i]);
}
```

Ključna riječ var

- Lokalne varijable možemo definirati s **`var`** umjesto navođenjem tipa

```
Transformer t = new Transformer() {
   double sum = 0.0;
   public double transform(double value) {
      sum += value; return value + 1;
   }
}
double x = t.transform(2.5);
System.out.println(t.sum);
```

Compile error! Transformer nema sum!

Ključna riječ var

- Lokalne varijable možemo definirati s **`var`** umjesto navođenjem tipa

```
var t = new Transformer() {
  double sum = 0.0;
  public double transform(double value) {
    sum += value; return value + 1;
  }
}
double x = t.transform(2.5);
System.out.println(t.sum);
Transformer t2 = t;
```

OK! "t" je primjerak razreda koji ima sum!

OK! "t" je primjerak razreda koji je "kastabilan" u Transformer.