Objektno orijentirano programiranje

4: Nasljeđivanje. Polimorfizam

Creative Commons

You are free to

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material

under the following terms

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **NonCommercial** — You may not use the material for commercial purposes.
- **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

- https://creativecommons.org/licenses/by-nc-sa/4.0/

Klasa *Object*

- Java sadrži klasu *Object* te je moguće kreirati objekte klase *Object*
  - …iako nije pretjerano korisno

- Značaj klase *Object* je u hijerarhijskoj organizaciji klasa pri čemu je klasa *Object* korijen takvog stabla

- Svaki referencijski tip (engl. *reference type*)

  – to su npr. klase, ali ne i primitivni tipovi –

  je nastao iz klase *Object* i u sebi sadrži metode iz klase *Object*

Upcast i *downcast*

- Adresa objekta bilo kojeg tipa se može spremiti u referencu općenitijeg tipa (***upcast***) – u ovom slučaju tipa *Object*
  - Koristeći tu referencu mogu se pozvati samo metode iz klase *Object*
- Pokušaj definiranja reference konkretnijeg tipa (npr. *Point*) iz općenitije reference (u ovom slučaju *Object)* naziva se (***downcast***)
  - Može uzrokovati probleme (bit će kasnije obrađeno)
- *Upcast* i *downcast* ne mijenjaju objekt na hrpi

…04_InheritancePolymorphism/hr/fer/oop/objectmethods/*.java

```
Point p1 = new Point(2, 5);
...
System.out.println(p1.getX());
Object o1 = p1;  //upcast
   System.out.println(o1.getX()); //compile error
Point po1 = (Point) o1; //downcast
System.out.println(po1.getX());
System.out.println(p1 == po1);
```

Usporedba i ispis objekata

- U prethodnim predavanjima korištena je vlastita metoda za ispis točke (*print*) i usporedbu s drugom točkom (*isEqualTo*)
  - tipična funkcionalnost za većinu klasa
- Java već „ima” ove metode u klasi *Object*

  boolean equals(Object o)
  - uspoređuje objekt nad kojim je metoda pozvana s drugim objektom

  String toString()
  - vraća string s podacima o objektu nad kojim je metoda pozvana
    - takav string se može naknadno ispisati
- Navedene (i još neke) metode su naslijeđene te ih imamo u našoj klasi, iako ih nismo eksplicitno napisali

Zašto nasljeđivanje?

- Želimo napraviti novu klasu, a već postoji klasa koja uključuje nešto što trebamo (kôd, članske varijable, metode…)?

- Umjesto pisanja klase ispočetka, nasljeđivanjem imamo mogućnost korištenja već postojećih metoda

- Novu klasu „obogatimo” svojstvima koja nedostaju - specifičnim samo za tu klasu

Korištenje naslijeđenih *equals* i *toString*

- Naslijeđene metode iz klase *Object* u ovom primjeru ne daju željene rezultate
  - *equals* iz klase *Object* uspoređuje radi li se o istim objektima u memoriji (uspoređuje reference, a ne sadržaj objekta)
  - *toString* vraća puno ime klase i određeni broj (*hashcode*)

…04_InheritancePolymorphism/hr/fer/oop/objectmethods/*.java

```
package hr.fer.oop.objectmethods;
public class Main {
	public static void main(String[] args) {
		Point p1 = new Point(2, 5);
		System.out.println(p1.toString());
		Point p2 = new Point(2, 5);
		Point p3 = p2;
		System.out.println(p1.equals(p2));
		System.out.println(p2.equals(p3));
	}
}
```

```
hr.fer.oop.objectmethods.Point@d716361
false
true
```

Nadjačavanje *equals* i *toString*

- Kad je naslijeđeno ponašanje metoda *equals* i *toString* neodgovarajuće, mogu se napisati prikladne implementacije
  - Postupak se naziva **nadjačavanje** – engl. ***override***
  - Oznaka *@Override* nije nužna, ali prevodiocu sugerira našu namjeru
    - U protivnom javlja upozorenje o postojanju metode istog imena i argumenata u stablu nasljeđivanja

…04_InheritancePolymorphism/hr/fer/oop/override/Point.java

```
package hr.fer.oop.override;
public class Point {
    @Override
    public String toString(){
       return String.format("(%.2f, %.2f)",  x,  y);
    }
    @Override
    public boolean equals(Object obj) {
       Point other = (Point) obj;     // downcast
       return Math.abs(x-other.x)<1E-8 && Math.abs(y-other.y)<1E-8;
    }
```

Korištenje nadjačanih *equals* i *toString*

…04_InheritancePolymorphism/hr/fer/oop/override/*.java

```
package hr.fer.oop.override;
public class Main {
    public static void main(String[] args) {
        Point p1 = new Point(2, 5);
        Point p2 = new Point(2, 5);
        System.out.println("p1.equals(p2) : " + p1.equals(p2));
        p1.setX(1);
        p1.setY(2);
        System.out.println("p1.equals(p2) : " + p1.equals(p2));
        System.out.println(p1);
        System.out.println(p2);
```

```
p1.equals(p2) : true
p1.equals(p2) : false
(1.00, 2.00)
(2.00, 5.00)
```

Nadjačavanje i preopterećivanje

- Nadjačavanje – engl. *override*
- Preopterećivanje – engl. *overload*
- Nadjačavanje definira novo ponašanje naslijeđene metode
  - Metoda mora imati isti potpis (naziv, povratni tip, broj i tipovi argumenata)
    - * Java omogućava da povratni tip u nekim slučajevima može biti i drugačiji – detaljnije kasnije u slajdu istog naslova
  - Napomena: novo ponašanje se primjenjuje na klasu u kojoj je došlo do nadjačavanja i na klase ispod u hijerarhiji
- Preopterećivanjem se definira dodatna metoda istog imena, ali drugačijeg potpisa

Bazna i izvedena klasa

- **Bazna** klasa (osnovna klasa, natklasa, klasa roditelj) je klasa iz koje se izvodi neka druga klasa
  - engl. ***base class, superclass***

- Klasa koja nasljeđuje neku klasu naziva se **izvedena** klasa (potklasa, klasa dijete)
  - engl. ***derived class, subclass***

- Izvedena klasa je specijalizacija bazne klase

- Bazna klasa je generalizacija svojih izvedenih klasa

Nasljeđivanje

- Izvedena klasa se sastoji od vlastitih članova i članova bazne klase
- Može pristupiti članovima (varijablama, metodama) označenim s *public* ili *protected*
  - … te onim bez modifikatora, ako se nalazi u istom paketu kao bazna klasa
  - ne može pristupiti privatnim članovima bazne klase

- Konstruktori se ne nasljeđuju, ali se mogu pozivati iz izvedene klase

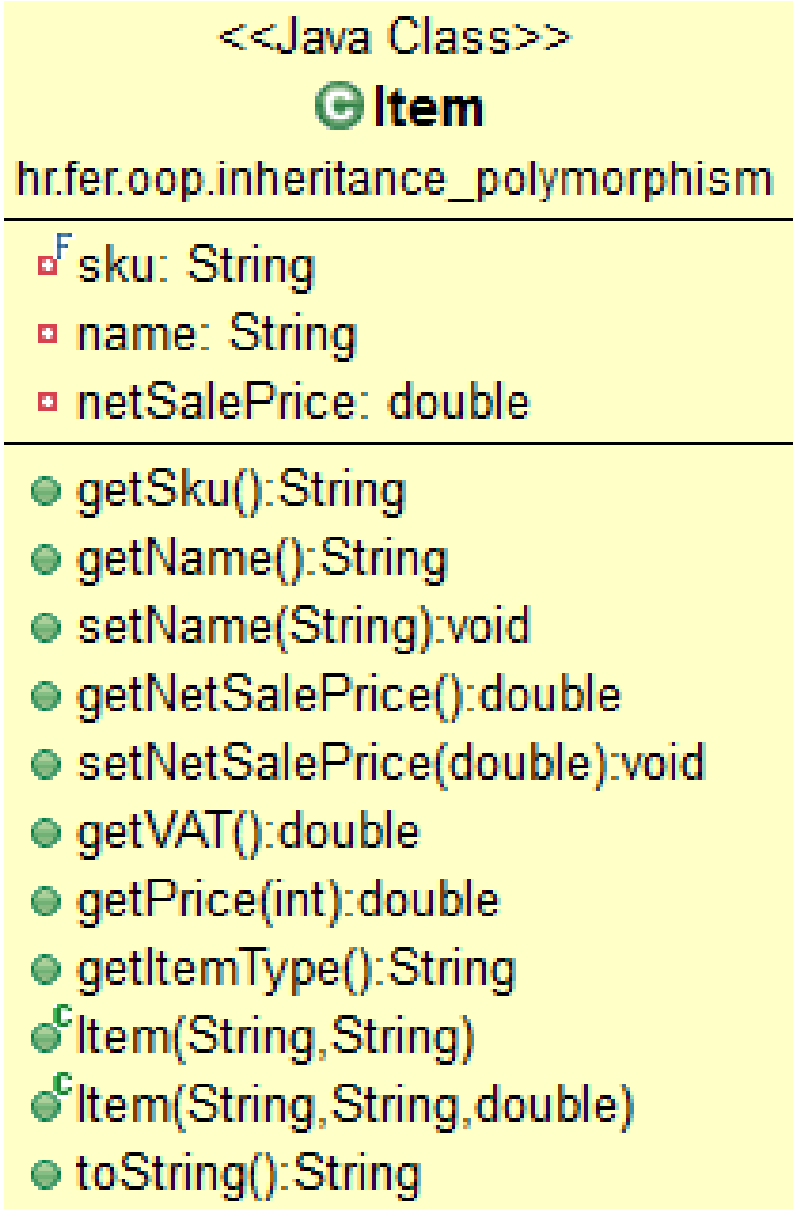
- U Javi sve klase osim klase *Object* imaju jednu i samo jednu direktnu klasu roditelja.
  - Ako to nije neka eksplicitno navedena klasa, onda je to klasa *Object.*

Kratki opis primjera s nasljeđivanjem

- Modeliramo artikle koji se prodaju u nekom dućanu.
- Primjer je pojednostavljen, jer nećemo modelirati sve vrste artikala, već samo one koji će poslužiti za razumijevanje osnovnih pojmova vezanih za nasljeđivanje i polimorfizam.
- Stoga u ovom primjeru modeliramo sljedeća svojstva/metode:
  - šifru (SKU – Stock Keeping Unit)
  - naziv
  - kategoriju proizvoda (npr. hrana, piće, odjeća ….)
  - jediničnu cijenu
  - stopu PDV-a (25%)
  - cijenu za n komada
    - načelno *n * jedinična cijena uvećana za PDV*, ali može npr. i 2+1 GRATIS…

Artikl

- klasa Item
- atributi (članske varijable) su privatni
  - pišemo gettere i settere
- Kroz programski kôd je moguće promijeniti samo naziv i iznos.
  - kategorija artikla je nepromjenjiva
  - jednom postavljena šifra artikla uvijek ista
  - PDV treba vrijediti za cijelu kategoriju proizvoda, a ne za pojedinačni proizvod
- dva konstruktora s argumentima
  - šifra, naziv
  - šifra, naziv, jedinična cijena
- metoda *toString* iz klase *Object* izmijenjena tako da se ispisuje šifra - naziv



Klasa *Item* (1)

- Artikl konstruiramo predajući šifru i naziv ili šifru, naziv i cijenu artikla.

```
package hr.fer.oop.inheritance_polymorphism;
public class Item {
   ...
   public Item(String sku, String name){
     this(sku, name, 0);
   }
   public Item(String sku, String name, double price){
     this.sku = sku;
     this.name = name;
     this.netSalePrice = price;
   }
  ...
```

- Npr. `new Item("1256", "Domaćica")`

Klasa *Item* (2)

- Za vrstu artikla vraćamo prazni String, jer ne možemo znati o kojoj vrsti artikla se radi (modeliramo općenito za bilo koji artikl).
- PDV je 25%
  - tvrdo kodiran zbog ilustracije nadjačavanja metoda. U stvarnom primjeru bi bio evidentiran u konfiguracijskoj datoteci ili u bazi podataka
- Za dohvat koristimo *gettere*, a ne direktno varijablu (razlog uskoro).

```
package hr.fer.oop.inheritance_polymorphism;
public class Item {
  ...
  public double getVAT(){ return 0.25; }
  public double getPrice(int count){
    return count * getNetSalePrice() * ( 1 + getVAT());
  }
  public String getItemType(){ return ""; }
  @Override
  public String toString() {
    return String.format("%s - %s", getSku(), getName());
  }
```

Primjer nasljeđivanja – hrana, piće i odjeća

- Hrana, piće i odjeća su artikli i imaju šifru, naziv i cijenu, ali svaki od njih ima nešto specifično baš za tu vrstu artikla, npr.
  - hrana (klasa *Food*) ima rok trajanja i masu
  - piće (klasa *Beverage*) ima volumen
  - odjeća (klasa *Cloth*) ima veličinu
  - naslijedimo postojeću funkcionalnost klase *Item*
    - npr. izračun cijene (mogao je npr. biti i složeniji od neto + PDV)
  - dodajemo funkcionalnost specifičnu za pojedinu klasu
- Za nasljeđivanje se koristi ključna riječ *extends*, npr.

```
public class Beverage extends Item {
        ...
}
```

Redoslijed pozivanja konstruktora

- Prilikom stvaranja novog objekta klase *Beverage* prvo se mora inicijalizirati dio koji je naslijeđen iz bazne klase (tj. klase *Item*)
- Kada bi Item imao prazni konstruktor (kao u kôdu ispod) tada bi `new Beverage()` ispisao

„Ispis”:
```
Item constructor
Beverage constructor
```

```
public class Item  {
    public Item() {
       this(…some random sku…, "no name")
       System.out.println("Item constructor");
    }
...
}
public class Beverage extends Item {
    public Beverage() {
        // ovdje prevodilac dodaje super();
        System.out.println("Beverage constructor");
    }...
```

super za poziv konstruktora bazne klase (1)

- *super* poziva konstruktor bazne klase
  - treba razlikovati od *this* koji poziva neki drugi konstruktor iste klase
- prva naredba u konstruktoru izvedene klase je:
  - eksplicitno pozvani konstruktor iste (*this*) ili bazne klase (*super*)

  ili
  - super() koji je ugradio prevodilac (implicitno)
- ne napišemo li konstruktor za *Beverage*, tada će se automatski pokušati stvoriti podrazumijevani konstruktor u kojem bi implicitno bila dodana naredba super() za poziv praznog baznog konstruktora
  - takav u ovom slučaju ne postoji, pa će prevodilac prijaviti pogrešku

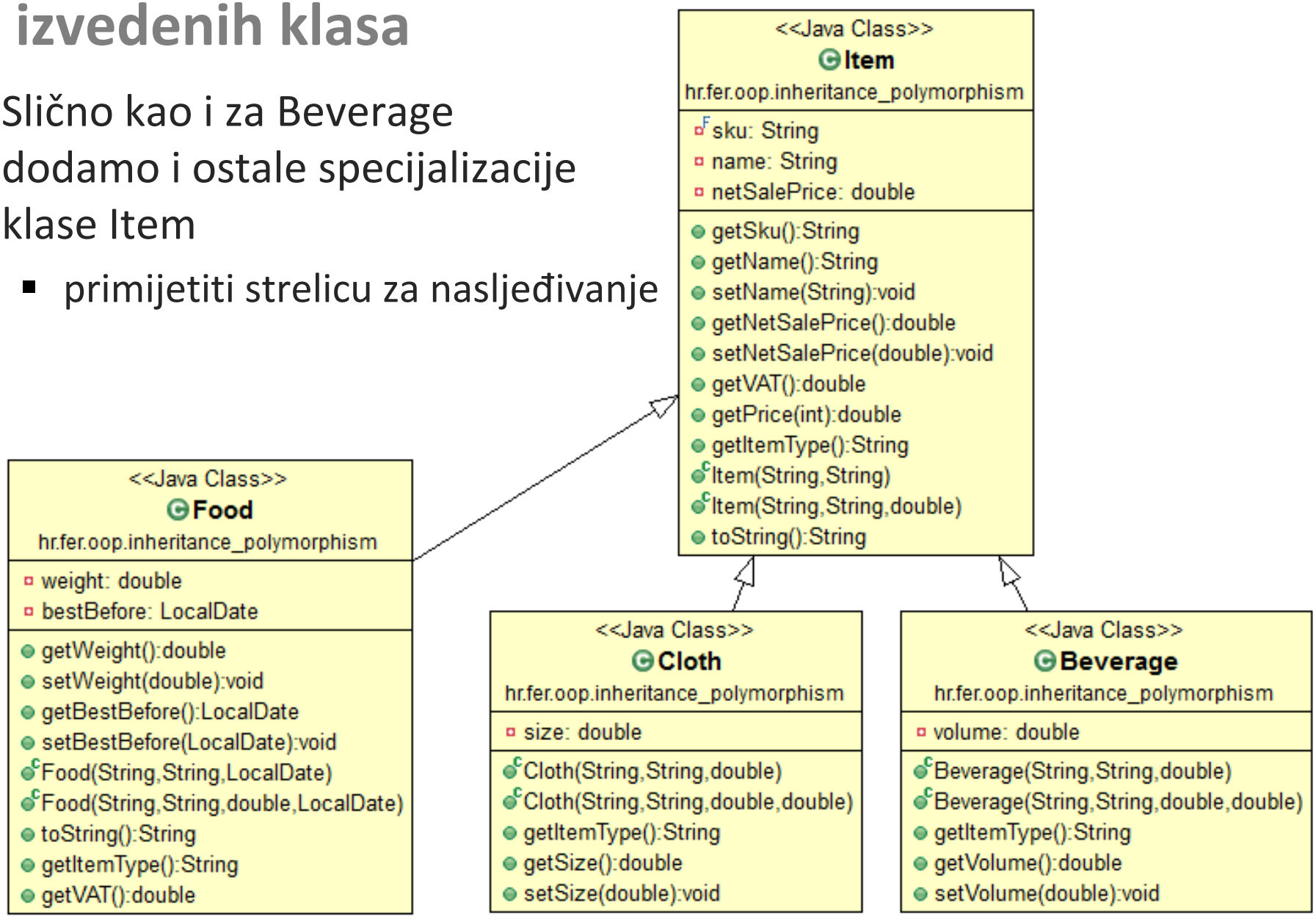
super za poziv konstruktora bazne klase (2)

- Što ako ne želimo ili ne možemo pozvati prazni konstruktor?

```
package hr.fer.oop.inheritance_polymorphism;
public class Beverage extends Item {
  private double volume;
  public Beverage(String sku, String name, double volume){
    super(sku, name);
    this.volume = volume;
  }
  public Beverage(String sku, String name, double price,
       double volume){
    super(sku, name, price);
    this.volume = volume;
  }
  @Override
  public String getItemType() {
    return "Beverage";
  }
```

Dijagram klasa s dodanim specifičnostima izvedenih klasa

- Slično kao i za Beverage dodamo i ostale specijalizacije klase Item
  - primijetiti strelicu za nasljeđivanje



Nadjačavanje metode (1)

- Nasljeđivanjem klase *Item*, klase *Food, Beverage, …* naslijedili su i metode *getPrice*, *getVAT*, *getItemType*, …
  - Svaka od klasa piše svoju verziju *getItemType* u kojoj ispisuje o kojoj vrsti artikla se radi.
  - PDV nije isti za hranu, pa je potrebno napisati ispravnu verziju.

- Metoda u izvedenoj klasi mora imati isto ime, argumente i povratni tip čime nadjačava (eng. *override*) metodu iz bazne klase.
- Uobičajeno se iznad metode izvedene klase koja nadjačava metodu bazne klase stavlja oznaka `@Override`.
  - Prevodilac će upozoriti ako takva metoda ne postoji u baznoj klasi.

Nadjačavanje metode (2)

- (Zamislimo da) hrana ima PDV u iznosu od 13% i za kategoriju vraća tekst Food.

```
public class Food extends Item {
	...
	@Override
	public String getItemType() {
		return "Food";
	}

	@Override
	public double getVAT() {
		return 0.13;
	}
```

super za poziv metode iz bazne klase

- Prethodne dvije metode su bile potpuno samostalne, ali ponekad je korisno samo malo modificirati naslijeđenu verziju.
  - Npr. u *toString* želimo još samo nadodati rok trajanja artikla
- *super* se može koristiti i za poziv metode (ili članske varijable) iz bazne klase
  - ne mora biti prva naredba u metodi

```
public class Food extends Item {
   @Override
   public String toString() {
         String s = super.toString();
         DateTimeFormatter formatter =
               DateTimeFormatter.ofPattern("dd.MM.yyyy.");
         s += ", best before: " + bestBefore.format(formatter);
         return s;
   } ...
```

Opaska vezana za nasljeđivanje i nadjačavanje

- Klase na nižim nivoima hijerarhijskog stabla mogu (ali ne moraju) definirati vlastite metode i nadjačavati postojeće
- Nasljeđuje se ona verziju koja je „zadnja” nadjačana
- *super* se odnosi na naslijeđenu metodu bez obzira gdje se nalazi
  - Nije moguće *super.super.method()* – bilo bi sintaksno neispravno i rušilo bi koncept nasljeđivanja

class A metode: m1, m2, m3

class B extends A

nadjačava m1 i definira m4

class C extends B

nadjačava m2 i definira m5

class D extends C

nasljeđuje i nadjačava m1 (iz klase B) – ne može pristupiti onoj iz A

naslijedila m2 (iz C), m3 (iz A) m4 (iz B), m5 (iz C)

Primjer korištenja naslijeđenih i nadjačanih metoda

…04_InheritancePolymorphism/…/inheritance_polymorphism/Main.java

```
Item item = new Item("1256", "Domaćica");
item.setNetSalePrice(10);
System.out.format("%s, price: %.2f, type: %s\n",
                  item, item.getPrice(1), item.getItemType());

Food food = new Food("777", "CaoCao", 2.5,
                     LocalDate.of(2016,5,11));
System.out.format("%s, price: %.2f, type: %s\n",
                  food, food.getPrice(1), food.getItemType());

Beverage beverage = new Beverage("23", "Coca cola", 10, 2);
System.out.format("%s, price: %.2f, type: %s\n",
                  beverage, beverage.getPrice(1),
                  beverage.getItemType());
```

```
1256 - Domaćica, price: 12,50, type:
777 - CaoCao, best before: 11.05.2016., price: 2,82, type: Food
23 - Coca cola, price: 12,50, type: Beverage
```

Nadjačavanje i preopterećivanje

- Prilikom nadjačavanja, za povratni tip se može upotrijebiti i neka izvedena klasa povratnog tipa metode u baznoj klasi – *covariant return type.*
  - Npr. ako metoda u nekoj baznoj klasi vraća *Item*, onda izvedena klasa može nadjačati tu metodu i vratiti npr. *Beverage.*
    - *Napomena: moguće u Javi, ali ne npr. u C#-u*
- Treba razlikovati nadjačavanje (eng. *overriding*) od preopterećenja (eng. *overloading*).
  - Pisanjem metode s drugačijim parametrima od onog u nadređenoj klasi ne nadjačava metodu iz bazne klase, već samo stvara novu metodu, jedinstvenu za izvedenu klasu
    - *Napomena: ne vrijedi npr. Za C++*

Zabrana nasljeđivanja i nadjačavanja

- Označavanjem metode s *final* onemogućava se njeno nadjačavanje
- Označavanjem klase s *final* onemogućuje se njeno nasljeđivanje
- Prevodilac prijavljuje pogrešku

Polimorfizam u programskim jezicima

- polimorfizam, višeobličje
- u drugim strukama
  - biologija - javljanje različitih oblika jedinki unutar jedne biljne ili životinjske vrste
  - kemija - odlika nekih spojeva ili elemenata da pri istom kemijskom sastavu kristaliziraju u kristalnim oblicima različite simetrije
  - izvor: Hrvatski jezični portal
- u računarstvu
  - *Stroustrup 2007: provision of a single interface to entities of different types*
  - *Cardelli, Wegner 1985: A polymorphic type is a type whose operations can also be applied to values of some other type, or types.*

Vrste polimorfizama

- implicitni polimorfizam
  - makro naredbe u C-u
  - dinamički tipovi podataka u nekim jezicima (npr. Lisp)
- ad-hoc polimorfizam
  - preopterećivanje funkcija
- parametarski polimorfizam
  - predlošci i generici
- hijerahijski polimorfizam
  - podtipovi i nasljeđivanje

Polimorfizam u Javi (i općenito OOP-u)

- Natklasa sadrži metode zajedničke svim izvedenim klasama u hijerarhiji, ostavljajući mogućnost da pojedina izvedena klasa nadjača metodu svojom specifičnom implementacijom.
  - Takve metode u izvedenoj klasi nazivamo virtualnim metodama.
  - Sve metode objekta (ne-statičke) u Javi su virtualne metode
- Osnovne tipove (prilikom pozivanja metode) može se bilo gdje zamijeniti izvedenim tipovima.
  - Npr. metodi koja prima objekt tipa *Item*, možemo proslijediti objekt tipa *Food* ili *Beverage*, jer su i hrana i piće artikli.
- JVM će pozvati metodu specifičnu za pojedini objekt, a ne za tip reference. Engleski termini koji se koriste za to su:
  - *virtual method invocation*
  - *dynamic method dispacth*
  - odluka se donosi u trenutku izvršavanja, a ne prilikom prevođenja

Primjer polimorfizma (1)

- I hrana i piće i odjeća su artikli, pa se mogu objediniti u isto polje.
- Takvo polje može biti argument neke funkcije

…04_InheritancePolymorphism/…/inheritance_polymorphism/Polymorphism.java

```
Item[] items = new Item[3];

items[0] = new Beverage("23", "Coca cola", 10, 2); //upcast…

items[1] = new Food("777", "CaoCao",
                    2.5, LocalDate.of(2016,5, 11));

items[2] = new Cloth("045", "Simple T-shirt", 350, 54);

calculatePrice(items);
```

Primjer polimorfizma (2)

- Metoda *calculatePrice* prima polje tipa *Item* u kojem su neki artikli tipa *Food*, neki *Beverage*, a neki *Cloth*.
  - pojedine klase imaju različit PDV (npr. hrana 13%)
  - svi su naslijedili (i nisu nadjačali) *getPrice* koja poziva metodu *getVAT*
- Koju/čiju metodu *getVAT* pozvati određuje se prilikom izvođenja na osnovu tipa objekta u polju, a ne na osnovu tipa reference.
  - dinamičko povezivanje objekta i metode koja se poziva

```
private static void calculatePrice(Item[] items) {
  double price = 0;
  for(Item item:items){
      System.out.format("%s, price: %.2f, type: %s\n",
              item, item.getPrice(1), item.getItemType());
      price += item.getPrice(1); ...
```

```
23 - Coca cola, price: 12.50, type: Beverage
777 - CaoCao, best before: 05.11.2016., price: 2.82, type: Food
045 - Simple T-shirt, price: 437.50, type: Cloth
Total price = 452.825
```

Downcast i mogući problemi

- Iako je prvi element polja objekt tipa *Beverage* ne može se koristiti metoda specifična za Beverage, jer je referenca tipa *Item*
  - **prevodilac** javlja grešku kod prevođenja

```
Item[] items = new Item[3];
items[0] =  new Beverage("23", "Juice", 10, 2);
System.out.println(items[0].getVolume()); //compile error
```

- Možemo izvesti *downcast* u *Beverage* …

```
System.out.println(((Beverage)items[0]).getVolume());
```

- … ali ako to napravimo za objekt koji to nije

```
System.out.println(((Beverage)items[2]).getVolume());
```

program će se srušiti uz poruku (preciznije izazvati iznimku)

```
Cloth cannot be cast to
hr.fer.oop.inheritance_polymorphism.Beverage
```

Nadjačavanje i modifikatori

- Vidljivost metode u izvedenoj klasi može biti jednaka ili veća od vidljivosti nadjačane metode
  - Npr. ako je u baznoj klasi metoda *getVAT* bila *protected* u izvedenoj metoda *getVAT* kojim se nadjačava *getVAT* iz bazne klase može biti *public*, ali ne može biti *private*.

- Ako je metoda u baznoj klasi članska (nestatička), tada i metoda u izvedenoj klasi mora biti članska. Ako je metoda u baznoj klasi statička, tada i metoda u izvedenoj klasi mora biti statička.
  - U protivnom prevodilac javlja pogrešku.

Nadjačavanje metoda i statičke metode

- Što ako bazna i izvedena klasa imaju statičku metodu istog imena?
  - Izvedena klasa skriva (engl. hide) metodu iz bazne klase.
  - Nema nadjačavanja, tj. ne dolazi do dinamičkog povezivanja, već se koristi statičko povezivanje.

…/04_Inheritance_Polymorphism/src/hr/fer/oop/hiding_overriding/Main.java

- Tablica prikazuje što se događa prilikom definiranja metode istog prototipa u baznoj i izvedenoj klasi.

	Članska metoda u baznoj klasi	Statička metoda u baznoj klasi
Članska metoda u izvedenoj klasi	Nadjačavanje	Pogreška prevodica
Statička metoda u izvedenoj klasi	Pogreška prevodica	Skrivanje