PREDMET

Osnove Java Programiranja

Čas 11-12

HERARHIJA

Oktobar 2011.

SADRŽAJ

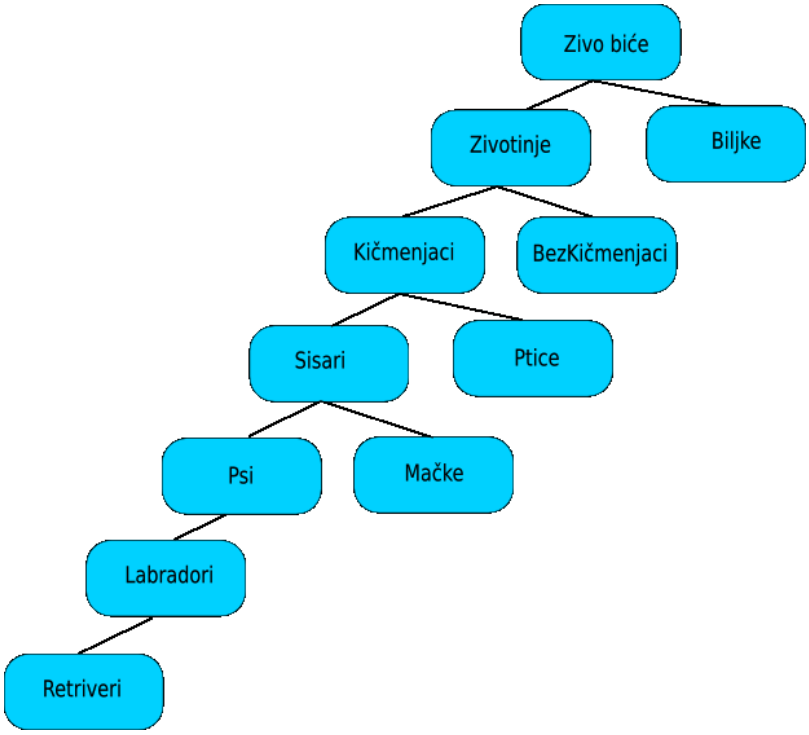
Vežbanje br. 10

Herarhija

Hiararhija – analogija sa prirodom

Na slici je prikazana hijerarhija živih bića onako kako je znamo iz biologije. U korenu cele hijerarhije nalazi se Živo biće. Živo biće delimo na životinje i biljke. Životinje možemo podeliti na kičmenjake i beskičmenjake. Kičmenjake na sisare i ptice. Sisare na pse, mačke,… Pse na labradore..., a retriveri su kao podgrupa labradora.

Ovim smo kreirali jednu hijerarhiju živih bića. Ako bismo želeli da predstavimo u kompjuteru u našem programu jednog psa, to možemo da uradimo tako što ćemo napraviti jednu klasu u koju ćemo da smestimo sve njegove osobine i sva njegova ponašanja. Međutim, mnogo je bolje kopirati prirodu, odnosno napraviti sistem kao što postoji u prirodi - podele određenih informacija, određenih ponašanja u određene kategorije. Tako mi možemo za opis našeg psa da iskoristimo hijerarhiju koja postoji u prirodi, koja nam je poznata iz biologije, i da njegove karakteristike podelimo u nekoliko klasa. Ovo je zgodan princip, jer ćemo da rasteretimo klasu psi; neće se sve nalaziti u njoj, već će osobine biti podeljene u više klasa, i svaka klasa će da odrađuje svoj deo posla. Ovo ima još jednu veliku prednost: u slučaju da želimo kasnije da napravimo novu klasu (recimo mačke), onda ne moramo da ponovo u toj klasi pišemo sve, već je dovoljno da nasledimo osobine sisara i samim tim da preuzmemo sve podatke, sve osobine i ponašanja koja sadrže sisari, kičmenjaci, životinje i živa bića (sve što se sastoji u hijerarhiji od te tačke na gore). Kada pišemo aplikacije, treba uvek prvo da razmišljamo o kreiranju adekvatne hijerarhije, koja će na najbolji način da predstavi ono što mi želimo. Kao i u analogiji sa prirodom, ni u našoj aplikaciji nisu sve klase izvršne, odnosno nisu sve klase klase od kojih se prave objekti. Kao što možemo da vidimo u ovom primeru sa živim bićima, ne možemo da napravimo instancu klase Sisar, zato što ne postoji objekat koji je samo sisar. On je ili pas ili mačka, a ima karakteristike sisara. To govori da je klasa Sisari abstraktna klasa, da ona nije striktno definisana, odnosno da ne može da se napravi objekat po njoj. Ona je abstraktna, i služi samo za građenje hijerarhije, odnosno za pripremanje podataka koji će kasnije imati u sebi neke konkretne klase. Sve klase iznad klase sisari (dakle kičmenjaci, životinje i živa bića) su takođe abstraktne klase, jer ne mogu da imaju svoje instance.



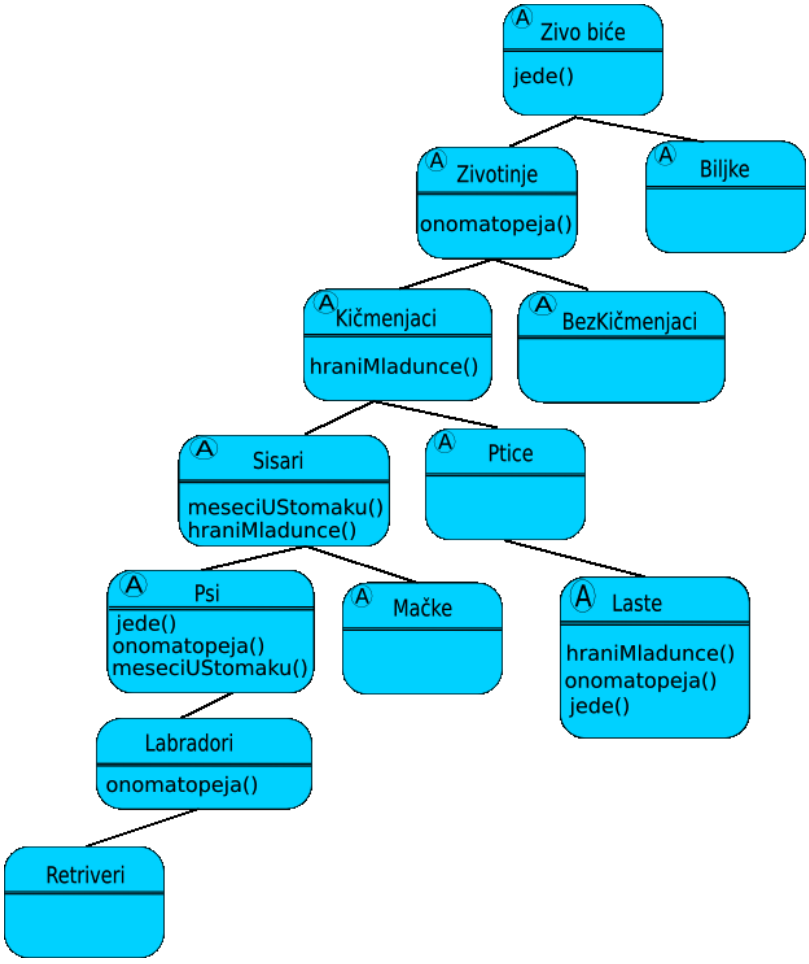
Hiararhija – sa funkcionalnošću

Na slici ispod predhodna hijerarija je proširena sa određenom funkcionalnošću, odnosno ubačene su metode. Uz to je stavljeno slovo a kao oznaka da je klasa abstraktna za sve klase koje ne mogu da naprave svoju instancu, i koje nam služe za kreiranje ove hijerarhije. Klase u donjem delu hierarhije, konkretne klase (klase Psi, Labradori, Retriveri, Mačke i Laste), su klase koje mogu da imaju svoje instance, dok su klase u gornjem delu hijerarhije (Sisari, Ptice, Beskičmenjaci, Kičmenjaci, Životinje, Biljke i Živa bića) abstraktne.

Prolazimo kroz hierarhiju od gore prema dole. Živo biće mora da jede. Definisali smo metodu jede u klasi Živo biće. U trenutku, pisanja klase Živo biće, se ne zna šta jede koja životinja. U klasi Živo biće ne može da se definiše šta psi jedu, šta mačke jedu, šta laste jedu,... odnosno ne može da se definiše ponašanje metode jede. Na ovom mestu može samo da se definiše metoda, može da se kaže da sva živa bića moraju da jedu, tako da pravimo jednu tzv abstraktnu metodu. Metoda jede je u ovom slučaju abstraktna, i ona ne može da ima svoju realizaciju. Isto se odnosi i na metodu onomatopeja, koja se nalazi u klasi Životinje. Znamo da se sve životinje na neki način oglašavaju, ali u klasi Životinje ne može da se definiše neko konkretno oglašavanje koje životinja ima jer se ne zna na koju životinju konkretno se misli. Zato i u slučaju klase Životinje, pravimo abstraktnu metodu onomatopeja. U klasi Kičmenjaci imamo takođe abstraktnu metodu hraniMladunče. Ova abstraktna metoda hraniMladunče služi da bise definisalo da svaki kičmenjak ima mladunče i da ga na neki način hrani, ali ovde se ne zna način na koji pojedinačne instance životinjato rade. Klasa Sisari, ima dve metode: MeseciUStomaku, definicija još jedne abstraktne metode pomoću koje se određuje koliko koji sisar provodi meseci u stomaku svoje majke pre nego što se rodi, i metoda hraniMladunče, koja u ovom slučaju nije abstraktna već implementirana. Zna se za sisare, da svi svoje mladunče hrane majčinim mlekom, tako da ovde može da se izvrši implementacija ove metode i pored toga što je klasa Sisari abstraktna klasa. Dakle, u klasi Sisari može da se uradi implementacija metode hraniMladunče, odnosno može da se ispiše pravo ponašanje metode hraniMladunče. Dakle, abstraktne klase, pored toga što mogu da definišu abstraktne metode za kasniju implementaciju, mogu i da implementiraju metode, odnosno mogu da imaju i konkretne implementacije. U slučaju sisara postoji konkretna implementacija hraniMladunče, tako da će sve klase koje se izvedu iz klase Sisari imati implementiranu metodu hraniMladunče koja će zaista vraćati pravilne informacije.

Klasa Psi je napravljena tako da nije abstraktna, ona je konkretna klasa, odnosno moguće je napraviti instancu klase Psi. Pošto se pravi konkretna instanca, mora da se implementiraju sve abstraktne metode koje su do sada postojale kroz hijerarhiju. Dakle, da se napiše implementacija metoda jede (treba da se definiše šta pas jede), mora da se napravi implementacija klase onomatopeja (treba da

se definiše kako se oglašava pas) i mora da se da implementacija metode meseciUStomaku (treba definisati koliko pas provodi meseci u majčinom stomaku). Ovde se vidii da ne mora da se implementira metoda hraniMladunče. Iako je metoda hraniMladunče abstraktna metoda iz klase Kičmenjaci, ne morama se implementirati u klasi Psi zbog toga što je ona već implementirana u klasi Sisari. Ovde se vidi kako se raspoređuju zadaci. Dakle, kroz hijerarhiju određene stvari se definišu na određenom nivou u hijerarhiji i ne moramo da u svakoj klasi definišemo sve, već možemo da posao podelimo na više klasa, kao što je to urađeno u klasi Sisari i Psi sa metodom hraniMladunče. Klasa Labradori, je takođe konkretna klasa i ona će naslediti sve osobine klase Psi. U klasi Labradori, se ne implementira ni jedna od abstraktnih gore definisanih metoda, osim metode onomatopeja. U čemu se metoda onomatopeja razlikuje od ostalih metoda? – Ne razlikuje se! Razlika je samo u logičkom ponašanju, odnosno želimo da labradori imaju drugačiju onomatopeju u odnosu na ostale pse. Zbog toga se vrši **overriding** metode onomatopeja u klasi Labradori, odnosno definiše se novo ponašanje metode onomatopeja. Kao što se vidi, klasa Labradori nasleđuje sve osobine klase Psi, pa zbog toga nije potrebno da se ponovo vrši implementacija metoda jede i meseciUStomaku. Klasa Laste, koja je nasledila klasu Ptice, je prva konkretna klasa u tom delu hijerarhije i samim tim mora da implementira sve metode koje su do tada bile definisane, a to su: jede, onomatopeja i hraniMladunče. Dakle ove tri metode koje su bile abstraktne u klasama Živo biće, Životinje i Kičmenjaci, moraju da se kreiraju kao konkretne metode u klasi Laste.



Primer Hiararhije – Klase ZivoBice

Klasa Živo biće, kao što se vidi na slici je abstraktna. To se definiše ključnom reči **abstract** koja stoji u definiciji same klase:

```
public abstract class ZivoBice
```

Sama klasa Živo biće ima jednu jedinu metodu i to abstraktnu:

```
public abstract void jede();
```

I u metodi i u klasi postoji reč **abstract**. Ako bar jedna metoda u okviru klasa ima ključnu reč **abstract** u sebi, odnosno ako klasa ima bar jednu abstraktnu metodu u sebi, to sa sobom povlači da i sama klasa mora da bude abstraktna. Nakon definicije metode jede stoji ; , odnosno metoda nema telo. Abstraktne metode nemaju telo. Telo metode služi za implementaciju, a već je rečeno da abstraktne metode nemaju implementaciju. Zbog toga se one samo definišu, dakle, imaju samo definiciju parametra koji vraća, parametra koji prima, modifikator vidljivosti i ključnu reč **abstract** – nema telo.

```
public abstract class ZivoBice
{
	public abstract void jede();
}
```

Primer Hiararhije – Klase Zivotinje

Kao što se vidi u hijerarhiji prikazanoj na početku lekcije sledeća klasa u hijerarhiji klasa Životinje. Klasa Životinje nasleđuje sve osobine klase Živo biće. To se u Javi piše pomoću reči **extends**, koja bi u nekom prevodu sa engleskog značila proširivanje:

```
public abstract class Zivotinje extends ZivoBice
```

Ključna reč extends, nam ukazuje na to da klasa Životinje nasleđuje klasu Živo biće. U objektno orijentisanim programskim jezicima ovo se zove nasleđivanje. Engleska reč **extends** dobro opisuje ono što klasa Životinje radi. Ona proširuje mogućnosti svoje nadklase (u ovom slučaju Živo biće). U telu klase se ne spominje abstraktna metoda jede koja se nalazi u klasi Živo biće i podrazumeva se da klasa Životinje u sebi poseduje i tu abstraktnu metodu. U okviru klase Životinje se samo definišeme nova abstraktna metoda onomatopeja.

```
public abstract class Zivotinje extends ZivoBice
{
	public abstract void onomatopeja();
}
```

Primer Hiararhije – Klase Kicmenjaci

Sledeća klasa u hijerarhiji je klasa Kičmanjaci. Klasa Kičmenjaci nasleđuje klasu Životinje, tako da opet pomoću ključne reči **extends** se proširujeme ponašanje klase Kičmenjaci osobinama klase Životinje i uvodi se nova abstraktna metoda koja se zove hraniMladunče.

```
public abstract class Kicmenjaci extends Zivotinje
{
	public abstract void hraniMladunce();
}
```

Primer Hiararhije – Klase Sisari

Sledeća klasa u hijerarhiji je klasa Sisari, koja je karakteristična po tome što ima konkretnu implementaciju jedne metode koja je bila abstraktna u jednoj od nadklasa. Kao i do sada, klasa Sisari nasleđuje klasu koja je iznad nje u hierarhiji u ovom slučaju klasa Kičmenjaci. Ovo se definiše ključnom reči **extends** u definiciji klase Sisari. Klasa Sisari definiše još jednu abstraktnu metodu meseciUStomaku ali, ima implementiranu metodu hraniMladunče. U datom primeru će poziv metode hraniMladunče da ispiše na ekranu “Mladunče hrani svojim mlekom”. Ovde vidimo konkretnu implementaciju konkretne metode hraniMladunče u abstraktnoj klasi. Metoda hraniMladunče nema reč **abstract** u sebi.

```
public abstract class Sisari extends Kicmenjaci
{
	public abstract void meseciUStomaku();
```

```
	public void hraniMladunce()
	{
		System.out.println("Mladunce hrani sa svojim mlekom");
	}
}
```

Primer Hiararhije – Klase Psi

Klasa Psi nam je prva klasa koja je konkretna (nije abstraktna), i to možete da se viditi i po deklaraciji klase:

```
public class Psi extends Sisari
```

Klasa Psi proširuje, odnosno nasleđuje klasu Sisari i to se vidi u njenoj definiciji. U klasi Psi mora da se implementiraju sve metode koje su do tada bile u hijerarhiji abstraktne. Metode koje moraju da se implementiraju u klasi Psi su: meseciUStomaku (implementacija abstraktne metode iz klase Sisari) i metoda onomatopeja.

Svi psi će da imaju onomatopeju Vau vau. Ovim je definisno ponašanje na poziv metode onomatopeja, koju će svi psi imati. Metoda iz abstraktne klase Živo biće, jede također mora ovde da se implementira. Dakle ispisuje se na ekran psi jedu meso. Kao što se vidi, nigde u klasi Psi nije izvršena implementaciju metode hraniMladunče; nije ni potrebno, jer je konkretna implementacija urađena u njenoj nadklasi Sisari.

```
public class Psi extends Sisari
{
	public void meseciUStomaku()
	{
		System.out.println("U stomaku majke provodi 4 meseca");
	}
	public void onomatopeja()
	{
		System.out.println("Vau vau");
	}
	public void jede()
	{
		System.out.println("Psi jedu Meso");
	}
}
```

Primer Hiararhije – Klase Labradori

Klasa Labradori nasleđuje klasu Psi, i jedino što je izmenjeno u odnosu na klasu Psi je promena ponašanja metode onomatopeja. Dakle, u klasi Labradori je definisano drugačije oglašavaju pasa koji su labradori. Oni će da kažu Vuf vuf vuuuf. **Override**-ovanje metoda iz nadklase se radi tako što damo isti potpis metode. Dakle, isti se tip vraća, isto se zove, isti broj parametara i tip parametra ima kao i metoda iz nadklase. Java 5 unosi novinu, tzv. anotation, pomoću koje možemo da olakšamo proveru da li smo zaista izvršili **override**. Pre metode kojom se želi da se izvrši **override** neke metodu iz nadklase, napiše se @Overreide i time se daje na znanje razvojnom okruženju da se **override**-uje metoda iz nadklase. Razvojno okruženje će da proveri da li u nekoj od nadklasa postoji metoda sa istim potpisom kao što je ova koja je napisana. Ako postoji, neće vratiti grešku; međutim, ako je kojim

slučajem načinjena greška u potpisu metode, odnosno dat je drugačiji naziv metode ili se razlikuje u nekom parametru, onda će razvojno okruženje da upozori da je ovo nova metoda i da ne gazi ni jednu metodu iz nadklasa. Pomoću ovih anotation-a se smanjuje mogućnost greške. Ovo je veoma važno kod velikih projekata, gde postoji puno **override** metoda.

```
public class Labradori extends Psi
{
	@Override
	public void onomatopeja()
	{
		System.out.println("Vuf vuf vuuuf");
	}
}
```

Primer Hiararhije – Klase Ptice

Klasu Ptice kao što se vidi, nasleđuje klasu Kičmenjaci i potpuno je prazna. Dakle, nema nikakvu metodu, ništa ovde nije implementirano, ostavljena je kao potpuno prazna. Ponekad ima smisla praviti prazne metode. Dakle, prilikom pravljenja hijerarhije može da se zna da na nekom mestu postoji posebno ponašanje, ali da u tom trenutku nije potrebna nikakva implementacija, odnosno nikakvo novo ponašanje na tom mestu, pa može da se kreira jedna prazna klasa u hijerarhiji koja će se koristiti prilikom kasnijeg unapređivanja programa (mesto gde može da se doda odgovarajuće ponašanje).

```
public abstract class Ptice extends Kicmenjaci
{
}
```

Primer Hiararhije – Klase Laste

Klasa Laste je konkretna klasa koja nasleđuje klasu Ptice i, samim tim što je prva konkretna klasa u tom delu hijerarhije, mora da implementira sve metode koje su do tada bile abstraktne. Dakle, metoda hraniMladunče od nadklase mora da se implementira, metoda onomatopeja iz nadklase mora da se implementira i metoda jede iz nadklase takođe mora da bude implementirana u klasi Laste.

```
public class Laste extends Ptice
{
	public void hraniMladunce()
	{
		System.out.println("Mladunce hrani sa sazvakanom ranom");
	}
	public void onomatopeja()
	{
		System.out.println("Piju piju");
	}
	public void jede()
	{
		System.out.println("Laste jedu bube");
	}
}
```

Pravljenje instanci klase

```
// Zivotinje zverko = new Zivotinje();
```

Dati primer pravljenja instance klase Zivotinja će da prijavi grešku kompajlera. Razlog prijavljivanja greške kompajlera je nemogućnost da kompajler napravi instancu abstraktne klase. S' obzirom da je rečeno da abstraktna klasa nema konkretnu implementaciju svojih metoda, onda ne može da se napravi konkretan objekat tog tipa jer on ne može da ima neko ponašanje. Iz tog razloga kompajler nije u mogućnosti da nam kreira instancu neke abstraktne klase. Dve kose crte na početku ovog reda su komentar, tako da se ova naredba, odnosno ovaj red, neće ni kompajlirati. Pravilno pravljenje instance je:

```
Psi lutalica = new Psi();
```

Ovde pomoću ključne reči new se kreira instanca klase Psi. Kreira se nova instanca – objekat klase Psi, i dat joj je naziv lutalica. Ako sme napravi hijerarhija kako treba, može da se napravi nova instanca klase Psi, ali da bude tretirana kao neka od abstraktnih nadklasa. To je prikazano u sledećem redu:

```
Zivotinje mezimac = new Psi();
```

Ovde je u promenljivoj mezimac smeštena konkretna klasa Psi, ali će u ostatku programa da se ova konkretna klasa tretira kao klasa tipa Životinja. Dakle, bez obzira što unutra postoji prava konkretna klasa, u programu se ona tretira kao klasa tipa Životinje. To znači da će biti dozvoljen pristup samo onim metodama koje su definisane u klasi Životinje ili u hijerarhiji iznad. Te metode ne moraju da budu implementirane u klasi Zivotinje da bi mogle da se koriste. Implementacija može da bude tek u klasi Psi, ali metode moraju da budu deklarisane, mora da se zna da one postoje, da bi moglo da im se pristupi. Sve metode koje se nalaze negde niže u hijerarhiji ispod životinja neće biti dostupne u ovoj instanci mezimac. Slična stvar je urađena i kod klase lastica. Kao što vidi, ovde su laste i psi stavljeni pod isti tip Životinje, tako da instance mezimac i lastica se tretiraju kao životinje, bez obzira što se ispod nalaze različite konkretne klase, odnosno različiti konkretni objekti.

```
Zivotinje lastica = new Laste();
```

Pozivanje metoda deteta

Ako se za instancu mezimac pozove metoda onomatopeja, dobiće se ispis na ekranu vau vau. Razlog ovog ponašanja je jednostavan: metoda onomatopeja je definisana u klasi Životinje, a instanci mezimac se trenunto pristupa kao tipu Zivotinje. To znači da i instanca lastica takođe ima odgovarajuću onomatopeju. Za razliku od metode onomatopeja koja je definisana u delu do klase Životinje, metoda meseciUStomaku je deklarisana tek u klasi Sisari koja se nalazi niže u hijerarhiji u odnosu na klasu Životinje. To znači da ne može da se pristupi toj metodi. Iako konkretan objekat koji se nalazi ispod instance mezimac ima u sebi metodu meseciUStomaku, ne može da se pristupi toj metodi jer je, u ovom trenutku, instanca mezimac definisana kao životinja. Ako probamo da pristupimo metodi meseciUStomaku, kompajler će da prijavi grešku, zato je taj kod stavljen pod komentar. Metoda jede je definisana u klasi Živo biće. To znači da i mezimac i lastica mogu da pozovu ovu metodu. Naravno, lastica ne može da pozove metodu hraniMladunče, kao što ni mezimac ne može da pozove metodu hraniMladunče, jer je metoda hraniMladunče definisana u kičmenjacima, a Kičmenjaci se nalaze niže u hijerarhiji u odnosu na klasu Životinje, kako se trenutno predstavljaju ove instance.

```
System.out.println("Pokretanje metoda od dece");

mezimac.onomatopeja();

lastica.onomatopeja();

// mezimac.meseciUStomaku();

mezimac.jede();

lastica.jede();

// lastica.hraniMladunce();
```

Kastovanje na niži nivo

Ako je potrebno da se instanci lastica pristupi i da se pozove njena metoda hraniMladunče, onda mora da se kastuje u neku od nižih klasa. Metoda hraniMladunče je definisana u klasi Kičmenjaci, pa treba instancu lastica da se prekastuje, odnosno prebaci na klasu Kičmenjaci. To se radi na sledeći način:

```
System.out.println("Posle kastovanja");
Kicmenjaci lastaNasa = (Kicmenjaci) lastica;
```

Kao što vidi, ne pravi se nov objekat (nije korišćena ključna reč new), već je starom objektu dato novo ime i, od tog trenutka pa na dalje, može da mu se pristupi kao kičmenjacima. Ova dva naziva lastaNaša i lastica su reference na isti konkretni objekat. To znači da će se sve izmene u instanci lastica automatski odraziti i na promenljivu lastaNaša, i obratno. Nakon ovog kastovanja, može da se pod nazivom lastaNaša pristupi metodi hraniMladunče, koja je definisana u klasi Kičmenjaci. Drugi način je kastovanje u letu. Dakle, moglo se ovoj metodi pristupiti i tako što bi instancu lastica kastovali u letu. Kastovanje u letu znači da se u istom redu izvrši kastovanje u okviru zagrade, pa se onda pristupi odgovarajućoj metodi. Dole je prikazano kastovanje u letu za lasticu i za promenljivu mezimac. U primeru je prikazano kastovanje promenljive mezimac u njenu originalnu klasu Psi. Nakon ovog, može da se pozovu sve metode koje klasa Psi ima

```
lastaNasa.hraniMladunce();
((Kicmenjaci)lastica).hraniMladunce();
((Psi)mezimac).meseciUStomaku();
```

Niz nad klase sa decom kao elementima

Jedna od korisnih mogućnosti prebacivanja objekata iz nižeg dela hijerarhije u objekte roditelje je i mogućnost pristupa različitim objektima preko iste instance. Napravljena je instanca niza životinja i unutra su ubačeni lutalica, mezimac i lastica. Sada je moguće proći kroz taj niz korišćenjem nove for petlje i pozvati metode onomatopeja i jede za svaku od tih instanci, bez obzira što one pripadaju različitim tipovima objekta. Ovo je jako zgodan mehanizam, koji se često koristi u objektno orijentisanom programiranju. For petlja će biti detaljnije objašnjena kada bude više reči o petljama, a ovde je prikazano kako može da se koristi niz našiLjubimci, pri čemu se pojedinačni element niza zove lokalno z, i on je tipa Životinje, pa će ova for petlja proći kroz sve elemente tog niza.

```
System.out.println("Zivotinje u nizu");
Zivotinje[] nasiLjubimci = new Zivotinje[3];
nasiLjubimci[0] = lutalica;
nasiLjubimci[1] = mezimac;
nasiLjubimci[2] = lastica;
for(Zivotinje z : nasiLjubimci)
{
      z.onomatopeja();
      z.jede();
}
```

Interface

Interface-i su u Javi izuzetno korisni imaju nekoliko svojih upotrebnih vrednosti. Razlika između Jave i C++-a je, između ostalog, i u načinima nasleđivanja. C++ ima tzv. višestruko nasleđivanje. Dakle, ako nam trebaju osobine dva različita objekta, mi to implementiramo tako što nasledimo oba objekta, i onda imamo kod sebe osobine (u tom novom objektu) i jednog i drugog objekta. U Javi to ne postoji. Java može da ima samo jednog roditelja, odnosno klasa može da nasledi samo jednu klasu, a osobine (neke druge koje su nam neophodne) možemo da dodamo pomoću interface-a. Interface-i u Javi, između ostalog, služe za napravljenje višestrukg nasleđivanja. **Interface**-i mogu da se koriste i kao šablon za pravljenje klasa. Dakle, može prvo da se napiše **interface** u okviru koga definišemo

metode koje neka klasa mora da ima, i onda svaka klasa koja implementira taj **interface** mora da izvrši implementaciju tih metoda. Tako se **interface**-i koriste kao šabloni. U realnom radu, u velikim timovima, **interface**-i su izuzetno korisni zato što pomoću njih može na brz i jednostavan način da se definiše ko šta treba da radi. Naime, jedan deo tima komunikaciju sa drugim delom tima može da ostvari preko odgovarajućeg dogovorenog **interface**-a, tako da ta dva dela tima mogu potpuno nezavisno da prave svoje klase i da izvršavaju konkretne implementacije, a jedino što je bitno je da imaju klase koje imaju implementiran dogovoreni **interface**, kako bi drugi deo tima mogao da im pristupi.

Interface se koristi za:

- Višestruko nasleđivanje
- Šablon za pravljenje klasa
- Dogovor u okviru tima

Primena Interface

Evo kako se primenjuje interface:

```
public interface Plivac
{
	public void pliva();
}
```

Osim što klasa Labradori proširuje klasu Psi, ona ujedno i implementira metode iz interface-a Plivač. Interface Plivač zahteva samo jednu metodu, a to je pliva. Interface-i se prave veoma slično abstraktnim klasama:

Kao kad deklarišemo abstraktne klase, ali ne pišemo ključnu reč abstract.

Implementacija interfejs-a

U primeru je data implementacija metode pliva, koja je deo interface-a Plivač, a koju klasa Labradori mora da implementira, jer su labradori poznati plivači.

```
public class Labradori extends Psi implements Plivac
{
	@Override
	public void onomatopeja()
	{
		System.out.println("Vuf vuf vuuuf");
	}

	public void pliva()
	{
		System.out.println("Pliva na kratko ume i da roni");
	}
}
```

Još jedan primer implementacije

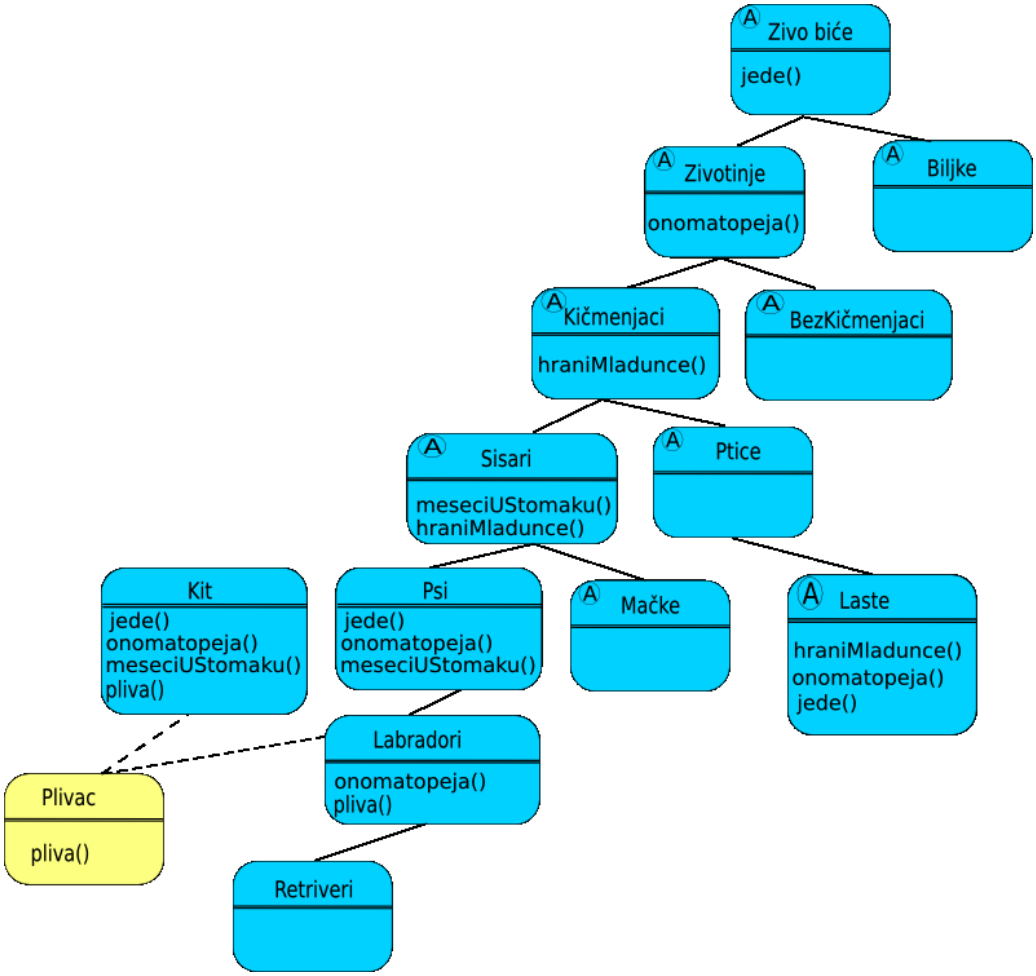
Još jedan primer implementacije **interface**-a. Klasa Kitovi proširuje klasu Sisari i implementira interface Plivač. S' obzirom da su kitovi prva konkretna klasa u hijerarhiji, oni moraju da implementiraju sve metode koje do tada nisu implementirane, a bile su definisane kao **abstract**. To znači: metoda

meseciUStomaku, metoda onomatopeja i metoda jede. Ovo su metode koje su bile definisane kao abstraktne u nadklasama (metoda jede je bila definisana kao abstraktna metoda u klasi Živa bića). Poslednja metoda u klasi Kitovi je pliva, koja je implementacija interface-a Plivač.

```
public class Kitovi extends Sisari implements Plivac
{
	public void meseciUStomaku()
	{
		System.out.println("U stomaku majke provodi 6 meseca");
	}
	public void onomatopeja()
	{
		System.out.println("Ne opisivo slovima");
	}
	public void jede()
	{
		System.out.println("Kitovi jedu planktone");
	}
	public void pliva()
	{
		System.out.println("Vešt plivač");
	}
}
```

Struktura klasa sa interfejsom

Na slici ispod je data strukturu klasa i primenjenog **interface**-a. Interface Plivač nije deo hijerarhije. On stoji nekako sa strane i samo utiče na određene klase, da moraju da implementiraju metode koje su definisane u **interface**-u.


Zivo biće
jede()
Zivotinje
onomatopeja()
Biljke
Kičmenjaci
hraniMladunce()
BezKičmenjaci
Sisari
meseciUStomaku()
hraniMladunce()
Ptice
Kit
jede()
onomatopeja()
meseciUStomaku()
pliva()
Psi
jede()
onomatopeja()
meseciUStomaku()
Mačke
Laste
hraniMladunce()
onomatopeja()
jede()
Plivac
pliva()
Labradori
onomatopeja()
pliva()
Retriveri