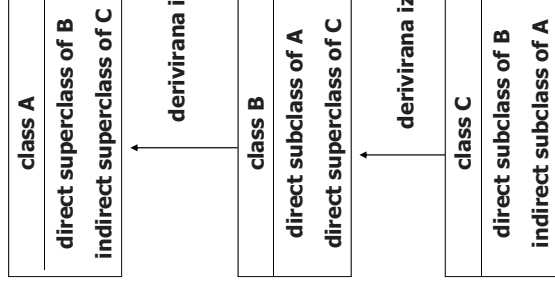


Java

Nadgradnja klase i nasljeđivanje - 1



Korištenje postojeće klase



Java © - Eugen Mudnić

Slide 3

```
class Dog
{
    // Članovi klase Dog
}
```

```
class Spaniel extends Dog
{
    // Članovi klase Spaniel
}
```

Korištenje postojećih klasa

- ❖ Riječnik:
 - ❖ **Derivacija** je definiranje nove klase na osnovu postojeće klase
 - ❖ Nova klasu tj. **deriviranu** klasu nazivamo direktnom **podklasom(subklasom)** klase iz koje je derivirana
 - ❖ Originalnu klasu nazivamo **nadklasom(superklasom)** derivirane klase

Slide 2

Java © - Eugen Mudnić

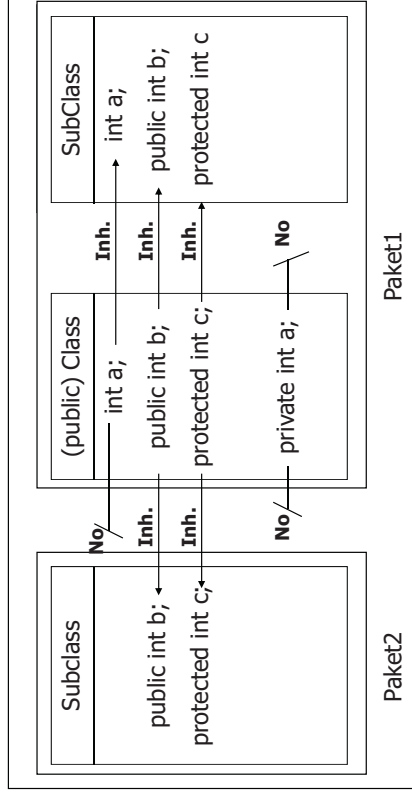
Nasljeđivanje

- ❖ objekt subklase (Spaniel) uvijek će unutar sebe sadržavati kompletn objekt superklase klase (Dog)
- ❖ To ne znači da su svi članovi superklase dostupni metodama koje su specifične samo za subklasu !
- ❖ **nasljeđivanje** : uključivanje članova bazne klase u deriviranu klasu na način da su **dostupni** (acesible) u deriviranoj klasi
- ❖ **nasljeđeni član** bazne klase je onaj koji je dostupan u deriviranoj klasi

Slide 4

Java © - Eugen Mudnić

nasljeđivanje podatkovnih članova



Slide 5

Java © - Eugen Mudnić

Slide 7

Java © - Eugen Mudnić

Protected

❖ **Protected : onemogućava pristup iz drugog paketa ali ne ograničava nasljeđivanje !!!**

Pravila

- ❖ Unutar paketa svi članovi su dostupni i nasljeđuju se osim članova označenih s **private**
- ❖ Klase van paketa mogu pristupiti članovima u drugom paketu ako je klasa kojoj pristupamo public i ako je član kojemu pristupamo označen s **public**
- ❖ Klase van paketa mogu naslijediti članove klase iz drugog paketa ako je klasa kojoj pristupamo označena s public i ako su članovi **public ili protected**

Slide 6

Java © - Eugen Mudnić

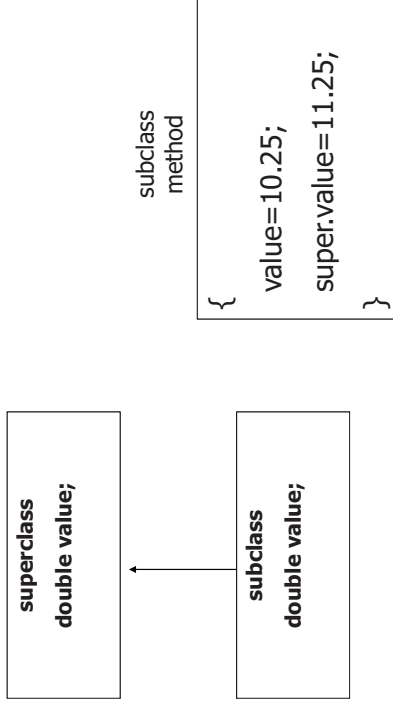
Slide 8

Java © - Eugen Mudnić

Skrivanje podatkovnih članova

- ❖ Možete definirati podatkovne članove (data member) s istim nazivom kao i član bazne klase
- ❖ Član bazne klase može i dalje biti naslijeđen, ali će biti skriven s članom derivirane klase istog naziva

Skrivanje podatkovnih članova



Slide 9

Java © - Eugen Mudnić

Nasljeđivanje metoda

- ❖ Konstruktori bazne klase se ne nasljeđuju, ali se mogu pozvati (za inicijalizaciju članova bazne klase)
- ❖ Štoviše, ako ne pozovete konstruktor bazne klase iz konstruktora derivirane klase, prevodilac će to učiniti umjesto vas

Slide 10

Java © - Eugen Mudnić

Deriviranje klase

```
public class Animal
{
    // Constructors for Dog object
    private String name;
    private String breed;
}

public String toString()
{
    return "This is a " + type;
}

private String type;
```

```
public class Dog extends Animal
{
    // Constructors for Dog object
    private String name;
    private String breed;
}
```

klasa Dog nasljeđuje samo toString() metodu

"Dog" or "Cat" or ...

Slide 11

Java © - Eugen Mudnić

Konstruktor derivirane klase

```
public class Dog extends Animal
{
    public Dog(String aName)
    {
        super("Dog"); // call the base constructor
        name=aName; // Supplied name
        breed="Unknown"; // Default breed value
    }

    public Dog(String aName,String aBreed)
    {
        super("Dog"); // call the base constructor
        name=aName; // Supplied name
        breed=aBreed; // Supplied breed
    }

    private String name;
    private String breed;
}
```

Trebali biste uvijek pozvati konstruktor bazne klase iz konstruktora derivirane klase

Ako ne pozovete konstruktor bazne klase iz klase prevodilac će umetnuti umjesto vas: super(); (međutim, je li ovaj konstruktor dio bazne klase ?)

Slide 12

Java © - Eugen Mudnić

Primjeri

❖Direktorij: Ch6\01_DerivedClass

- ❖[Animal.java](#)
- ❖[Dog.java](#)
- ❖[TestDerived.java](#)

Slide 13

Java © - Eugen Mudnić

Primjeri

❖Ch6\02_OverrideBaseMember

- ❖[Animal.java](#)
- ❖[Dog.java](#)
- ❖[TestDerived.java](#)

❖Ch6\03_CallingBaseMember

- ❖[Animal.java](#)
- ❖[Dog.java](#)
- ❖[TestDerived.java](#)

Slide 15

Java © - Eugen Mudnić

Overriding a Base Class Method

- ❖Možete definirati metodu u deriviranoj klasi koja ima istu signaturu kao i metoda bazne klase
- ❖atribut pristupa za metodu u deriviranoj klasi može biti **isti** kao u baznoj klasi ili **manje restriktivan**
- ❖Za objekt derivirane klase bit će pozvana metoda definirana u deriviranoj klasi, a ne metoda bazne klase – **overriding**

Slide 14

Java © - Eugen Mudnić

Odabir atributa bazne klase

- ❖Metode koje sačinjavaju vanjsko sučelje klase definiramo kao public
- ❖Normalno podatkovni članovi nisu public osim konstanti namijenjenih za opću upotrebu
- ❖Ako očekujete da će drugi ljudi koristiti vaše klase kao bazne klase tada podatkovne članove držite private, ali obezbijedite public accessor i mutator metode

Slide 16

Java © - Eugen Mudnić

Odabir atributa bazne klase

- ✧ Koristite protected kada želite neometan pristup od strane klase u istom paketu, ali za klase iz drugih paketa dozvoljen samo za subklase
- ✧ Ispuštanje atributa pristupa omogućava vidljivost člana klase u svim klasama paketa, dok je za klase van paketa to isto kao i upotreba private atributa

Slide 17

Java © - Eugen Mudnić

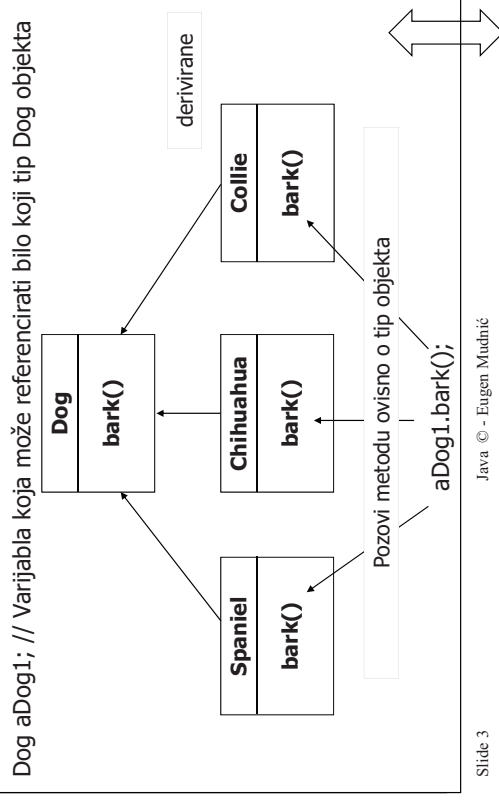
Polimorfizam je:

- ✧ mogućnost da s jednom varijablom određenog tipa referenciramo objekte različitih tipova i da automatski pozivamo metode koje su specifične za tip objekta na koji varijabla referencira (uh!)

Slide 2

Java © - Eugen Mudnić

Polimorfizam



Slide 3

Java © - Eugen Mudnić

Java

Nadgradnja klasa i nasljeđivanje - 2



Polimorfizam

```
Dog aDog1, aDog2, aDog3;  
aDog1 = new Dog();  
aDog2 = new Spaniel();  
aDog3 = new Collie();  
aDog1.bark();  
aDog2.bark();  
aDog3.bark();
```

- ✧ Možete koristiti varijable bazne klase kao referencu bilo koje klase koja je iz nje derivirana (vrlo zgodno!)

Slide 4

Java © - Eugen Mudnić

Polimorfizam

- ✧ Tip objekta na koji referencira varijabla nije poznat prije izvršavanja programa
- ✧ Polimorfizam se odnosi samo na metode – reference baznog tipa mogu se koristiti samo za pristup podatkovnim članovima baznog tipa

Slide 6

Java © - Eugen Mudnić

Polimorfizam - uvjeti

- ✧ Poziv metoda derivirane klase kroz varijablu bazne klase
- ✧ Pozvana metoda također mora biti član bazne klase
- ✧ Signatura metode i povratni tip moraju biti isti i u baznoj i deriviranoj klasi
- ✧ Atribut pristupa ne smije biti restriktivniji u deriviranoj klasi nego što je u baznoj klasi

Slide 5

Java © - Eugen Mudnić

Primjeri

- ✧ Ch6\04_Polymorphism
 - ❖ Animal.java
 - ❖ Cat.java
 - ❖ Dog.java
 - ❖ Duck.java
 - ❖ TryPolymorphism.java

Slide 7

Java © - Eugen Mudnić

Višestruki nivoi nasljeđivanja

```
class Spaniel extends Dog
{
    public Spaniel(String aName)
    {
        super(aName, "Spaniel");
    }
}
```

- ❖ Možete koristiti deriviranu klasu kao baznu klasu za nove subklase
- ❖ Ch6\05_MultiLevelInheritance
 - ❖ [Animal.java](#)
 - ❖ [Cat.java](#)
 - ❖ [Dog.java](#)
 - ❖ [Duck.java](#)
 - ❖ [Spaniel.java](#)
 - ❖ [TryPolymorphism.java](#)

Slide 8

Java © - Eugen Mudnić

Apstraktne klase

- ❖ Ne možete kreirati instancu apstraktne klase
- ❖ Možete deklarirati varijablu kao referencu na apstraktnu klasu
- ❖ public abstract type <-> abstract public type
- ❖ apstraktni metodi ne mogu biti private (jer je tada nemoguće nasljeđivanje)

Slide 10

Java © - Eugen Mudnić

Apstraktne klase

```
public abstract class Animal
{
    // Apstraktni metod
    public abstract void sound();

    public Animal(String aType)
    {
        type = new String(aType);
    }

    public String toString()
    {
        return "This is a " + type;
    }

    private String type;
}
```

- ❖ **Apstraktna klasa** je klasa čije jedna ili više metoda su deklarirane, ali ne i definirane
- ❖ Metode bez tijela – **abstraktne metode**

Slide 9

Java © - Eugen Mudnić

Apstraktne klase

```
Animal thePet; // Deklariraj varijablu tipa Animal (apstraktna klasa)
```

```
thePet = new Animal("Lion");
```

Error

```
thePet = new Dog("Doopsy");
thePet = new Spaniel("Spooty");
```

Slide 11

Java © - Eugen Mudnić

Apstraktne klase

- ❖ Ako derivirate subklasu iz apstraktne klase nije potrebno definirati sve apstraktne klase -> subklasa će biti apstraktna -> morate koristiti ključnu riječ **abstract**
- ❖ Prije ili kasnije potrebno je definirati subklasu koja ne sadrži apstraktne metode (tek tada možete kreirati objekte od iz klase)

Slide 12

Java © - Eugen Mudnić

Članovi naslijeđeni od klase Object

❖ Sedam public metoda

- ❖ **toString()** – vraća String objekt s opisom trenutnog objekta . U naslijeđenoj verziji ove metode to će biti naziv klase nakon kojeg slijedi znak '@' i heksadecimalna reprezentacija objekta (hash code). Ovaj metod se poziva automatski kada spajate (concatenate) objekte sa String varijablama korištenjem '+'. Moguće je napraviti override ove metode. Očito je da metodu toString() u deriviranoj klasi moramo uvijek označiti kao public
- ❖ **equals()** – uspoređuje objekt proslijeđen kao argument s trenutnim objektom. Ona vraća **true** ako se radi o istim objektima (ne samo iste vrijednosti)

Slide 14

Java © - Eugen Mudnić

Univerzalna supreklassa - Object

- ❖ Sve klase koje definirate su subklase – željeli vi to ili ne !!!
- ❖ Sve klase imaju standardnu klasu, **Object**, kao baznu klasu -> **Object** je superklasa svake klase
- ❖ Nikada ne trebate specificirati klasu **Object** kao baznu klasu vaše klase – ona je to automatski !
- ❖ **Object** je univerzalna superklasa !
- ❖ Varijable tipa Object može referencirati na objekt bilo koje klase !

Slide 13

Java © - Eugen Mudnić

Članovi naslijeđeni od klase Object

- ❖ **getClass()** – vraća objekt tipa **Class** koji identificira klasu trenutnog objekta
- ❖ **hashCode()** – vraća hash code vrijednost za objekt (koristi se za pospremanje objekata u hash tabele)
- ❖ **notify()** – budi niti asociirane s objektom
- ❖ **notifyAll()** – budi sve niti asociirane s objektom
- ❖ **wait()** – uzrokuje da trenutna nit čeka sve dok druga nit je ne pokrene

Slide 15

Java © - Eugen Mudnić

Članovi naslijeđeni od klase Object

- ✧ Dvije protected metode
 - ❖ `clone()` – kreira objekt koji je kopija trenutnog objekta. Ovo radi tzv. plitku kopiju objekta
 - ❖ `finalize()` – zove se za čišćenje kada se objekt uništava tj. kad garbage collector zaključuje da nema više referenci na objekt

Slide 16

Java © - Eugen Mudnić

Klasa Class

- ✧ Kada se program izvršava JVM generira instance klase **Class** za sve klase i sučelja (interfaces) u vašem programu
- ✧ klasa **Class** nema public konstruktor tako da ne možete kreirati vlastite **class** objekte
- ✧ Klasa Class definira više metoda ali se rijetko koriste
- ✧ metoda klase `Object getName()` vraća referencu na pripadni Class objekt

Slide 18

Java © - Eugen Mudnić

Određivanje tipa objekta

- ✧ Metoda `getClass()`, koju sve klase nasljeđuju od klase `Object`, vratit će objekt tipa `Class` koji identificira klasu kojoj objekt pripada

```
Animal pet= new Duck(...);
Class objectType = pet.getClass(); // Uzmi tip klase
System.out.println(objectType.getName()); // Ispiši naziv klase
System.out.println(pet.getClass().getName()); // sve u jednome
```

Izlaz je:

Duck
Duck

Slide 17

Java © - Eugen Mudnić

Klasa Class – primjer upotrebe

```
if(pet.getClass() == Duck.class)
    System.out.println("It is a duck !");
```

(upotreba `.class`)
Referenca na Class
objekt klase `Duck`

Slide 19

Java © - Eugen Mudnić

kopiranje objekata - Clone() metoda

- ❖ protected metoda clone() koji se nasljeđuje iz klase **Object** kreirat će objekt koji je kopija trenutnog objekta
- ❖ objekti koji se kloniraju moraju naznačiti da je kloniranje nad njima prihvatljivo – trebaju implementirati sučelje **Cloneable**
- ❖ clone() metoda klonira objekt kreiranjem objekta istog tipa te postavljanjem svih polja u kloniranom objektu na istu vrijednost kao i polja u kopiranom objektu

Slide 20

Java © - Eugen Mudnić

Clone poslije interfacea !

Slide 22

Java © - Eugen Mudnić

Clone() metoda

- ❖ Kada podatkovni članovi referiraju na objekte, referirani objekti se ne dupliciraju !

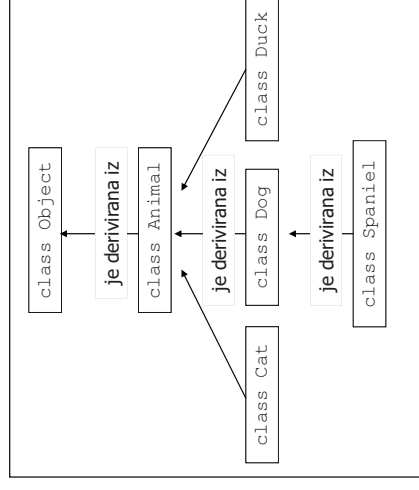
```
class Dog implements Cloneable
{
}
```

Slide 21

Java © - Eugen Mudnić

Casting objects

- ❖ Može se raditi cast objekta u bilo koju superklasu (prema gore)



Slide 23

Java © - Eugen Mudnić

Casting objects

```
Spaniel aPet = new Spaniel("Fang"); // Kreiraj objekt tipa Spaniel
```

Java zadržava sve informacije o klasi kojoj taj objekt pripada !!!

```
Animal theAnimal = (Animal)aPet;  
Animal theAnimal = aPet; // upward cast  
Dog aDog = (Dog)theAnimal; // downward cast
```

Slide 24

Java © - Eugen Mudnić

Kada - cast objects

❖Cast prema gore

- ❖ kada referencirate objekte s varijablom bazne klase
- ❖ prosljeđivanje referenci metodama koje prihvataju reference bazne klase

❖Cast prema dolje

- ❖ kada želite pozvati metodu specifičnu da deriviranu klasu

❖Primjer: Ch6\07_Casting

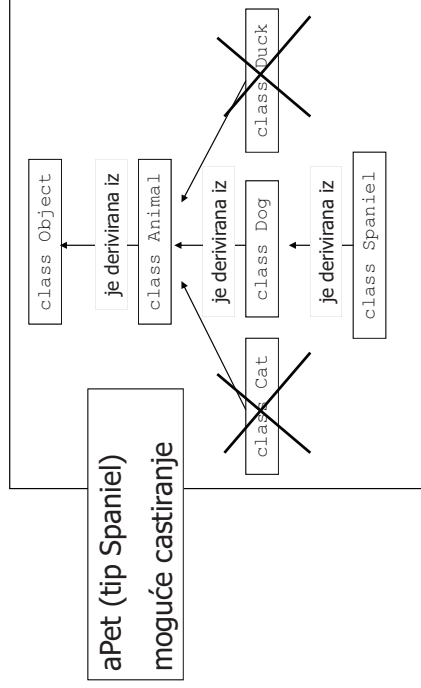
- ❖ Duck.java
- ❖ Animal.java
- ❖ LayEggs.java

Treba izbjegavati eksplicitni cast. Potencira mogućnost pogreške.

Slide 26

Java © - Eugen Mudnić

Casting objects



Slide 25

Java © - Eugen Mudnić

Identifikacija objekata

- ❖ operator instanceof vraća true ako je objekt referenciran lijevim operatorom istog tipa kao i desni operand ili je bilo koja subklasa.

Pretpostavimo da imate varijablu **pet** tipa Animal i da je želite kastirati u tip Duck.

Varijabla **pet** može biti referenca na bilo koju deriviranu klasu!

```
Duck aDuck; // Declare a duck  
if(pet instanceof Duck)  
{  
    ((Duck)pet).layEgg();  
}
```

Slide 27

Java © - Eugen Mudnić

Identifikacija objekata

pretpostavimo da **pet** pohranjuje referencu na objekt tipa

Spaniel.

Ako želimo pozvati metodu definiranu u klasi Dog moramo provjeriti da li pet stvarno referencira na objekt tipa Dog

```
if(pet instanceof Dog)
```

```
System.out.println("We have a dog!");
```

```
else
```

```
System.out.println("It's not a dog!");
```

```
if(pet.getClass() == Dog.class)
```

```
System.out.println("We have a dog!");
```

```
else
```

```
System.out.println("It's not a dog!");
```

Slide 28

Java © - Eugen Mudić

final modifikator

- ✧ Možete upotrijebiti ključnu riječ final za fiksiranje vrijednosti podatkovnog člana (već pokazano)
- ✧ metode deklarirane kao final : sprečava da subklasa izvede overriding iste metode
- ✧ klasa deklarirana kao final: sprečava da iz te klase deriviramo bilo koju klasu

```
public final void Add()  
{  
}  
}
```

```
public final class Line()  
{  
}  
}
```

Slide 30

Java © - Eugen Mudić

Dizajn klasa

- ✧ Polimorfizam je primarni razlog korištenja subklasa (i sučelja)
- ✧ To što imamo niz različitih objekata koje možemo tretirati odnosno upotrebljavati na isti način može značajno pojednostaviti pisanje programa

Slide 29

Java © - Eugen Mudić

Java

Nadgradnja klasa i nasljeđivanje – 3



Što je sučelje (Interface)

- ❖ Definicija: **interface** je imenovani skup definicija metoda (bez implementacije). Interface deklarira i konstante.
- ❖ Klasa koja **implementira interface** slaže se da će implementirati sve metode deklarirane u sučelju, tj. slaže se s određenim ponašanjem.

Slide 2

Java © - Eugen Mudnić

Interface

- ❖ Kako se razlikuje interface i apstraktne klase
 - ❖ Interface ne može implementirati nijednu metodu dok apstraktna klasa može.
 - ❖ Klasa može implementirati više sučelja, ali može imati samo jednu superklasu.
 - ❖ Interface nije dio hijerarhije klasa. Nevezane klase mogu implementirati isto sučelje.

Slide 4

Java © - Eugen Mudnić

Interface

Metode u sučelju su uvijek public i abstract – nije ih potrebno tako specificirati

```
public interface ConversionMethods
{
    double inchToMM(double inches);
    double ounceToGram(double ounces);
    double poundToGram(double pounds);
}
```

Konstante su uvijek public, static i final – nije ih potrebno tako specificirati

```
public interface ConversionFactors
{
    double INCH_TO_MM = 25.4;
    double OUNCE_TO_GRAM = 28.3;
    double POUND_TO_GRAM = 453.5;
}
```

Slide 3

Java © - Eugen Mudnić

Nadgradnja sučelja (Extending Interfaces)

- ❖ Možete definirati jedan interface na osnovu drugog

```
public interface Conversions extends ConversionFactors
{
    double inchToMM(double inches);
    double ounceToGram(double ounces);
    double poundToGram(double pounds);
}
```

```
public interface MyInterface extends HisInterface, HerInterface
{
    // Interface – članovi (konstante i apstraktne metode)
}
```

Slide 5

Java © - Eugen Mudnić

Implementacija sučelja

- ❖ Dodaj ključnu riječ, **implements**, koju slijedi naziv sučelja, nakon naziva klase + implementaciju svih metoda deklariranih u definiciji sučelja

```
public class MyClass implements Conversion
{
    // Implementation of the methods in the interface
    // Definition of the rest of the class ...
}
```

Slide 6

Java © - Eugen Mudnić

Implementacija sučelja koje definira metode

- ❖ Potrebno je implementirati metode koje su u sučelju
- ❖ Svaka klasa može imati različitu implementaciju metoda sučelja

```
public class MyClass implements Conversions
{
    public inchToMM(double inches)
    {
        return inches*INCH_TO_MM;
    }
    public double ounceToGram(double ounces)
    {
        return ounces*OUNCE_TO_GRAM;
    }
    public poundToGram(double pounds)
    {
        return pounds*POUND_TO_GRAM;
    }
    // Definition of the rest of class
}
```

Implementacija sučelja koje definira konstante

- ❖ Sve konstante koje su definirane u sučelju ConversionFactors su dostupne u klasi MyClass.

```
public class MyClass implements ConversionFactors
{
    double poundsWeight;
    public double getMetricWeight()
    {
        return poundsWeight*POUND_TO_GRAM;
    }
    // Definicija ostatka klase
}
```

Slide 7

Java © - Eugen Mudnić

Djelomična implementacija sučelja

```
public abstract class MyClass implements Conversions
{
    public inchToMM(double inches)
    {
        return inches*INCH_TO_MM;
    }
    public double ounceToGram(double ounces)
    {
        return ounces*OUNCE_TO_GRAM;
    }
}
```

Klasu je potrebno deklarirati kao apstraktnu ako izostavite implementaciju bar jedne metode sučelja.

Slide 9

Java © - Eugen Mudnić

Cloneable sučelje

- ❖ **Cloneable** sučelje je prazno sučelje ! Sve što je potrebno za njegovu implementaciju je navesti ga u deklaraciji klase
- ❖ Ono djeluje kao flag koji signalizira da dozvoljavate kloniranje svoje klase (marker interface)

```
public class MyClass implements Cloneable
{
    // Detail of the class ...
}
```

Slide

Slide 11

Java © - Eugen Mudnić

Primjer: korištenje sučelja

```
public interface PetOutput
{
    void sound();
}
```

- ❖ Sučelje PetOutput uključuje metodu koja je prije bila definirana u baznoj klasi Animal

Slide 12

Java © - Eugen Mudnić

Korištenje sučelja

- ❖ Za pakiranje konstanti
- ❖ za implementaciju polimorfizma bez potrebe za definiranjem subklasa
 - ❖ Možete separirati u sučelje skup metoda zajedničkih za više klasa . Sučelje onda može biti implementirano u svakoj od klasa.

```
public class Dog implements PetOutput
{
    public Dog(String aName)
    {
        name = aName; // Proslijeđeni naziv
        breed = "Unknown"; // Pretpostavljena vrsta
    }

    public Dog(String aName, String aBreed)
    {
        name = aName; // Proslijeđeni naziv
        breed = aBreed; // Proslijeđena vrsta
    }

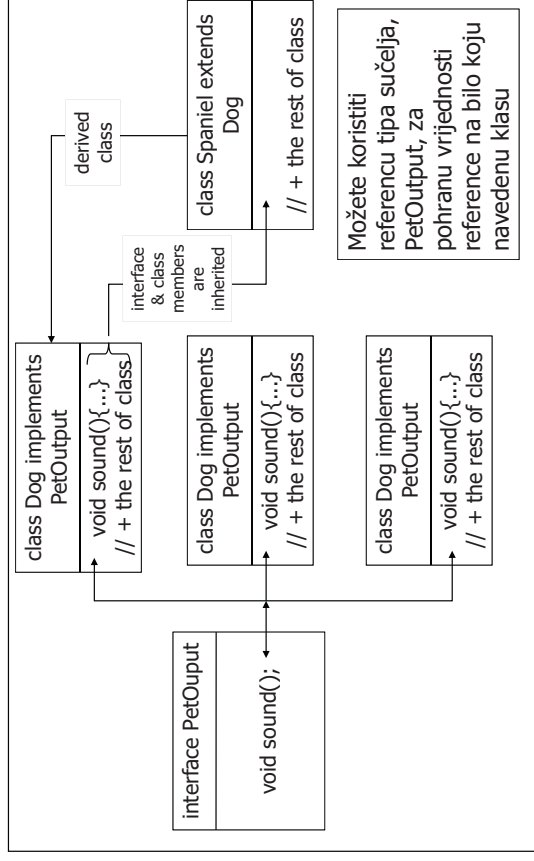
    // Prezentiraj detalje o psu kao string
    public String toString()
    {
        return "It's " + name + " the " + breed;
    }

    // Metoda za lajanje
    public void sound()
    {
        System.out.println("Vau Vau !");
    }
    ...
}
```

nema bazne klase, uklonjen super()

```
class Spaniel extends Dog
{
    public Spaniel(String aName)
    {
        super(aName, "Spaniel");
    }
}
```

Primjer: korištenje sučelja



Slide 16

Java © - Eugen Mudnić

Korištenje višestrukih sučelja

- ❖ U Javi derivirane klase mogu imati samo jednu baznu klasu, ali broj sučelja koje klasa može implementirati nije ograničen
- ❖ Možete koristiti jedno sučelje za konstante, a drugo za metode koje želite polimorfno koristiti

Primjeri: korištenje sučelja

- ❖ Ch6\10_Interface

- ❖ [Duck.java](#)
- ❖ [Cat.java](#)
- ❖ [Dog.java](#)
- ❖ [TestInterface.java](#)
- ❖ [Spaniel.java](#)
- ❖ [PetOutput.java](#)

Slide 15

Java © - Eugen Mudnić

Gniježdenje klasa u definiciji sučelja

- ❖ Možete staviti definiciju klase unutar definicije sučelja. Klasa će biti inner klasa sučelja, pretpostavljeno static i public.

```
interface Port
{
    // Methods & Constants declared in the interface
    class Info
    {
        // Definition of the class ...
    }
}
```

Klasa koja implementira sučelje nema direktnu vezu s inner klasom sučelja – ona treba implementirati metode definirane sučeljem. Može i koristiti objekte tipa inner klase

Slide 17

Java © - Eugen Mudnić

Anonimne klase

- ❖ Umjesto deklariranja lokalne klase s jednom Java klasom te instanciranja i korištenja u drugoj liniji, anonimna klasa kombinira dva koraka u jedan Java izraz

```
pickButton.addActionListener( new ActionListener()  
{  
    // Code to define the class  
    // that implements the ActionListener interface  
}  
);
```