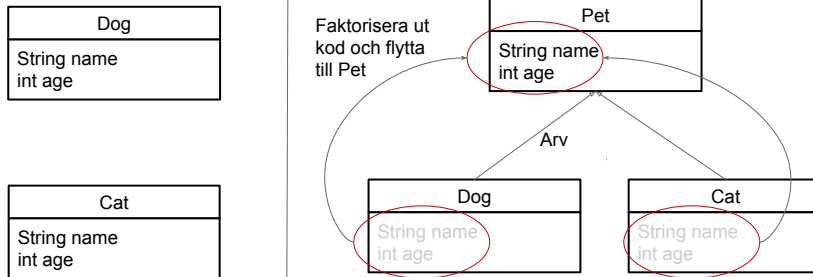


ex1inheritance

Joachim von Hacht

Redundant Kod



2

Ibland har flera klasser likadana instansvariabler och metoder.

- Samma kod på flera ställen. Dåligt. Redundant kod.
- Lösning: Skapa en klass för den gemensamma koden och flytta den dit.
- Låt därefter klasserna "ärva tillbaka" koden (på samma sätt som att alla klasser ärver Objects metoder).
- Aside: Pilarna med ofyllt huvud är UML notation för arv, läs nerifrån/upp. Dog och Cat ärver Pet.

Explicit Arv

```
public class Pet {  
    private String name;  
    private int age;  
    public String getName() {return name;}  
    public int getAge() { return age;}  
    ...  
}  
  
public class Dog extends Pet {  
    ...  
}  
  
public class Cat extends Pet {  
    ...  
}  
  
Dog d = new Dog(...);  
Cat c = new Cat(...);  
out.println(d.getName()); // Call inherited methods  
out.println(c.getAge());
```

3


Explicit arv

- Genom att ange extends vid klassdeklarationen kommer klassen **(subklassen)** att ärva från en annan angiven klass **(superklassen, basklassen)**
 - Instansvariabler och icke-private metoder ärvs.
 - Konstruktörer ärvs inte.
- Denna typ av arv kallas **implementationsarv** eftersom vi ärver körbar kod.
 - Till skillnad mot gränssnitt som benämns gränssnittsarv
- Vi använder denna typ av arv bl.a för att undvika redundant kod!

För att komma åt de ärvda privata instansvariablerna i subklassen måste vi använda setters/getters, vi kommer inte åt dem direkt.

super(...)

```
public class Pet {  
    private String name;  
    private int age;  
    public Pet(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
}  
  
public class Dog extends Pet {  
    public Dog(String name, int age) {  
        super(name, age);  
    }  
}  
  
Dog d = new Dog("Fido", 3);
```



4

Alla klasser har alltid en (osynlig) parameterlös konstruktor (default constructor).

- Använder vi bara den så behövs inget särskilt i samband med arv

Om superklassen har en konstruktor med parametrar måste subclassens konstruktor anropa denna först i sin konstruktor.

- För att initiera instansvariabler deklarerade i superklassen.
- **super(..)** syftar på den direkta superklassens konstruktor (jämför this(...)).
 - Kan ha implementationsarv i flera nivåer
 - Detta innebär att i en arvshierarki så initieras klasserna uppifrån ner (alltid superklasser först)

Standard för Konstruktorer

```
// Evil game character
public class Creep extends AbstractCreep {

    // Constructor, setting all values (set some default in super)
    public Creep(Path path, double width, double height) {
        super(path, width, height, 1, 30, 10, 10);
    }

    // Overloaded constructor
    public Creep(Path path) { // Remove some params to simplify
        this(path, 5, 5);    // Call another constructor with
                             // default values
    }
}
```

5

Ett vanligt mönster för konstruktorer är att:

- Ha en baskonstruktor som sätter alla värden, och om arv används, anropar super(...)
- Av bekvämlighetsskäl ha en eller flera andra konstruktorer.
 - Dessa anropar baskonstruktorn med förvalda värden.
 - De anropar inte super()

Subtyper vid Explicit Arv

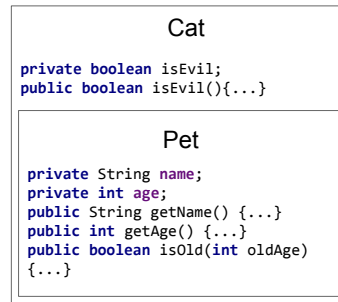
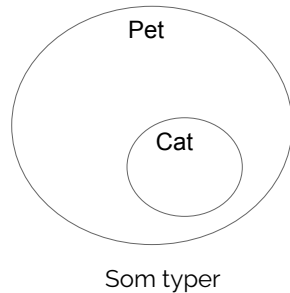
```
public final Dog extends Pet {  
    ...  
}  
  
Pet p = new Dog(); // super = sub  
Dog d = (Dog) p;   // sub = (sub) super
```

Om en klass B extends en annan klass A så gäller $B <: A$.

- Subtyp innebär att vi skall kunna ersätta super med sub, kommer alltid att fungera eftersom B ärver från A.
- B har minst lika många operationer som A.

Olika Perspektiv på Arv

Cat <: Pet



```
private boolean isEvil;  
public boolean isEvil(){...}
```

```
private String name;  
private int age;  
public String getName() {...}  
public int getAge() {...}  
public boolean isOld(int oldAge)  
{...}
```

Antag Cat är subclass till Pet (Cat <: Pet)

Utifrån typer är en subtyp en delmängd till supertypen.

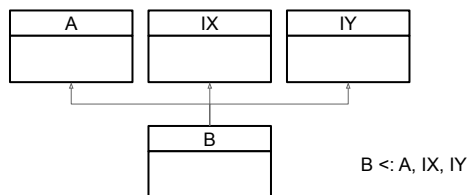
- Alla katter är Pet men inte tvärtom.
- Det finns fler operationer för subtypen.

Utifrån objekt finns ett delobjekt av typ Pet i objektet Cat

- Man kan anropa alla metoder från Pet på ett Cat objekt.
- OBS! Att bara ett objekt skapas vid new Cat()!
 - Inget super klass objekt skapas, det ingår som en del i subclassobjektet.

Multipelt Arv

```
public interface IX {  
}  
  
public interface IY {  
}  
  
public class A {  
}  
// Multiple interface inheritance and  
// single implementation allowed  
// (combination of allowed)  
// Multiple supertypes to B is A, IX, IY  
public class B extends A implements IX,  
IY {  
    // ....  
}
```



Java kan bara använda implementationsarv från en superklass

- Java har inte multipelt implementationsarv (som t.ex C++).
- Gränssnittsarv (implements) kan däremot var multipelt.

Summa: En typ kan ha multipla super eller subtyper.

Typinformation under Körning

```
Vehicle vehicle = new Vehicle();  
Car car = new Car();  
  
// Is car type Vehicle or subtype  
out.println(car instanceof Vehicle);  
  
// What's exact is the type of the object?  
out.println(vehicle.getClass() == Vehicle.class);
```

Man kan använda typinformation under exekvering

- Operatoren instanceof ger true om variabeln refererar ett objekt av angiven typ eller subtyp till angiven typ.
 - Om variabeln är null så false
- Metoden getClass() ger exakta typ för det refererade objektet under körning.

final för Klasser

```
public final class Dog extends Pet {  
    ...  
}
```

Klasser som är final kan man inte ärva från.

- En del av Java's klasser är final t.ex. String
- enum är alltid final.