Inheritance

Objektorientering - Arv

Arv - “Inheritance”

Varför?

- Uttrycka hierarkier av liknande klasser
- Återanvända kod
- Polymorfism - Behandla flera klasser som en

Vad?

- En klass kan sägas ärva en annan klass

Hur?

- I Java uttrycker man arv med nyckelordet `extends`

Objektorientering - repetition

- Abstraktion - förenklade modeller av objekt
- Inkapsling - egenskaper och beteende i en och samma enhet
- Arv - Hierarkier av relaterade klasser
- Polymorfism - En referens kan referera olika typer av (relaterade) objekt

Repetition - Abstraktion

- Dölj detaljer, beskriv det viktiga
- Vi behöver inte veta hur en miniräknare eller en bil fungerar internt för att använda den
- Om vi ska programmera ett kortspel, behöver vi bara fokusera på det väsentliga hos objekten. Ett kort har färg och valör (vi behöver inte bry oss om vikten dock)

Repetition - Inkapsling

- Inom objektorientering ser man objekt som det centrala begreppet
- Objekt har egenskaper och beteenden
- Inkapsling är att samma enhet rymmer såväl egenskaper som beteende
- Detta beskrivs i klasser -> av klasser skapar man objekt, klasserna beskriver objekten

Arv

- Låt säga att Chalmers har ett register med personer
- Personer är antingen anställda eller studenter (klasser: Employee, Student)
- Klasserna liknar varandra väldigt mycket
- Samma kod dupliceras - name, personalNumber, address, ...

Arv - kodåtervinning

- Systemet lagrar personer och kan skicka ut brev till dem
- Om vi har två typer (Student, Employee) så måste vi ju ha två av varje metod som behandlar en person, men där själva koden i metoderna är väldigt lika

```
public void mailStudent(Student s)   {...}
public void mailEmployee(Employee e) {...}
```

Arv - utbyggbarhet

Om vi lägger till ytterligare en typ av person i systemet blir det ännu värre. Vi måste då skriva om och lägga till saker...

```
public void mailStudent(Student s)   {...}
public void mailEmployee(Employee e) {...}
public void mailGuestLecturer(GuestLecturer gl){...}
```

En lösning kan vara arv

- Vi kan i stället vara ännu mer abstrakta och tänka, vad har dessa klasser gemensamt?
- De är personer (namn, personnummer, adress,,,)
- En möjlig lösning är då att skapa en ny klass och lägga gemensamma egenskaper och beteenden där

Före: två liknande klasser

```
public class Student{
  private String personalNumber;
  private String address;
  private String name;
  public Student(String p, String a, String n){
    this.personalNumber=p;
    this.address = a;
    this.name = n;
  }

  public int age(){
    // räkna ut och returnera ålder utifrån p-nr
  }
  public String address(){
    return address;
  }
}
```

```
public class Employee{
  private String personalNumber;
  private String address;
  private long salary;
  public Student(String p, String a, String n, long salary){
    this.personalNumber=p;
    this.address = a;
    this.name = n;
    this.salary = salary;
  }
  public long salary(){ // calculate and return salary}
  public int age(){
    // räkna ut och returnera ålder utifrån p-nr
  }
  public String address(){
    return address;
  }
}
```

Före: två liknande mail-metoder

```
public void mailStudent(Student s)   {...}
public void mailEmployee(Employee e) {...}
```

Möjlig lösning: arv via ny klass

```
public class Person{
  private String personalNumber;
  private String address;
  private String name;
  public Person(String personalNumber, String address, String name){
    this.personalNumber = personalNumber;
    this.address = address;
    this.name = name;
  }
  public int age(){ /* räkna ut och returnera ålder */ }
  public String address(){ return address; }
  public String name(){ return name;}
}
```

Båda är ju en Person

```
public class Student extends Person{
  public Student(String pNumber, String address, String name){
    super(pNumber, address, name); // Använd superklassens konstruktor
  }
  //... fler metoder...
}
public class Employee extends Person{
  private long salary;
  public Employee(String pNumber, String address, String name, long salary){
    super(pNumber, address, name);
    this.salary = salary;
  }
  public long salary(){ // räkna ut och returnera lönen }
}
```

mail-metoden är nu generell

```
public void mail(Person p){
   printAddress(p.address());
   ... etc
}
```

Att p kan referera till olika sorters Person kallas polymorfism. mail kan anropas med referenser till Student, Employee, GuestLecturer... om de ärver Person.

Vad är det man gör när man ärver?

Man utgår från det generella (Person) och skapar en specialisering (t ex Student).

Lite Java-syntaxregler

Keyword för arv är `extends`

`super(<eventuella args>)` anropar konstruktor i superklassen/basklassen

Ett anrop av `super()` måste ske på första raden i konstruktorn i den ärvande klassen.

Mer om `super()`

Alla klasser ärver Objekt. För att ett objekt skall intialiseras korrekt måste alltid superklassens konstruktor anropas. Gör vi inte det explicit, så lägger `javac` till ett anrop till `super()` åt oss, precis så, utan argument, först i konstruktorn.

Övning

Nu är det lämpligt att göra övningarna Inheritance 01, uppgifterna 1-3.