Arv - Inheritance

Del 1 - fortsättning

Har vi stött på arv förut?

- Faktum är att vi har använt arv hela tiden
- Alla klasser i Java ärver i något led Object
- Object är en abstraktion av objekt
- Alla klasser vi skrivit själva ärver direkt Object

`extends object` - implicit

```
public class Person{
  String name;
  // osv
}
```

Vi kan alltså lika gärna skriva:

```
public class Person extends Object{
  String name;
  // osv
}
```

Poängen med att allt ärver Object

Arv innebär att man får beteende (metoder) från basklassen (superklassen kallas det också) och så lägger man till beteende.

Ni kommer ihåg metoden toString()?

toString revisited

```
public class Person{
  private String name;
  private int age;
  public Person(String name, int age){
    this.name = name;
    this.age  = age;
  }
}
// i någon main-metod:
Person p = new Person("Kalle Anka", 68);
String s = p.toString(); // Hur är det möjligt?, Vad är resultatet?
```

`toString()` ärvs från Object

Alla klasser ärver ju Object. Vad för beteende (metoder) får de då?

Metoder i Object

Method Summary

Methods

Modifier and Type	Method and Description
protected Object	clone() Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
protected void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	getClass() Returns the runtime class of this Object.
int	hashCode() Returns a hash code value for the object.
void	notify() Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll() Wakes up all threads that are waiting on this object's monitor.
String	toString() Returns a string representation of the object.
void	wait() Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
void	wait(long timeout) Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
void	wait(long timeout, int nanos) Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Så, vad returneras av toString()?

Om vi använder den toString() som vi ärvt, får vi något som ser ut som en adress.

Det är förmodligen inte det vi vill ha, så vi har övat på att skriva egna toString()-varianter.

Att omdefiniera en ärvd metod kallas *overriding* i Java-litteraturen.

Vad händer om vi stavar fel?

```
public class Person{
  private String name;
  private int age;
  public Person(String name, int age){
    this.name = name;
    this.age  = age;
  }
  public String toStrong(){
    return "Name: " + name + " age: " + age;
  } // Kompilerar detta?
}
```

toStrong() fortstättning

```
// I en main-metod någon stans
Person p = new Person("Kalle Anka", 68);
String s = p.toString(); // Vad händer nu?
```

För att undvika detta finns något som kallas Annotations

@Override

```
public class Person{
  private String name;
  private int age;
  public Person(String name, int age){
    this.name = name;
    this.age  = age;
  }
  @Override
  public String toStrong(){
    return "Name: " + name + " age: " + age;
  } // Kompilerar detta?
}
// File: Arv/Arv02/src/override/Person.java
```

@Override hjälper oss

```
$ javac Person.java
Person.java:8: error: method does
not override or implement a method
from a supertype
  @Override
  ^
1 error
```

Typer och arv

Vi har sett att en Employee “är en Person”. I Java skrev vi `Employee extends Person.`

Men om det står så här i t ex en main-metod:

```
Person p = new Employee(pnr, addr, name, salary);
```

- Vilken typ har p?
- Kan man skriva p.salary(); //salary() är en metod i Employee

Typ vid kompilering och vid runtime

```
Person p = new Employee(p,a,n,s);
```

När vi kompilerar detta så kommer kompilatorn betrakta p som en Person, trots att p refererar till en Employee.

Det gör att vi inte får skriva:

```
p.salary(); // salary() finns bara i klassen Employee!
```

Oavsett om p refererar till ett objekt som är Employee, så väljer vi att betrakta p som en Person.

Metoden equals()

En metod vi ärver från `Object` är

```
public boolean equals(Object o)
```

Object är ju rotklassen, superklassernas superklass och har ingen egen superklass. Därför känner Object bara till andra Object och kan bara jämföra med dem.

Vi ärver equals som den är

Den version vi ärver tar alltså en referens till ett Object som argument.

Hur implementerar vi att två Person är equals?

Person och equals

En Person o är “equals” this om:

- `o != null`
- `o.getClass() == this.getClass()`
- `o.personalNumber == this.personalNumber`

Exempelimplementation av equals

```
@Override
public boolean equals(Object o){
  if (o==null || o.getClass() != this.getClass()){
    return false;
    // Can't be equal if you're null or different class!
  }else{
    return ((Person)o).personalNumber
             .equals(this.personalNumber);
  }
}
```

Kontroll av null

```
if (o==null || o.getClass() != this.getClass())
```

Kontroll av klasstillhörighet

Version 1:

```
if ( o.getClass() != this.getClass() ){...}
```

getClass() har vi ärvt från Object också!

Version 2:

```
if ( o instanceof Person ){...}
```

Type cast

```
((Person)o).personalNumber
```

Obegripligt?

Vi hade ju en Object-referens, o och vill veta personnumret.

Object som klass vet inget om personnummer! Så vi måste göra om o till en Person (nu när vi vet att det går!).

```
Person p = (Person)o; // Kanske lättare?
 return p.personalNumber.equals(this.personalNumber);
```

getClass versus instanceof

```
Person p   = new Person(a,b,c);
Employee e = new Employee(a,b,c,d);
//  p.getClass() != e.getClass()
//e instanceof Person == true
//p instanceof Person == true

// instanceof: av klassen Person
// eller av en subklass till Person
```