Klassen `Class`

Ett objekt av klassen `Class` innehåller information om en klass, t.ex. vilka variabler och metoder den har.

Hur får man ett `Class`-object?

- Metoden `getClass` för ett visst objekt:

```
Djur d = new Tiger();
Class c = d.getClass();
```

- Typliteral:

```
Class c = Tiger.class;
```

- Metoden `forName`, om klassnamnet inte är känt vid kompilering:

```
String s = ...;  // klassens fullständiga namn (inkl. paketnamn)
try {
   Class cl = Class.forName(s);
   ...
}
catch (ClassNotFoundException e) {
   ...
}
```

Jämförelser av objekts typer:

```
if (d.getClass() == d2.getClass())
   System.out.println("Samma sorts djur");
```

```
if (d.getClass() == Tiger.class)
   System.out.println("Är en Tiger");
```

```
if (d instanceof Tiger)
   System.out.println("Är en Tiger eller en subklass till Tiger");
```

Ett par metoder i klassen `Class`:

```
Class c = ...;

System.out.println("Klassen heter " + c.getName());


try {
   Object obj = c.newInstance();  // Skapar ett nytt
   ...
}
catch (Exception e) {
   ...
}
```

Klassen `Object`

- Roten i klasshierarkin.
- Ärvs (direkt eller indirekt) av alla klasser.

Viktiga metoder:

`getClass`	ger ett `Class`-objekt som beskriver objektets typ.
`toString`	ger en `String`-representation av ett objekt. Bör överskuggas i subklasser.
`equals`	i klassen `Object` jämför bara referenserna. Bör därför överskuggas i subklasser.
`hashCode`	ger ett värde av typen int. Används i hashtabeller. Måste överskuggas om man överskuggar `equals`.
`clone`	utför s.k. grund kopiering. Är **`protected`**. Måste överskuggas i subklasser.

- `x.equals(null)` skall ge resultatet **`false`**
- `x.equals(x)` skall ge resultatet **`true`**
- `x.equals(y)` skall ge resultatet **`true`** om och endast om `y.equals(x)` ger **`true`**
- om man inte ändrar något i `x` eller `y` så skall upprepade anrop av `x.equals(y)` ge samma resultat

Enkel modell för `equals`:

```
class MinKlass {
  private int i;          // enkel variabel
  private EnKlass r;      // referensvariabel
  @Override
  public boolean equals(Object obj) {
    if (obj == null || obj.getClass() != getClass())
      return false;
    else {
      MinKlass m = (MinKlass) obj;
      return r.equals(m.r) && i == m.i;
    }
  }
}
```

Enkel modell för `equals` vid arv:

```
class MinSubklass extends EnSuperklass {
  private int i;          // enkel variabel
  private EnKlass r;      // referensvariabel

  @Override
  public boolean equals(Object obj) {
    if (obj == null || obj.getClass() != getClass())
      return false;
    else {
      MinSubklass m = (MinSubklass) obj;
      return getVariabel().equals(m.getVariabel()) &&
             r.equals(m.r) && i == m.i;
    }
  }
}
```

där `getVariabel` är en metod som avläser en variabel i klassen `EnSuperklass`

Metoden `hashCode` ger i det ideala fallet ett unikt heltalsvärde för alla objekt som är lika.

Krav vid överskuggning av `hashCode`:

- Om och `o1.equals(o2)` ger **`true`** så skall `o1.hashCode()` == `o2.hashCode()`.

- Om `o1.equals(o2)` ger `false` så är det önskvärt, men inte nödvändigt, att `o1.hashCode()` och `o2.hashCode()` ger olika resultat.

- Om man anropar `hashCode` upprepade gånger för ett visst objekt och inte mellan anropen har ändrat något i objektet som påverkar metoden `equals`, så skall `hashCode` alltid ge samma resultat.

- `hashCode` i klassen `Object` skiljer på unika objekt => Man måste överskugga `hashCode` om två objekt som inte är samma objekt kan betraktas som lika.

Enkel modell för `hashCode` vid arv:

```
class MinSubklass extends EnSuperklass {
  private int i;          // enkel variabel
  private EnKlass r;      // referensvariabel

  @Override
  public int hashCode() {
    return getVariabel().hashCode() + r.hashCode() + i;
  }
}
```

där `getVariabel` är en metod som avläser en variabel i klassen `EnSuperklass`

Metoden `clone` skall skapa en kopia av ett existerande objekt.

```
public class MinKlass implements Cloneable {
  private int i;          // enkel variabel
  private EnKlass r;      // referensvariabel

  @Override
  public Object clone() throws CloneNotSupportedException {
     MinKlass kopia = (MinKlass) super.clone();   // ger grund kopia
     kopia.r = (EnKlass) kopia.r.clone();         // gör djup kopia
     return kopia;
  }
  ...
}
```