Class och Object

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.

Arv från Object

`toString` bör överskrivas (override).

Överlagring - Metoder som heter samma sak men har olika parametrar/typer.

Överskuggning - `@Override` – är när man ärver en metod och i en subklass gör en ny version av metoden. Den nya metoden ligger i en annan klass men har exakt samma parametrar och namn.

Equals

Skapa din egna.

- `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;
    }
  }
}
```

Equals vid arv

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`

hashCode

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`

Clone

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;
  }
  ...
}
```


Felhantering

Det finns tre typer av fel:

1. Kompileringsfel
2. Logiskt fel
3. Exekveringsfel

Kontrollera att indata är ett korrekt heltal

```
// Ett avsiktligt felaktigt program, version 1
import javax.swing.*;
public class DemoAvFel1 {
  public static void main (String[] arg) {
    String t = JOptionPane.showInputDialog("Ett tal?");
    for (int i=1; i<=t.length(); i++)
      if (t.charAt(i) >= '0' &&  <= '9')
        JOptionPane.showMessageDialog(null,"Talet är OK");
      else
        JOptionPane.showMessageDialog(null,"Inget tal");
  }
}
```

Går det att kompilera?

```
DemoAvFel1.java:7: error: illegal start of expression
      if (t.charAt(i) >= '0' &&  <= '9')
```

Rätta kompileringsfel.

Ändra i if-satsen:

```
// Ett avsiktligt felaktigt program, version 2
import javax.swing.*;
public class DemoAvFel2 {
  public static void main (String[] arg) {
    String t = JOptionPane.showInputDialog("Ett tal?");
    for (int i=1; i<=t.length(); i++)
      if (t.charAt(i) >= '0' &&  t.charAt(i) <= '9')
        JOptionPane.showMessageDialog(null,"Talet är OK");
      else
        JOptionPane.showMessageDialog(null,"Inget tal");
  }
}
```

Vad händer när man kör programmet?

Rätta logiska fel:

```
// Ett avsiktligt felaktigt program, version 3
import javax.swing.*;
public class DemoAvFel3 {
  public static void main (String[] arg) {
    String t = JOptionPane.showInputDialog("Ett tal?");
    boolean ok = true;    // korrekt, än så länge
    for (int i=1; i<=t.length(); i++)
      if (t.charAt(i) < '0' || t.charAt(i) > '9') {
        ok = false;       // inte längre korrekt
        break;
      }
    if (ok)
      JOptionPane.showMessageDialog(null,"Talet är OK");
    else
      JOptionPane.showMessageDialog(null,"Inget tal");
  }
}
```

Varför måste man krångla till det med variabeln OK?
Räcker det inte med att lägga en **break**-sats i **else**-delen av **if**-satsen i version 2 av programmet?

Kan man skriva version 3 av programmet utan att använda en **break**-sats?
Tips: Utöka villkors-delen i **for**-satsen.

```
    for (int i=1; i<=t.length() && ok; i++)
      if (t.charAt(i) < '0' || t.charAt(i) > '9')
        ok = false;       // inte längre korrekt
```

Vad händer när man provkör version 3 med ett korrekt tal som indata?

```
Exception in thread "main"
java.lang.StringIndexOutOfBoundsException:
       String index out of range: 3
       at java.lang.String.charAt(String.java:658)
       at DemoAvFel3.main(DemoAvFel3.java:8)
```

Rätta exekveringsfel.

Ändra till:

```
for (int i=0; i<t.length(); i++)
```


Exceptions

Det finns olika exceptions, de som man behöver fånga och de man inte behöver fånga. Fel man inte behöver fånga kan vara runtimeexceptions som index out of bounds, eller allvarliga fel i JavaRuntimeEnviroment.

Specificera exception och släng skiten.

Fånga exception och gör något med det.

Generera egna exceptions

Ta hand om exekveringsfel

Ex.

```
Thread.sleep(10000);   // vänta 10 sek
```

Kompileringsfel:

```
unreported exception java.lang.InterruptedException;
must be caught or declared to be thrown
```

Specificera exceptions

```
public class SignalDemo1 {
  public static void main (String[] arg)
                          throws InterruptedException {
    ...
    Thread.sleep(10000);  // vänta 10 sek
    ...
  }
}
```

Fånga exceptions

```
import javax.swing.*;
public class SignalDemo2 {
  public static void main (String[] arg) {
    ...
    try {
      Thread.sleep(10000);   // "farlig" sats
      // Hit kommer man om det inte blev fel
    }
    catch(InterruptedException e) {
      JOptionPane.showMessageDialog(null, "Det blev fel");
    }
    // Hit kommer man till slut i båda fallen
    ...
  }
}
```

Generera exceptions

```
 throw new ArithmeticException("negative area");

 throw new IllegalArgumentException("size < 0");
```

Egna exception-klasser:

```
public class CommunicationException extends Exception {
   public CommunicationException() {
     super();
   }
   public CommunicationException(String s) {
     super(s);
   }
 }

 throw new CommunicationException("Timeout in reader");
 throw new CommunicationException();
```

Java 8-bös

Try-with-resources

```
// Med resurshantering
public static void copyFile(String f1, String f2)
                                              throws IOException {
  try (
    BufferedReader in = new BufferedReader(new FileReader(f1));
    PrintWriter out = new PrintWriter(new BufferedWriter
                      (new FileWriter(f2)));
  ) {
    while (true) {
      String line = in.readLine();
      if (line == null)
        return;
      out.println(line);
    }
  }
}
```

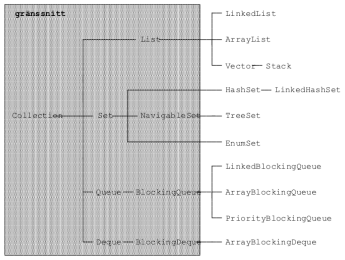
Krav:

Skall implementera gränssnittet `Auto-Closeable` vilket som enda metod har metoden `close`.

12

Containers

Containers är objekt vilka i sin tur innehåller referenser till andra objekt.



Skapa samlingar

```
Set<typ> h = new HashSet<>();
... // placera element i mängden h
List<typ> l = new LinkedList<>(h); // kopia av h

l.add(new String("en text"));
```

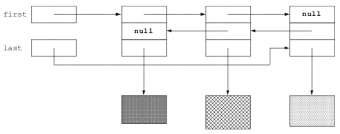
Klassen Collections

```
Integer imax = Collections.max(li);
Collections.fill(l, "Java");   // lägger "Java" i alla element i l

List<Integer> nollor = Collections.nCopies(100, 0);
Collections.copy(lTill, lFrån);

Collections.sort(l);
int k = Collections.binarySearch(ll, sökt);

Jämförare jfr = new Jämförare();
Collections.sort(l, jfr);
int k = Collections.binarySearch(l, sökt, jfr);
```

3

Man kan ha olika typer så länge de ingår i samma familj. Tänk på fordon som kan vara bilar, båtar, osv.

Implementering av listor



4