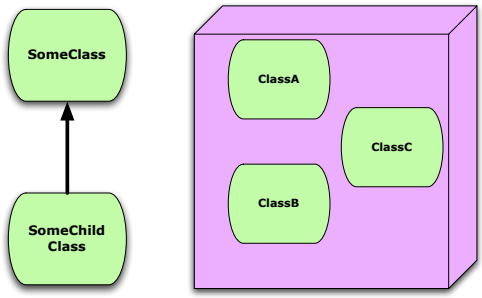
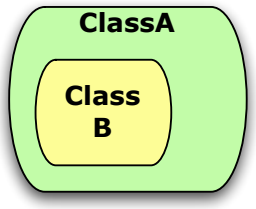
Nested & Inner Classes

I dagsläget kan vi ha följande samband mellan klasser, tillhör samma paket eller arv



Klasser inuti varandra



Nested Classes

- En klass definierad inuti en annan
- Precis som vilken annan klass som helst
- Synlighetsmodifierare
- Static


```
public class ShortTest
{
   public    int aVal;
   protected int bVal;
   private   int cVal;

   static public class  Inside
   {
      public void demo()
      {
         int a = aVal;
         a = bVal;
         c = cVal;
      }
   }
}
```



```
public class Demo
{
   public static void main( String[] args )
   {
      ShortTest.Inside s = new ShortTest.Inside();
   }
}
```

Nested classes är ju lite lika vanliga paket, dvs klasserna ligger ju "var för sig" men vad händer om jag vill ha en uppbyggnad där klasserna "ligger inuti" varandra?

Inner classes

- Objekten existerar "innuti" en annan klass
- Tre varianter
  - Member inner
  - Local inner
  - Anonymous inner

Varför ???

- Adaptor
- ... men trassla inte till saker i onödan


```
public class ShortTest {
    public    int aVal;
    protected int bVal;
    private   int cVal;

    public class  Inside {
        public void demo() {
            int a = aVal;
            a = bVal;
            a = cVal;
        }
    }
}
```

OK


```
public class Demo
{
    public static void main( String[] args )
    {
        ShortTest st = new ShortTest();

        ShortTest.Inside s = st.new Inside();
    }
}
```


Namn

- Två hierarkier
  - Inneslutning
  - Arv

```
class Outer {
  int x;
    class Inner
        extends Parent {

        void setX( int value) {
            x=value;
        }
    }
}
```

```
public class Parent {
   int x;
}
```

```
public class Demo {
   public static void main( String[] args ) {
      Outer o = new Outer();
      Outer.Inner i = o.new Inner();

      i.setX(50);

      System.out.println(o.x);
   }
}
```

```
> java Demo
0
```



```
class Outer {
  int x;
    class Inner
        extends Parent {

        void setX( int value) {
            this.x=value;
        }
    }
}
```

```
public class Parent {
   int x;
}
```

```
public class Demo {
   public static void main( String[] args ) {
      Outer o = new Outer();
      Outer.Inner i = o.new Inner();

      i.setX(50);

      System.out.println(o.x);
   }
}
```

```
> java Demo
0
```



```
class Outer
{
   int x;
   class Inner
      extends Parent
   {
      void setx( int value )
      {
         Outer.this.x = value;
      }
   }
}
```

```
public class Parent
{
   int x;
}
```

```
public class Demo
{
   public static void main( String[] args )
   {
      Outer o = new Outer();
      Outer.Inner i = o.new Inner();

      i.setx(50);

      System.out.println(o.x);
   }
}
```

```
> java Demo
50
```


Lokala klasser

```
public class Top
{
   class Minor
   {
      int val;
   }

   public void someMethod ()
   {
      class SomeClass
      {
         int x;
      }
   }
}
```


Anonyma klasser

- Har inget namn!!!
- "En-gångs" klasser
- Implementerar ofta ett enkelt interface


```
new {

    public void someMethod() {
        // bla bla
    }

}

new SomeExistingClassInterface() {

    public void someMethod() {
        // bla bla
    }

}
```

Ingen konstruktor

```
public Coordinates getCoord()
{
  final int methodVal = 0;

  return new Coordinates()
    {
      private int xVal;
      private int yVal;

      public int xCoordinate()
      {
        return xVal;
      }

      public int yCoordinate()
      {
        return yVal;
      }

      public String toString()
      {
        return "x = " + xVal + "  y = " +
          yVal + methodVal;
      }
    };
}
```



```
public Coordinates getCoord() {
    final int methodVal = 0;

    return new Coordinates() {
        private int xVal;
        private int yVal;

        public int xCoordinate() {
                  return xVal;
        }

        public int yCoordinate() {
            return yVal;
        }

        public String toString() {
            return "x = " + xVal + "  y = " + yVal +
                   methodVal;
        }
        {
            xVal = 100;
            yVal = 120;
        }
    };
}
```

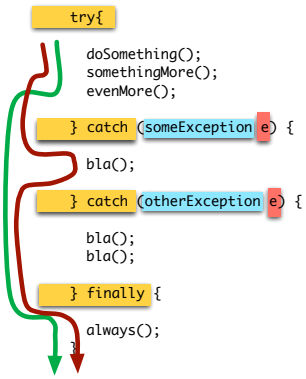

Tips

- Använd lokala klasser bara för små "enkla" klasser då koden annars blir väldigt svårläst

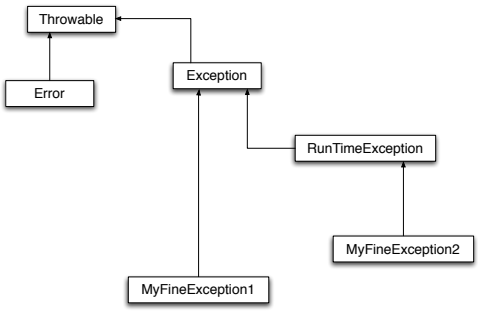
Lite mer Java

Exceptions

- För att hantera "oväntade" fel
- Snyggare kod (förhoppningsvis)
- Lätt att slarva



```
int doCalc( float xxx )
    throws YetAnotherException
{
    ....
    if ( xxxx ){
       throw new YetAnotherException();
    }
    ....
}
```


```
class MyFineException1
   extends Exception
{
}
```

```
class Demo
{
   public void fun( )
   {
      throw new MyFineException1();
   }
}

public class BigText
{
   public static void main( String argv[] )
   {
      Demo d = new Demo();

      d.fun();
   }
}
```



```
 > javac BigText.java
BigText.java:11: unreported exception MyFineException1; must be
                caught or declared to be thrown
                throw new MyFineException1();
                ^
1 error
```



```
class Demo
{
   public void fun( )
      throws MyFineException1
   {
      throw new MyFineException1();
   }
}

public class BigText
{
   public static void main( String argv[] )
      throws MyFineException1
   {
      Demo d = new Demo();

      d.fun();
   }
}
```



```
public class BigText
{
   public static void main( String argv[] )
   {
      Demo d = new Demo();

      try{
         d.fun();
      }
      catch (MyFineException1 x)
      {
         System.out.println( "Opps" );
      }
   }
}
```

```
class MyFineException2
   extends RuntimeException {

}

class Demo {
   public void fun( ) {
      throw new MyFineException2();
   }
}

public class BigText {
   public static void main( String argv[] ) {
      Demo d = new Demo();

      d.fun();
   }
}
```


Exempel på Error

- LinkageError
- VirtualMachineError
- CoderMalfunctionError
- AssertionError

Exempel på Exception

- IOException
- ParseException
- CertificateException
- BackingStoreException

Exempel på RuntimException

- IllegalArgumentException
- IndexOutOfBoundsException
- NullPointerException
- NoSuchElementException

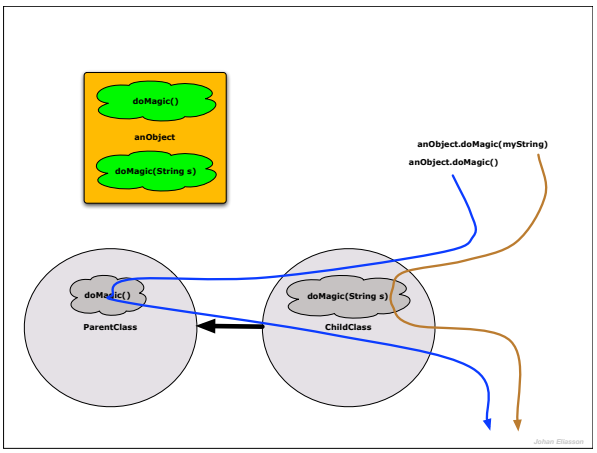
- Overloading
- Shadowing
- Overriding

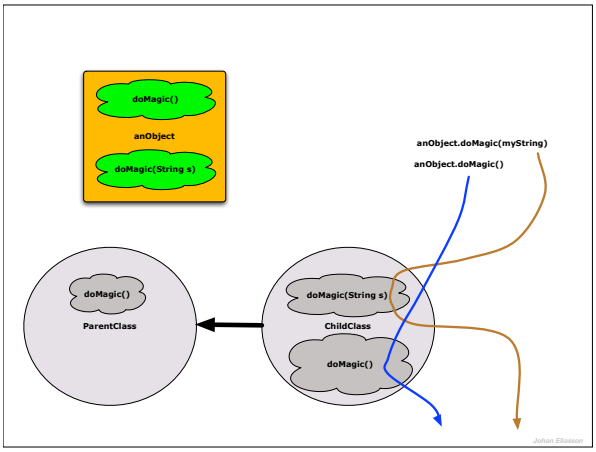
Overloading

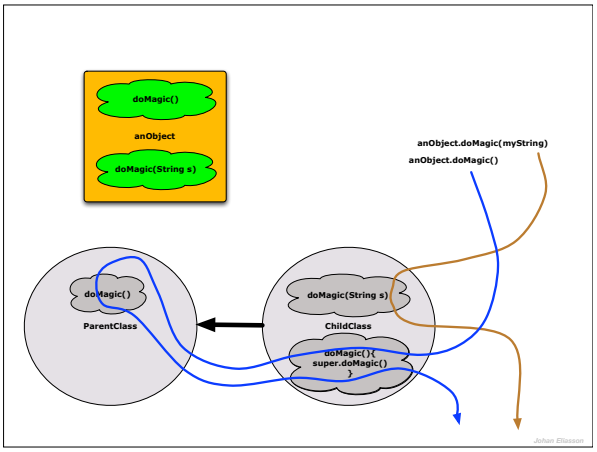
- En metod kan anta olika former
  - int create( ) { ... }
  - int create(String s) { ... }
  - int create(float f) { ... }
- Speciellt bra vid konstruktorer då vi där inte kan välja namn själva

Overriding

- En "barnklass" har en metod som "ersätter" förfaderns metod







Shadowing

- Gömmer en föräldraklass attribut
- Uppstår då vi deklarerar ett attribut med samma namn som ett som vi ärvt

Exempel på shadowing

```
class Parent {
    public int someField = 0;

    public int getSomeField() {
        return someField;
    }
}

class Child extends Parent {
    public int someField = 1;

    public int getSomeField() {
        return someField;
    }
}
```

```
public class Shadowing {
    public static void main( String argv[] ) {
        Child    c = new Child();
        Parent   p = c;

        System.out.println( "c.someField = " + c.someField );
        System.out.println( "p.someField = " + p.someField );
        System.out.println( "(((Child)p).someField = " + (((Child)p).someField );
        System.out.println( "((Parent)c).someField = " + ((Parent)c).someField );

        System.out.println( "c.getSomeField() = " + c.getSomeField() );
        System.out.println( "p.getSomeField() = " + p.getSomeField() );
        System.out.println( "(((Child)p).getSomeField() = " + (((Child)p).getSomeField() );
        System.out.println( "((Parent)c).getSomeField() = " + ((Parent)c).getSomeField() );
    }
}
```

```
c.someField = 1
p.someField = 0
((Child)p).someField = 1
((Parent)c).someField = 0
c.getSomeField() = 1
p.getSomeField() = 1
((Child)p).getSomeField() = 1
((Parent)c).getSomeField() = 1
```


Kodkonvensioner och god programmeringsstil

Bakgrund och syfte

- Gör rätt från början
  - *Mindre arbete för handledarna*
  - ***Mycket** mindre arbete för er*

Kodkonvention

- Ett standardiserat sätt att skriva kod
- Det finns *fler* standarder
- Ofta tvingas man hålla sig till en standard som gäller på företaget
- Var konsekvent!

- Tänk noggrant på *struktur* och *namngivning*!
  (ett skrämmande exempel)

```
if (lortgrisar.size() > 0) {
    RT.updateNejbor();
    RT.doChimpans();
    RT.doGorilla()
}
```


- Suns kodkonvention:
  http://java.sun.com/docs/codeconv/

- Exempel på kommentarer:

```
/*
 * Here is a block comment.
 */
if (condition) {

    /* Handle the condition. */
    ...
}
if (a == 2) {
    return true;        /* special case */
} else {
    return isPrime(a);  /* works only for odd a */
}
```


if-satser

```
if (condition) {
    statements;
}

if (condition) {
    statements;
} else {
    statements;
}

if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statements;
}
```

Undvik onödig kod

```
if (booleanExpression) {
    return true;
} else {
    return false;
}
```

- ska skrivas som

```
return booleanExpression;
```


Klassnamn, gränssnitt, variabler och konstanter

```
class MyClass { ... }
interface Interfacable { ... }
int variable = 4;
double getDouble () { ... }
static final int MY_CONSTANT = 3;
```


Dokumentation

- Viktigt
- Viktigt
- Viktigt
- javadoc

- // kommentar på en rad
- /* kommentar på flera rader */

- /** kommentar som javadoc förstår */
- @parameter namn förklaring
- @return förklaring


```
/**
 Vilket pris har en viss vara

 @param id  Den vara man vill
            veta priset på

 @return Priset på varan

 @throws myshop.ItemNotFoundException  Det fanns ingen
            vara med den givna identifikationen
*/
public float getPrice( String id )
    throws ItemNotFoundException
{
    // Leta reta på varan finns. Om varan inte finns så genereras ett
    // undantag och exekveringen av denna metod avslutas på en gång
    // samtidigt som felet skickas vidare uppåt i hierarkin.
    int pos = findStockItem(id);

    return storage[pos].price();
}
```