Klassenhierarchie (I)

```
class Point {

  double x, y;

  public void setX(double x) {
    this.x = x;
  }

  ...

  public double distance(Point pt) {
    ...
  }

}
```


Klassenhierarchie (II)

```
class Pixel extends Point {

  Color color;

  public void setColor(Color color) {
    this.color = color;
  }

  ...
}
```


Klassenhierarchie (III)

Unterklassen erben Daten und Verhalten (Methoden) ihrer Oberklasse.

```
Point pt = new Point();
Pixel pix = new Pixel();
double dist;

pt.setX(1.2);
pt.setY(-3.4);

pix.setX(5.6);
pix.setY(7.8);

dist = pix.distance(pt);
dist = pt.distance(pix);

pt = pix;
```


Klassenhierarchie (IV)

Java-Klassenhierarchie ist ein Baum.
Wurzel der Klassenhierarchie: `Object`.

```
class Point {
  ...
}
```

ist äquivalent zu

```
class Point extends Object {
  ...
}
```

super (I)

```
class Point {
  double x, y;

  Point(double x1, double y1) {
    x = x1;
    y = y1;
  }
  ...
}

class Pixel extends Point {
  Color color;

  Pixel(double x1,double y1,Color c) {
    super(x1, y1);
    color = c;
  }
  ...
}
```


super (II)

```
class Point {
  double x, y;

  public void clear() {
    x = 0;
    y = 0;
  }
  ...
}

class Pixel extends Point {
  Color color;

  public void clear() {
    super.clear();
    color = null;
  }
  ...
}
```


Casts

```
Point pt;
Pixel pix = new Pixel();

pix.setX(5.6);
pix.setY(7.8);
pix.setColor(someColor);

pt = pix;

pix = (Pixel)pt;

((Pixel)pt).setColor(otherColor);
```


Zugriffsrechte

`private`

Zugriff: nur aus der eigenen Klasse.

„default“, „package“, „friendly“

Zugriff: nur aus dem gleichen Package.

`protected`

Zugriff: nur aus dem gleichen Package *und aus Unterklassen*.

`public`

Zugriff: unbeschränkt.
Nur eine `public`-Klasse pro Source-Datei!
(⟨Dateiname⟩ = ⟨Klassenname⟩.java)

Abstrakte Oberklassen (I)

```
abstract class AbstractClass {

  abstract int abstractMethod();
  // Diese Methode wird hier *nur*
  // deklariert.
  // Unterklassen, die nicht selbst
  // abstract sind, *müssen* diese
  // Methode implementieren.

  int concreteMethod() {
     ...
  }
  ...

}
```


Abstrakte Oberklassen (II)

```
int test (AbstractClass a) {
  return a.abstractMethod();
}
// funktioniert


AbstractClass a;
a = new AbstractClass();
// funktioniert nicht!
```


Interfaces (I)

```
interface Colored {
  void setColor(Color color);
  Color getColor();
  // implizit: public.
}


class ColoredObj implements Colored {
  Color color;

  public void setColor(Color color) {
    this.color = color;
  }

  public Color getColor() {
    return color;
  }
}
```


Interfaces (II)

Interfaces erlauben mehrfache Vererbung:

```
class ColoredPoint
    extends Point implements Colored {

  ColoredObj coloredObj;

  public void setColor(Color color) {
    coloredObj.setColor(color);
  }

  public Color getColor() {
    return coloredObj.getColor();
  }
}
```

Strings

```
String str = "abc";

str = "ABC" + "DEF";

str += "GHI";

i = str.length();

ch = str.charAt(3);

System.out.println("abc" + 023);
```

String-Objekte sind unveränderlich.
Alternative: `StringBuffer`

Exceptions (I)

Fehlerbehandlung in Java geschieht mittels Exceptions (Ausnahmen).

```
class MyException extends Exception {
}

class MyClass {
  static public void A(int n)
      throws MyException {
    if (n < 0) {
      throw new MyException();
    }
  }
  ...
}
```


Exceptions (II)

```
class MyClass {
  ...

  static public void B(int n) {
    try {
      A(n);
      System.out.println("OK!");
    } catch (MyException myEx) {
      System.out.println("Huch!");
    } finally {
      System.out.println("Aufräumen!");
    }
  }

}
```


Beispiel

Siehe Arnold & Gosling,
Kap. 13.1 (`BitSet`),
Kap. 13.2/13.3 (`Enumeration`).