Kapitel 3. Haustiere und Fische – Java Klassen

Translation to German by Thomas Kunst.

Java Programme bestehen aus *Klassen*, die Objekte der wirklichen Welt repräsentieren. Obwohl die Leute unterschiedliche Vorstellungen davon haben wie man gute Programme schreibt, stimmen doch die meisten darin überein, dass man sie im sogenannten *objektorientierten* Stil schreiben sollte. Das bedeutet, dass gute Programmierer zunächst einmal entscheiden welche Objekte im Programm enthalten sein sollen, und welche Java Klassen diese Objekte repräsentieren werden. Erst dann beginnen sie Java Code zu schreiben.

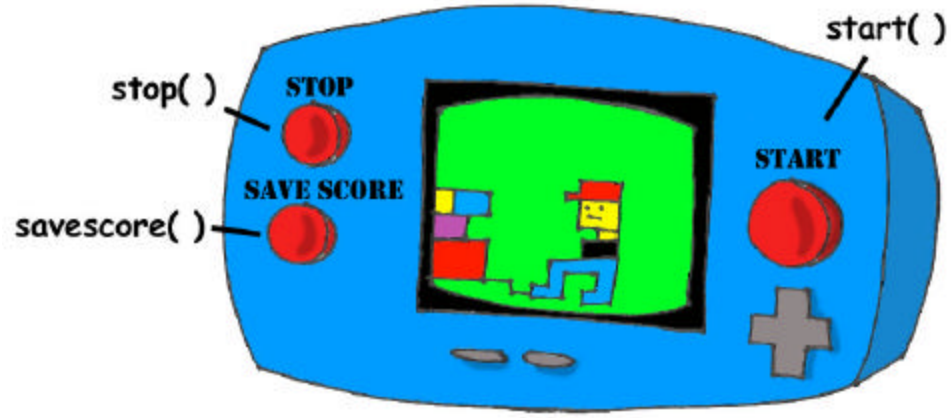
Klassen und Objekte

Klassen in Java können *Methoden* und *Attribute* haben.

Methoden definieren die Aktivitäten die eine Klasse ausführen kann.

Attribute beschreiben die Klasse.

Erzeugen wir eine Klasse namens `VideoGame` und besprechen sie. Die Klasse kann verschiedene Methoden haben, die aussagen *was Objekte dieser Klasse tun können*: das Spiel starten, stoppen, den Punktestand speichern (engl.: "save score"), und so weiter. Die Klasse kann auch ein paar Attribute oder Eigenschaften haben: Preis, Farbe des Bildschirms, Anzahl der Fernbedienungen und andere.



In der Sprache Java könnte diese Klasse so aussehen:

```
class VideoGame {
  String color;
  int price;

  void start () {
  }
  void stop () {
  }
  void saveScore(String playerName, int score) {
  }
}
```

Unsere Klasse `VideoGame` sollte anderen Klassen ähnlich sein, die ebenfalls Videospiele repräsentieren – sie alle haben Bildschirme verschiedener Grösse und Farbe, alle führen ähnliche Aktivitäten aus und alle kosten sie Geld.

Wir können konkreter werden und eine weitere Java Klasse namens `GameBoyAdvance` erzeugen. Sie gehört ebenfalls zur Familie der Videospiele, hat aber einige Eigenschaften die spezifisch für das Modell GameBoy Advance sind, zum Beispiel den Cartridge-Typ.

```
class GameBoyAdvance {
  String cartridgeType;
  int screenWidth;

  void startGame() {

  }
  void stopGame() {

  }
}
```

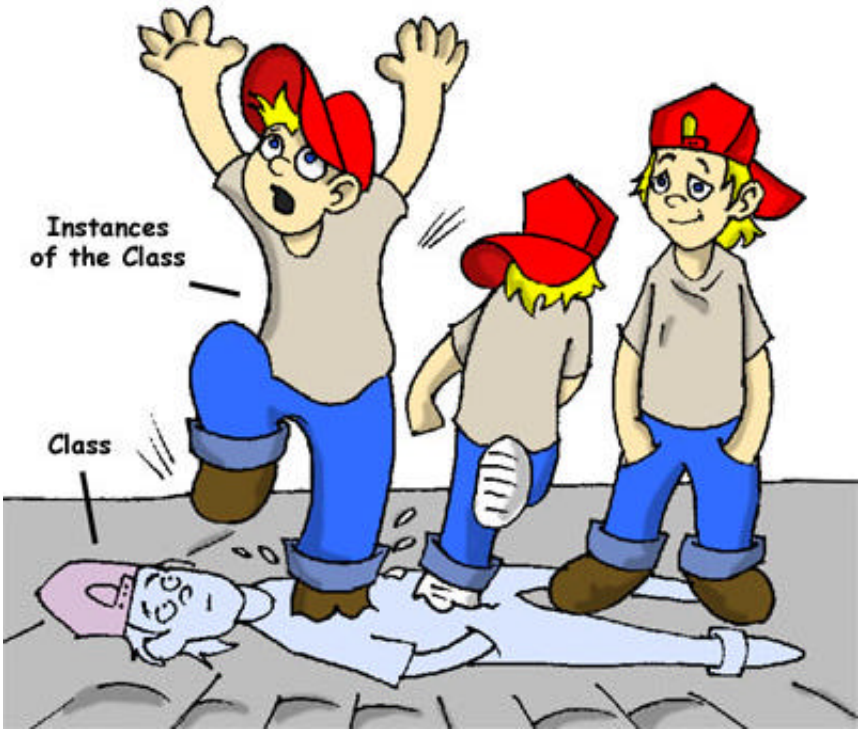
In diesem Beispiel definiert die Klasse `GameBoyAdvance` zwei Attribute – `cartridgeType` und `screenWidth` (“Breite des Bildschirms”) sowie zwei Methoden – `startGame()` und `stopGame()`. Aber diese Methoden können bisher noch keine Aktivitäten ausführen, denn sie haben noch keinen Java Code zwischen den geschweiften Klammern.

Ausser an das Wort *Klasse*, wirst du dich auch an eine neue Bedeutung des Wortes *Objekt* gewöhnen müssen.

Der Ausdruck “Eine Instanz eines Objektes erzeugen” oder “Ein Objekt instantiieren” bedeutet, ein solches Objekt im Speicher des Computers zu erzeugen, gemäss der Definition seiner Klasse.

Eine Baubeschreibung zum GameBoy Advance steht zur richtigen Spielkonsole in der gleichen Beziehung wie eine Java Klasse zu einer ihrer Instanzen im Speicher. Der Ablauf in der Spielefabrik beim Bau von Spielkonsolen gemäss dieser

Beschreibung ist vergleichbar mit dem Vorgang des Erzeugens von Instanzen von `GameBoyAdvance` Objekten in Java.



In vielen Fällen kann ein Programm eine Java Klasse erst dann verwenden, wenn eine Instanz von ihr erzeugt wurde. Die Hersteller produzieren ausserdem tausende von Spielkonsolen, alle gemäss der gleichen Beschreibung. Obwohl diese Konsolen alle die gleiche Klasse repräsentieren, können sie doch unterschiedliche *Werte* in ihren Attributen haben – einige sind blau, andere silbern, und so weiter. Mit anderen Worten, ein Programm kann *mehrere Instanzen* von `GameBoyAdvance` Objekten erzeugen.

Datentypen

Java *Variablen* repräsentieren Attribute einer Klasse, Argumente von Methoden oder sie können innerhalb einer Methode verwendet werden um kurzzeitig Daten zu speichern. Variablen müssen zunächst deklariert werden, dann erst kannst du sie verwenden.

Erinnerst du dich an Gleichungen wie y=x+2? In Java müsstest du die Variablen `x` and `y` mit einem numerischen *Datentyp* wie `integer` oder `double` deklarieren:

```
int x;
int y;
```

Die nächsten beiden Zeilen zeigen, wie du diesen Variablen einen *Wert* zuweisen kannst. Wenn dein Programm der Variablen x den Wert fünf zuweist, dann wird die Variable y den Wert sieben haben:

```
x=5;
y=x+2;
```

In Java kannst du den Wert einer Variablen auch auf ziemlich ungewöhnliche Weise verändern. Die folgenden beiden Zeilen ändern den Wert der Variablen y von fünf auf sechs:

```
int y=5;
y++;
```

Trotz der zwei Pluszeichen, wird Java den Wert der Variablen y nur um `eins` erhöhen.

Nach dem nächsten Codefragment ist der Wert der Variablen `myScore` (“meine Punktzahl”) ebenfalls sechs:

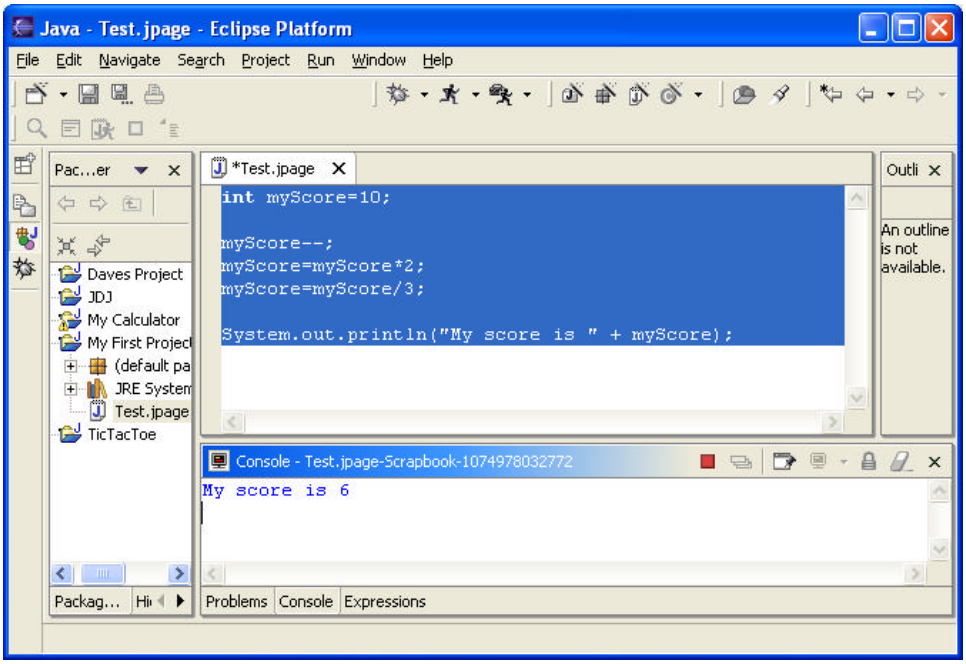
```
int myScore=5;
myScore=myScore+1;
```

Auf die gleiche Weise kannst du auch Multiplikation, Division und Subtraktion verwenden. Sieh'dir das folgende Stück Code an:

```
int myScore=10;

myScore--;
myScore=myScore*2;
myScore=myScore/3;

System.out.println("My score is " + myScore);
```

Was dieser Code wohl ausgeben wird? Eclipse hat eine coole Einrichtung namens *scrapbook* ("Schnipsel-Klebebuch") mit der man schnell mal eben einen Code-Schnipsel wie den obigen testen kann, ohne überhaupt eine Klasse zu erzeugen. Wähle im Menü *File*, *New*, *Scrapbook Page* und tippe `Test` als Name deiner scrapbook Datei.

Gib jetzt die fünf Zeilen, die verschiedene Dinge mit `myScore` tun, in's scrapbook ein, markiere sie, und klicke dann auf die kleine Lupe in der Werkzeugleiste.



Um das Resultat der Berechnung der Punktzahl zu sehen, klicke einfach auf die Registerkarte *Console* am unteren Ende des Bildschirms:

```
My score is 6
```

In diesem Beispiel wurde das Argument der Methode `println()` aus zwei Teilen zusammengeleimt – dem Text "My score is " und dem Wert der Variablen `myScore`, der sechs war. Einen `String` aus Stücken zusammenzusetzen, nennt man *concatenation* ("Verkettung"). Obwohl `myScore` eine Zahl ist, ist Java schlau genug, diese Variable in einen `String` umzuwandeln und ihn dann an den text *My Score is* anzuhängen.

Hier sind noch ein paar andere Möglichkeiten, den Wert von Variablen zu verändern:

`myScore=myScore*2;` ist das gleiche wie `myScore*=2;`
`myScore=myScore+2;` ist das gleiche wie `myScore+=2;`
`myScore=myScore-2;` ist das gleiche wie `myScore-=2;`
`myScore=myScore/2;` ist das gleiche wie `myScore/=2;`

Es gibt acht einfache, oder *primitive* Datentypen in Java, und du musst selbst entscheiden welche davon du verwendest, abhängig von Typ und Grösse der Daten die du in deinen Variablen speichern willst:

- Vier Datentypen, um ganzzahlige Werte zu speichern – `byte`, `short`, `int`, und `long`.
- Zwei Datentypen für Werte mit Dezimalkomma – `float` und `double`.
- Einen Datentyp, um einen einzelnen Buchstaben zu speichern – `char`.

- Einen *logischen* Datentyp, genannt `boolean`, der nur zwei Werte erlaubt: `true` (“wahr”) oder `false` (“falsch”).

Du kannst einer Variablen bei ihrer Deklaration einen Anfangswert zuweisen, und das nennt man *Initialisierung* der Variablen:

```
char grade = 'A';
int chairs = 12;
boolean playSound = false;
double nationalIncome = 23863494965745.78;
float gamePrice = 12.50f;
long totalCars =4637283648392l;
```

In den letzten beiden Zeilen steht `f` für float und `l` für long.

Wenn du Variablen nicht initialisierst, dann wird Java es an deiner Stelle tun, indem es jeder numerischen Variablen den Wert null zuweist, einer `boolean` Variablen den Wert `false`, und einem `char` den speziellen Code ‘\u0000’.

Es gibt auch noch ein spezielles Schlüsselwort `final` (“endgültig”), und wenn es in einer Variablendeklaration verwendet wird, dann kannst du dieser Variablen nur ein einziges Mal einen Wert zuweisen und dieser Wert kann danach nicht mehr geändert werden. In manchen Programmiersprachen nennt man `final` Variablen *Konstanten*. In Java bezeichnen wir `final` Variablen üblicherweise mit Namen in lauter Grossbuchstaben:

```
final String STATE_CAPITAL="Washington";
```

Ausser den primitiven Datentypen kannst du auch Java Klassen verwenden, um Variablen zu deklarieren. Jeder primitive Datentyp hat eine zugehörige *wrapper* Klasse (to wrap= “einwickeln”), zum Beispiel `Integer`, `Double`, `Boolean`, etc. Diese Klassen haben nützliche Methoden um Daten von einem Typ in einen anderen umzuwandeln.

Neben dem Datentyp `char`, der benutzt wird um einen einzelnen Buchstaben zu speichern, gibt es in Java die Klasse `String` um mit längeren Texten zu arbeiten, zum Beispiel:

```
String lastName="Smith";
```

In Java dürfen Namen von Variablen nicht mit einer Ziffer beginnen und keine Leerzeichen enthalten.

Ein *bit* ist das kleinste Stück Daten, das in einem Speicher abgelegt werden kann. Es kann entweder eine 1 eine 0 enthalten.

Ein `byte` besteht aus acht bits.

Ein `char` belegt in Java zwei Byte im Speicher.

Ein `int` und ein `float` brauchen in Java je vier Byte Speicherplatz.

Variablen der Typen `long` und `double` belegen je acht Byte.

Numerische Datentypen die mehr Byte belegen, können auch grössere Zahlenwerte speichern.

1 Kilobyte (KB) hat 1024 Byte

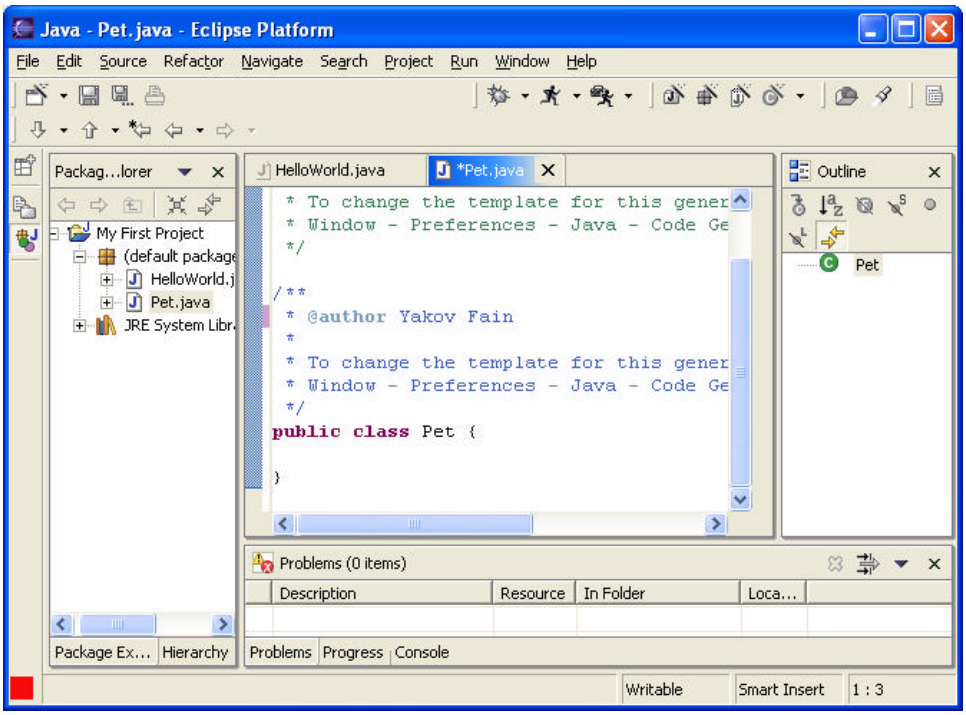
1 Megabyte (MB) hat 1024 Kilobyte

Ein Haustier wird erschaffen

Lass uns eine Klasse `Pet` (“Haustier”) entwerfen. Zunächst müssen wir entscheiden, welche Aktivitäten unser Haustier ausführen kann. Wie wär's mit essen (“eat”), schlafen (“sleep”) und sprechen (“say”)? Wir werden für diese Aktivitäten Methoden in der Klasse `Pet` programmieren. Ausserdem werden wir unserem Haustier die folgenden Attribute geben: Alter (“age”), Größe (“height”), Gewicht (“weight”), und Farbe (“color”).

Lege als erstes eine neue Java Klasse `Pet` in *My First Project* an, wie im Kapitel 2 beschrieben, aber markiere nicht das Kästchen für die Methode `main()`.

Der Bildschirm sollte dann ungefähr so aussehen:



Jetzt sind wir so weit, dass wir Attribute und Methoden in der Klasse `Pet` deklarieren können. In Java müssen Klassen und

Methoden jeweils in geschweiften Klammern eingeschlossen werden. Jede offene geschweifte Klammer muss eine passende schliessende Klammer haben:

```
class Pet{
}
```

Um Variablen für die Attribute der Klasse deklarieren zu können, müssen wir Datentypen für sie auswählen. Ich schlage für das Alter den Typ `int` vor, für das Gewicht und die Größe `float`, und für die Farbe des Haustiers `String`.

```
class Pet{
   int age;
   float weight;
   float height;
   String color;
}
```

Im nächsten Schritt müssen wir ein paar Methoden zu dieser Klasse hinzufügen. Bevor du eine Methode deklarierst, solltest du entscheiden, ob sie irgendwelche Argumente benötigt und ob sie einen Wert zurückgibt:

- ✍✍Die Methode `sleep()` wird nur eine Nachricht ausgeben *Gute Nacht, bis morgen* – sie braucht keine Argumente und wird keinen Wert zurückgeben.

- ✍✍Das gleiche gilt für die Methode `eat()`. Sie wird die Nachricht *Ich habe solchen Hunger… gib mir Nachos!* ausgeben.

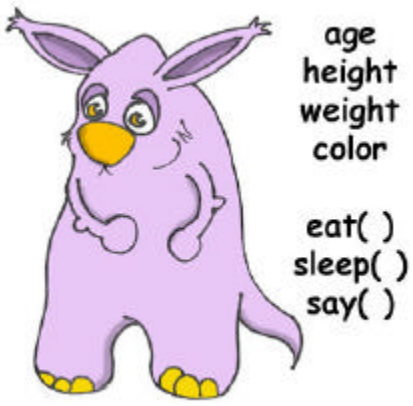
- ✍✍Die Methode `say()` wird ebenfalls eine Nachricht ausgeben, aber das Haustier soll das Wort oder den Satz "sagen", den wir ihm vorgeben. Wir werden der Methode dieses Wort als *Argument* übergeben. Die Methode wird dann aus diesem Argument einen Satz

bilden und den Satz an das aufrufende Programm zurückgeben.

Die neue Version der Klasse `Pet` sieht so aus:

```
public class Pet {
 int age;
 float weight;
 float height;
 String color;

 public void sleep(){
   System.out.println(
      "Gute Nacht, bis morgen");
 }

 public void eat(){
   System.out.println(
   "Ich habe solchen Hunger... gib mir Nachos!");
 }

 public String say(String aWord){
   String petResponse = "OK!! OK!! " + aWord;
   return petResponse;
 }
}
```

Diese Klasse repräsentiert ein freundliches Wesen aus der wirklichen Welt:



Ich möchte jetzt etwas zur Signatur der Methode `sleep()` sagen:

```
public void sleep()
```

Sie sagt uns, dass diese Methode von jeder anderen Java Klasse aus aufgerufen werden kann (`public`= "öffentlich"), und sie gibt keine Daten zurück (`void`= "leer"). Die leeren Klammern bedeuten, dass diese Methode keine Argumente hat, denn sie benötigt keine Daten aus der äusseren Welt – sie gibt immer den gleichen Text aus.

Die Signatur der Methode `say()` sieht so aus:

```
public String say(String aWord)
```

Diese Methode kann ebenfalls von jeder anderen Java Klasse aufgerufen werden, aber sie muss einen Text zurückgeben und das ist die Bedeutung des Schlüsselwortes `String` vor dem Methodennamen. Ausserdem erwartet sie einen Text von aussen, deshalb das Argument `String aWord`.

Wie kannst du entscheiden, ob eine Methode einen Wert zurückgeben soll, oder nicht? Wenn eine Methode irgendwelche Veränderungen an Daten vornimmt, und das Ergebnis dieser Veränderungen von der aufrufenden Klasse verwendet werden

soll, dann muss sie einen Wert zurückgeben. Du könntest jetzt sagen, die Klasse `Pet` hat gar keine aufrufende Klasse! Das stimmt, also lass uns eine erzeugen. Wir nennen sie `PetMaster` ("Herrchen" bzw. "Frauchen"). Die Klasse wird eine Methode `main()` haben, und diese wird sich mit der Klasse `Pet` unterhalten. Erzeuge also eine weitere Klasse in Eclipse, `PetMaster`, und diesmal machst du ein Kreuzchen bei der Frage, ob die Methode `main()` erzeugt werden soll. Erinnere dich, ohne diese Methode kannst du die Klasse nicht als Programm laufen lassen. Ändere den von Eclipse generierten Code bis er so aussieht:

```
public class PetMaster {

  public static void main(String[] args) {

   String petReaction;

   Pet myPet = new Pet();

   myPet.eat();
   petReaction = myPet.say("Quiek!! Quiek!!");
   System.out.println(petReaction);

   myPet.sleep();

  }
}
```

Vergiss nicht, *Ctrl-S* zu drücken, um diese Klasse zu speichern und zu compilieren!
Um die Klasse `PetMaster` laufen zu lassen, klicke in den Eclipse Menüs *Run*, *Run…* , *New* und tippe den Namen der Startklasse ("main class") ein: `PetMaster`. Drücke den Knopf *Run* und das Programm wird den folgenden Text ausgeben:

```
Ich habe solchen Hunger... gib mir Nachos!
OK!! OK!! Quiek!! Quiek!!
Gute Nacht, bis morgen
```

`PetMaster` ist die *aufrufende Klasse*, und sie beginnt damit, eine *Instanz* des Objektes `Pet` zu erzeugen. Sie deklariert eine Variable `myPet` (“meinHaustier”) und benutzt dazu den Java Operator `new` (“neu”):

```
Pet myPet = new Pet();
```

Diese Zeile deklariert eine Variable vom Typ `Pet` (richtig, du kannst alle selbst erzeugten Klassen wie neue Java Datentypen verwenden). Die Variable `myPet` weiss jetzt, wo die Instanz von `Pet` im Speicher des Computers erzeugt wurde, und du kannst diese Variable benutzen, um die Methoden der Klasse `Pet` aufzurufen, zum Beispiel:

```
myPet.eat();
```

Wenn eine Methode einen Wert zurückgibt, solltest du sie auf etwas andere Weise aufrufen. Deklariere eine Variable, die den gleichen Typ hat wie der Rückgabewert der Methode. Jetzt kannst du diese Methode so aufrufen:

```
String petReaction;

petReaction = myPet.say("Quiek!! Quieck!!");
```

An diesem Punkt wird der zurückgegebene Wert in der Variablen `petReaction` gespeichert und wenn du sehen willst, was darin ist, so geht’s:

```
System.out.println(petReaction);
```

Vererbung – Fische sind auch Haustiere

Unsere Klasse `Pet` wird uns helfen eine weitere wichtige Eigenschaft von Java kennenzulernen, die man *Vererbung* (“inheritance”) nennt. Im wirklichen Leben erbt jeder einige Eigenschaften von den Eltern. Ähnlich kannst du in Java eine neue Klasse erzeugen, die auf einer existierenden aufbaut.

Die Klasse `Pet` hat Verhalten und Attribute, die vielen Haustieren gemeinsam sind – sie essen, sie schlafen, manche geben Laute von sich, sie haben verschiedene Farben und so weiter. Auf der anderen Seite sind Haustiere ganz verschieden – Hunde bellen, Fische schwimmen und sind stumm, Papageien sprechen besser als Hunde. Aber alle essen, schlafen, haben ein Gewicht und eine Grösse. Das ist der Grund, warum es einfacher ist eine Klasse `Fish` zu erzeugen, die gewisses gemeinsames Verhalten und Attribute von der Klasse `Pet` *erbt*, als Klassen wie `Dog` (“Hund”), `Parrot` (“Papagei”) oder `Fish` (“Fisch”) jeweils von Grund auf neu zu bauen.
Das spezielle Schlüsselwort `extends` (“erweitert”) vollbringt dieses Kunststück:

```
class Fish extends Pet{

}
```

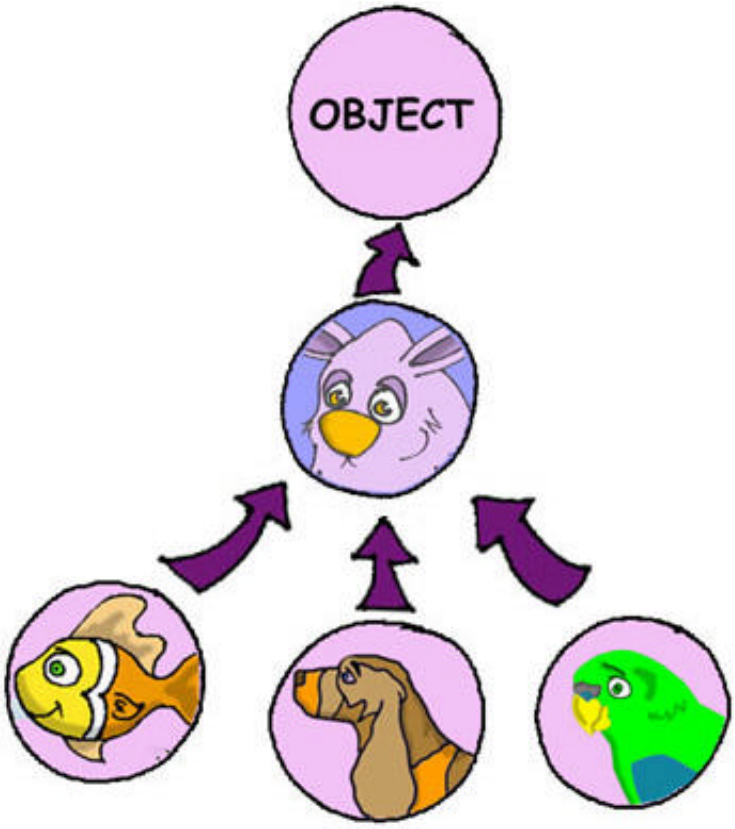
Man kann sagen, dass unser `Fish` eine *Subklasse* (engl. “subclass”, auch *abgeleitete Klasse* genannt) der Klasse `Pet` ist, und die Klasse `Pet` ist eine *Superklasse* (engl. “superclass”, *Basisklasse*) der Klasse `Fish`. Mit anderen Worten, du benutzt die Klasse `Pet` als Vorlage für die Klasse `Fish`.

Selbst wenn du die Klasse `Fish` genau so lässt wie sie bis jetzt ist, kannst du bereits alle Methoden und Attribute der Klasse `Pet` verwenden. Schau’dir das an:

```
Fish myLittleFish = new Fish();
myLittleFish.sleep();
```

Obwohl wir bisher überhaupt keine Methoden in der Klasse `Fish` deklariert haben, dürfen wir die Methode `sleep()` ihrer Basisklasse aufrufen!

Mit Eclipse abgeleitete Klassen zu erzeugen ist wirklich ein Kinderspiel! Wähle in den Menüs *File*, *New*, *Class*, und tippe `Fish` als Name für die neue Klasse. Ersetze `java.lang.Object` im Feld *superclass* durch das Wort `Pet`.



Wir wollen aber nicht vergessen, dass wir eine abgeleitete Klasse von `Pet` erzeugt haben, um einige Eigenschaften hinzuzufügen die nur Fische haben, und ausserdem den Code

wiederzuverwenden, den wir bereits für ein allgemeines Haustier geschrieben haben.

> Es wird Zeit ein Geheimnis zu lüften – alle Klassen in Java erben von der Super-Duper-Klasse `Object`, egal ob man das Schlüsselwort `extends` verwendet oder nicht.
>
> Aber Java Klassen können nicht zwei verschiedene Eltern haben. Wenn das bei den Menschen auch so wäre, dann wären Kinder nicht abgeleitete Klassen ihrer beiden Eltern, sondern alle Buben würden von Adam abstammen und alle Mädchen von Eva ✍.

Nicht alle Haustiere können tauchen, aber Fische können das ganz bestimmt. Fügen wir also eine neue Methode `dive()` (“tauchen”) zur Klasse `Fish` hinzu.

```
public class Fish extends Pet {

   int currentDepth=0;

   public int dive(int howDeep){
      currentDepth=currentDepth + howDeep;
      System.out.println("Diving for " + howDeep +
                                          " feet");
      System.out.println("I'm at " + currentDepth +
                           " feet below sea level");
      return currentDepth;
   }
}
```

Die Methode `dive()` hat ein *Argument* `howDeep` (“wieTief”) das dem Fisch sagt wie tief er gehen soll. Wir haben ausserdem eine Klassenvariable `currentDepth` (“aktuelleTiefe”) deklariert, die speichern wird wie tief der Fisch gerade ist, und die jedesmal einen neuen Wert erhalten wird, wenn du die Methode `dive()`

aufrufst. Diese Methode gibt den jeweiligen Wert der Variablen `currentDepth` an die aufrufende Klasse zurück.

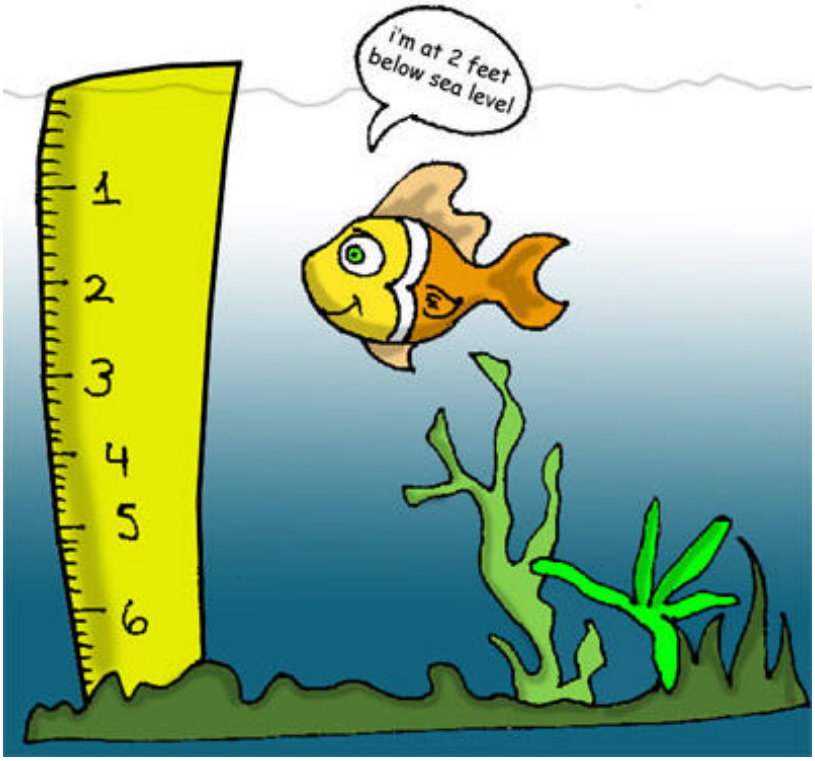
Bitte erzeuge jetzt eine weitere Klasse `FishMaster` die so aussieht:

```
public class FishMaster {

      public static void main(String[] args) {

            Fish myFish = new Fish();

            myFish.dive(2);
            myFish.dive(3);

            myFish.sleep();
      }
}
```

Die Methode `main()` instantiiert das Objekt Fish und ruft dessen Methode `dive()` zweimal mit verschiedenen Argumenten auf. Danach ruft es die Methode `sleep()` auf. Wenn du das Programm `FishMaster` laufen lässt, wird es die folgenden Nachrichten ausgeben:

```
Diving for 2 feet
I'm at 2 feet below sea level
Diving for 3 feet
I'm at 5 feet below sea level
Good night, see you tomorrow
```

Hast du bemerkt, wie der `FishMaster` ausser den Methoden, die in der Klasse `Fish` definiert sind, auch Methoden der Basisklasse `Pet` aufruft? Darin besteht der Sinn von Vererbung – du brauchst den Code der Klasse `Pet` nicht zu kopieren – benutze einfach das Wort `extends`, und die Klasse `Fish` kann die Methoden von `Pet` benutzen!



Eine Bemerkung noch: obwohl die Methode `dive()` den Wert von `currentDepth` zurückgibt, macht unser `FishMaster` keinen Gebrauch davon. Das ist in Ordnung, unser `FishMaster` braucht diesen Wert nicht, aber es könnte andere Klassen geben, die auch die Klasse `Fish` verwenden, und die könnten das nützlich finden. Denk' zum Beispiel an eine Klasse `FishTrafficDispatcher` ("FischVerkehrsLeiter"), der die Positionen von anderen Fischen kennen muss bevor er einem Fisch erlauben darf zu tauchen, damit es keine Verkehrsunfälle gibt ✍.

Methoden überschreiben

Wie du weißt, sprechen Fische nicht (jedenfalls nicht laut). Aber unsere Klasse `Fish` hat von der Klasse `Pet` die Methode `say()` geerbt. Das bedeutet, dass dich niemand hindern kann zum Beispiel zu schreiben:

```
myFish.say();
```

Nun, jetzt beginnt unser Fisch zu reden… Wenn du nicht willst dass das passiert, dann muss die Klasse `Fish` die Methode `say()` der Klasse `Pet` *überschreiben* (engl. "override"= übersteuern, sich über etwas hinwegsetzen). Und so wird's gemacht: wenn du in einer abgeleiteten Klasse eine Methode mit genau der gleichen Signatur deklarierst wie in ihrer Basisklasse, dann wird die Methode der abgeleiteten Klasse anstelle derjenigen der Basisklasse benutzt. Fügen wir also die Methode `say()` zur Klasse `Fish` hinzu.

```
public String say(String something){
  return "Don't you know that fish do not talk?";
}
```

Und jetzt füge folgenden Code zur Methode `main()` der Klasse `FishMaster` hinzu:

```
String fishReaction;
fishReaction = myFish.say("Hello");
System.println(fishReaction);
```

Lass'das Programm laufen und es wird ausgeben:

```
Don't you know that fish do not talk?
```

Dies beweist, dass die Methode `say()` von `Pet` *überschrieben* oder, mit anderen Worten, unterdrückt wurde.

Wenn die Signatur einer Methode das Schlüsselwort `final` enthält, dann kann diese Methode nicht überladen werden. Beispiel:

```
final public void sleep(){…}
```

Wow! In diesem Kapitel haben wir eine Menge gelernt – jetzt müssen wir mal eine Pause machen.

Weiterführende Literatur

1.Java Datentypen:
http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html

2.Über Vererbung:
http://java.sun.com/docs/books/tutorial/java/concepts/inheritance.html

Übungen

1. Erzeuge eine neue Klasse `Car` ("Auto") mit folgenden Methoden:

```
public void start()
public void stop()
public int drive(int howlong)
```

Die Methode `drive()` ("fahren") soll die Strecke zurückgeben, die das Auto in der angegebenen Zeit (`howlong`= wieLange) zurückgelegt hat. Benutze die folgende Formel um die zurückgelegte Strecke ("distance") auszurechnen:

```
distance = howlong*60;
```

2. Schreibe eine weitere Klasse `CarOwner` ("AutoBesitzer") die eine Instanz des Objektes `Car` erzeugt und ihre Methoden aufruft. Das Ergebnis jeder Methode soll mit `System.out.println()` ausgegeben werden.

Übung für Schlaumeier

Erzeuge eine abgeleitete Klasse von `Car`, nenne sie `JamesBondCar` und überschreibe die Methode `drive()`. Benutze folgende Formel um die Distanz auszurechnen:

```
distance = howlong*180;
```

Denk' dir ein paar lustige Nachrichten aus!