Глава 3. Домашнее Животное и Рыба на Языке Java

Программы на языке Java состоят из классов, которые представляют объекты реального мира. Люди понимают по-разному что значит хороший стиль программирования, однако большинство согласно что нужно стараться программировать в так называемом объектно-ориентированном стиле. Это значит, что хорошие программисты сначала решают какие объекты включить в программу и какие Java классы будут их представлять, а только потом уже они начинают писать программу.

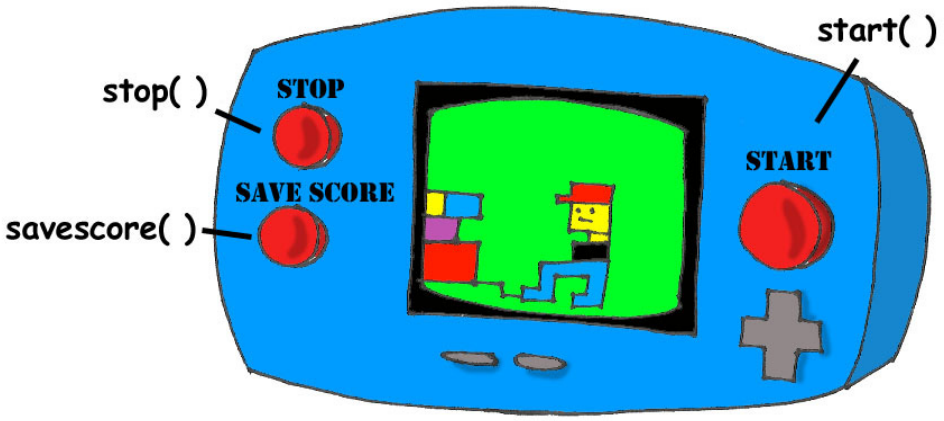
Классы и объекты

Java-классы могут иметь *методы* и *аттрибуты*.

Методы определяют *что класс может делать*.

Аттрибуты это просто характеристики класса.

Давайте придумаем и обсудим класс который будет называться `VideoGame` (видео игра). Этот класс может иметь несколько методов, из которых будет ясно что могут делать объекты этого класса: начать игру, остановить её, сохранить (запомнить) счет, и так далее.
А ещё этот класс может иметь какие-нибудь аттрибуты, например цена, цвет экрана, количество ручек управления, и все такое прочее.



На языке Java наш класс может выглядеть вот так:

```
class VideoGame {
  String color;     // цвет
  int price;        // цена

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

Наш класс `VideoGame` будет похож на другие классы, которые представляют видео игры – у всех у них есть экраны правда разного цвета и размера, все они выполняют поохожие действия, и все они стоят денег.

А теперь давайте добавим побольше конкретных деталей и создадим другой класс называемый `GameBoyAdvance` – это популярная в Америке электронная игра. Этот класс тоже принадлежит семейству видео игр, однако он будет имеет некоторые аттрибуты, которые имеет только игра GameBoy Advance, например тип кассеты.

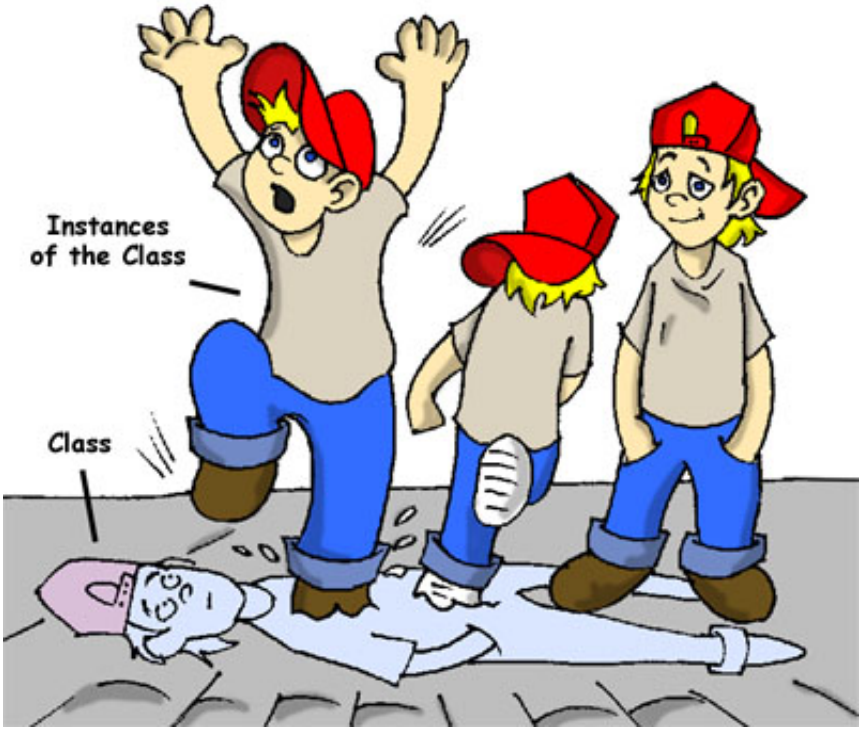
```
class GameBoyAdvance {
  String cartridgeType;      // тип кассеты
  int screenWidth;           // ширина экрана

  void startGame() {

  }
  void stopGame() {

  }
}
```

В этом примере класс `GameBoyAdvance` объявляет два аттрибута – `cartridgeType` and `screenWidth` и два метода – `startGame()` и `stopGame()`. Но эти методы ещё не готовы выполнять какие-либо

действия, потому что у них нет никакого программного кода между фигурными скобочками.

> Помимо слова *класс* вам придется привыкать к ещё одному новому значению слова *объект*.
>
> А фраза “создать экземпляр объекта” просто значит создать копию объекта в памяти компьютера согласно описанию этого класса.

Фабричное описание игры `GameBoy Advance` имеет такое-же отношение к уже сделанной игре, как Java-класс к его экземпляру в памяти компьютера. А процесс изготовления настоящих игр на основе токого описания похож на процесс создания экземпляров объектов `GameBoyAdvance` на языке Java.



Во многих случаях программа может пользоваться классом только после создания его экземпляра. Производители игр ведь тоже создают тысячи копий по одному и тому же описанию. Не смотря на то, что эти копии представляют тот же самый класс, их аттрибуты могут иметь разные значения - какие-то из них голубые, какие-то перламутровые, и так далее. Иными словами, программа может создавать множество экземпляров объектов `GameBoyAdvance`.

Типы Данных

Переменные представляют аттрибуты класса, параметры метода или просто могут использоваться для краткосрочного хранения каких-нибудь данных. Переменные сначала должны быть объявлены, и только после этого ими можно пользоваться.

Помните уравнения типа y=x+2? На языке Java вам придется объявить переменные x и y используя какой-нибудь числовой тип данных, например целое число (integer или `int`) или число двойной длины (`double`):

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

Следующие две строчки кода присваивают значение этим переменным. Если ваша программа присвоит переменной икс значение пять, переменная игрек будет равна семи:

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

Java разрешает менять значение переменной немного необычным способом. Вот например, как можно изменить значение переменной игрек с пяти на шесть:

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

Несмотря на два знака плюс, Java увеличит значение игрека на единичку.

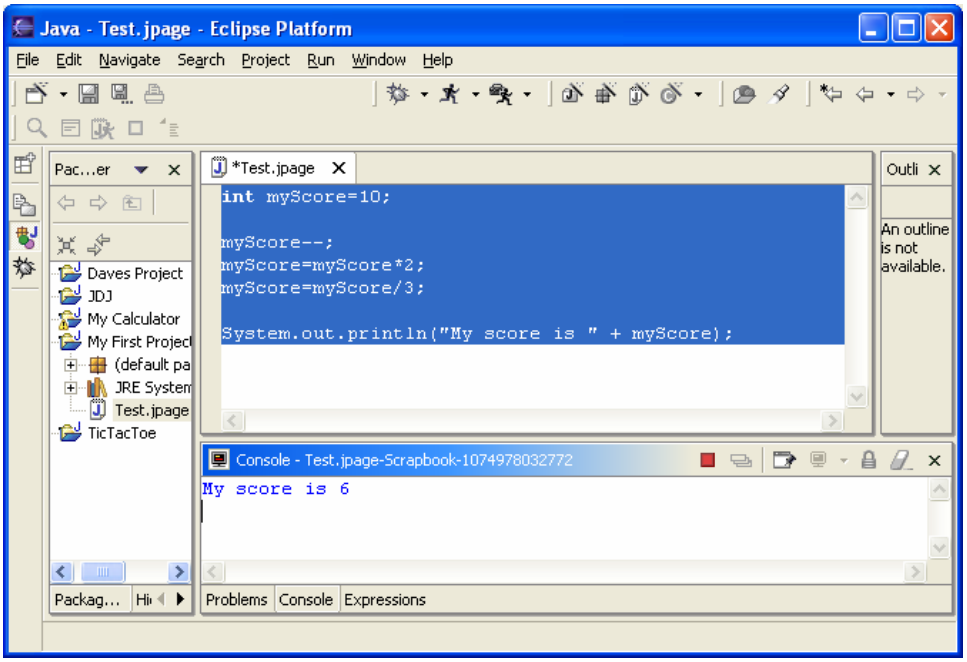
А после вот этого примера, значение переменной `myScore` тоже шесть:

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

Точно также можно делать умножение, деление и вычитание, вот смотрите:

```
int myScore=10;

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

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

Что-же напечатает этот код (кстати, “My score is” переводится как ”Мой счет”)? У приложения Eclipse, где мы теперь пишем программы, есть классная штука под названием черновик (scrapbook) которая позволяет вам легко тестировать любой кусочек кода даже без создания класса. Выберите меню *File*, *New*, *Scrapbook Page* и напечатайте слово Test – это будет имя вашего файла-черновика. Теперь напечатайте в черновике пять строчек предыдущего примера, высветите их и нажмите на кнопочку с маленьким увеличительным стеклом:



Чтобы получить результат вычислений, просто нажмите на закладку Console внизу экрана:

My score is 6

В этом примере параметр метода `println()` был склеен из двух кусочков – текста “My score is ” и значения переменной `myScore`, которая была равна шести. Такое “склеивание” строк из кусочков называется *конкатенация* (*concatenation)*. Несмотря на то, что переменная `myScore` жхранит не текст а число, Java достаточно умна чтобы преобразовать эту переменную в тип данных `String` и потом уже приклеить ее значение к тексту “*My Score is*”.

Вот ещё несколько примеров того, как можно менять значения переменных:

```
myScore=myScore*2;  то же что  myScore*=2;
myScore=myScore+2;  то же что  myScore+=2;
myScore=myScore-2;  то же что  myScore-=2;
myScore=myScore/2;  то же что  myScore/=2;
```

В языке Java есть восемь простых (примитивных) типов данных, и вам решать какими пользоваться в вашей программе. Это конечно зависит от того данные какого типа и размера вам нужно хранить в этих переменных:

- ✓ Четыре типа данных для хранения целых чисел – `byte, short, int, and long`.
- ✓ Два типа данных для значений с десятичной точкой – `float` и `double`.
- ✓ Один тип данных для хранения одной буквы – `char`.

- ✓ Один логический тип называемый `boolean` который может иметь только два значения: `true` или `false` (истина и ложь).

Java разрешает присваивать начальное значение переменной во время ее объявления. Это называется *инициализация переменных*:

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

В последних двух строчках `f` значит `float`, а `l` значит `long`.

Если вы все-же забудете инициализировать переменные, Java сама присвоит ноль числовым переменным, `false` переменных типа `boolean`, и специальный код '\u0000' переменным типа `char`.

А ещё есть специальное ключевое слово `final`, и если оно присутствует в объявлении переменной, вам будет разрешено присвоить значение этой переменной только один раз и вы не сможете уже изменить это значение после. В некоторых языках программирования `final`-переменные называются *константами*. Принято называть эти переменне большими буквами.

```
final String STATE_CAPITAL="Вашингтон";
```

Помимо примитивных типов данных, вы можете использовать классы для объявления переменных. У каждого примитивного типа данных есть соответствующий класс-обертка, например `Integer`, `Double`, `Boolean`, и другие. Эти классы имеют много полезных методов чтобы преобразовывать данные из одного типа в другой.

Примитивный тип данных `char` может хранить только одну букву, но в языке Java существует класс `String` для работы с более длинным текстом:

```
String lastName="Смит";
```

Имена переменных не могут начинаться с цифры и не могут содержать пробелы.

Бит это самая маленькая порция данных , которая может храниться в памяти. Вы можете хранить в бите только 1 или 0.

Байт состоит из восьми битов.

char занимает два байта в памяти компьютера.

int и float занимают четыре байта памяти.

Переменным long и double нужно по восемь байтов.

Числовые типы данных, которые занимают больше памяти могут хранить большие величины.

1 килобайт (KB) это 1024 байтов

1 мегабайт (MB) это 1024 килобайтов

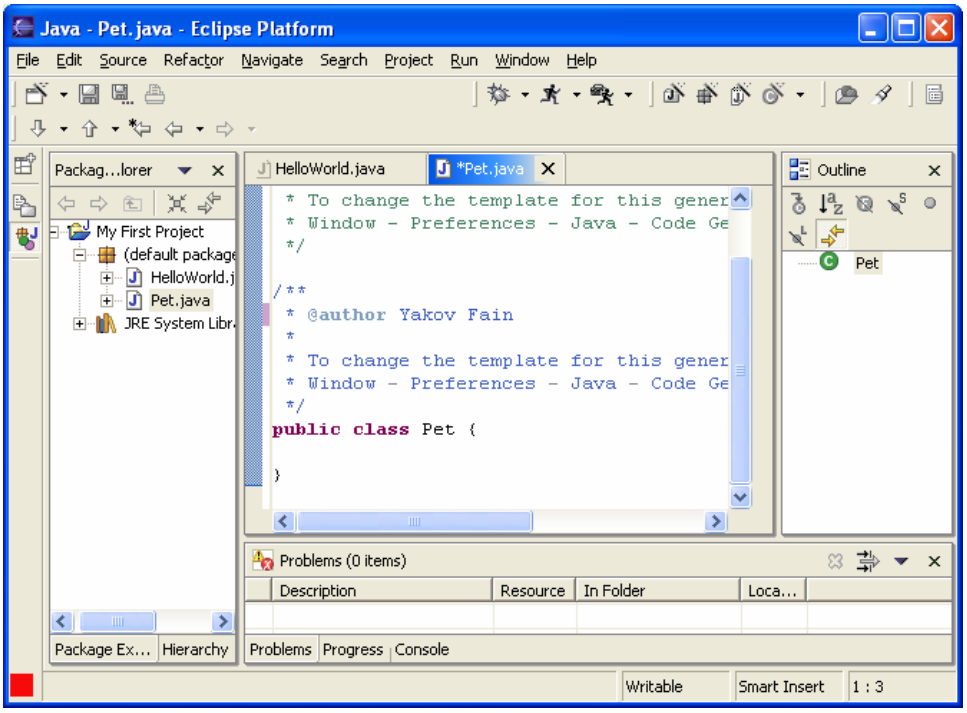
1 гигабайт (GB) имеет 1024 мегабайтов

Создаём Домашнее Животное

Давайте придумаем и создадим класс Домашнее Животное (по английски просто `Pet`). Сначала нушно решить, какие действия наш `Pet` сможет выполнять. Как насчет есть, спать и говорить (eat, sleep, say) ? Мы запрограммируем эти действия в методах класса `Pet`. А ещё мы дадим нашему домашнему животному такие аттрибуты: возраст (age), рост (height), вес (weight) и цвет (color).

Начнем с создания нового класса по названием `Pet` в проекте *My First Project* (смотри главу 2), но при этом не ставьте птичку требующую создания метода `main()`.

Ваш экран будет выглядеть примерно так:



Теперь мы готовы объявить аттрибуты и методы в классе `Pet`. Код в классах и методах должен быть окружен фигурными скобками. Каждая отрывающая скобка должна иметь свою закрывающую:

```
class Pet{
}
```

Давайте выберем типы данных для аттрибутов нашего класса. Я предлагаю `int` для возраста, `float` для веса и роста, `String` для цвета.

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

А сейчас добавим несколько методов в наш класс. Здесь нужно решить будут ли эти методы иметь параметры и возвращать какие-нибудь данные:

- ✓ Метод `sleep()` будет просто печатать фразу *Спокойной ночи, до завтра* – ему не нужны никакие параметры и он не будет возвращать никаких значений.

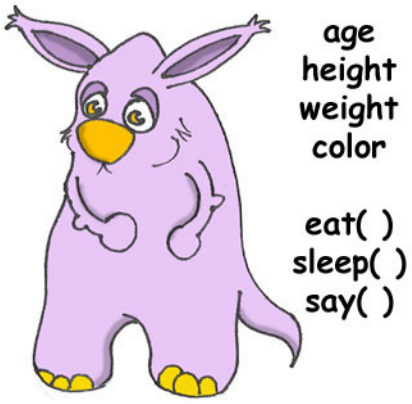
- ✓ То же самое относится и к методу `eat()` – он будет печатать сообщение *Я очень голоден…давайте перекусим чипсами!*

- ✓ Хотя метод `say()` тоже будет печатать сообщение, но наше домашнее животное будет ещё и "произносить" слово или фразу, которую мы ему дадим как *параметер*. Этот метод будет строить фразу включающую значение этого параметра, после чего этого фраза будет возвращаться вызывающей программе.

Новая версия класса `Pet` будет выглядет так:

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

 public void sleep(){
        System.out.println(
                    "Спокойной ночи, до завтра");
 }

 public void eat(){
  System.out.println(
  "Я очень голоден, давайте перекусим чипсами!");
 }

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

Этот класс представляет вот такое симпатичное существо из реального мира:



Давайте поговорим о заголовке метода `sleep()`:

```
public void sleep()
```

Этот заголовок говорит нам, что метод `sleep()` можно вызывать из любого другого класса (`public`) и что метод не возвращает никаких данных (`void`). Пустые скобочки значат что этот метод не имеет параметров – ему ведь не нужны нинакие данные из окружающей среды, потому что он всегда печатает один и тот же текст.

Заголовок метода `say()` выглядит так:

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

Этот метод тоже можно вызывать из любого другого класса, но он ещё и возвращает какой-то текст – это и есть роль ключевого слова `String` стоящего перед именем метода. А кроме того, этот метод ожидает какие-то текстовые данные извне и для этого в заголовок метода включен параметр `String aWord`.

А как же определить – должен-ли метод возвращать данные? Если метод выполняет какие-то действия над данными и должен передать результат этих действий вызывающему классу, то он должен возвращать данные. Вы можете возразить, что у класса `Pet` нет никакого вызывающего класса! Верно, поэтому мы сейчас и напишем класс `PetMaster` (Хозяин). У этого класса будет метод `main()` содержащий код работающий с классом `Pet`.

Создайте новый класс `PetMaster`, но на этот раз поставьте птичку возле в Eclipse возле вопроса *создавать-ли метод main()*. Не забывайте, что без этого метода вы не сможете стартовать программу. Добавьте несколько строчек кода к классу который сделал для вас Eclipse, чтобы он выглядел так.:

```
public class PetMaster {

  public static void main(String[] args) {

   String petReaction;

   Pet myPet = new Pet();

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

   myPet.sleep();

  }
}
```

Не забудьте нажать *Ctrl-S* чтобы сохранить и откомпилировать этот класс! А чтобы стартовать программу `PetMaster`, выберите следующие меню в Eclipse *Run*, *Run…*, *New* и напечатайте имя главного класса - PetMaster. Нажмите на кнопку *Run* и программа напечатает следующий текст:

```
Я очень голоден, давайте перекусим чипсами!
Ну ладно!! Чик!! Чирик!!
Спокойной ночи, до завтра
```

PetMaster – это вызывающий класс и он сначала создает экземпляр объекта `Pet`. Он объявляет переменную `myPet` and и использует оператор `new`:

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

Эта сточка объявляет переменную `myPet` типа `Pet` (да это так, вы можете использовать любые классы созданные вами как новые типы данных). Теперь переменная `myPet` знает место в памяти где был создан экземпляр объекта `Pet`, и можно пользоваться этой переменной чтобы вызывать любые методы класса `Pet`, например:

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

Если метод возвращает какое-нибудь значение, его можно вызывать по другому. Объявите переменную того-же типа что и возвращаемое значение и вызывайте метод так, чтобы присвоить это значение переменной, например так:

```
String petReaction;

petReaction = myPet.say("Чик!! Чирик!!");
```

Теперь возвращенное значение находится в переменной `petReaction` и его очень просто можно распечатать:

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

Наследование – Рыбка Тоже Домашнее Животное

Наш класс Pet познакомит вас с ещё одной важной особенностью языка Java, которая называется *наследование*. В реальном мире каждый человек наследует что-то от своих родителей. В мире Java вы тоже можете создать новый классспо типу уже существующего.

Класс Pet и ведет себя, и имеет аттрибуты типичные для многих домашних животных – они едят, спят, некоторые из них издают звуки, их кожа имеет цвет и так далее. С другой стороны, домашние животные отличаются друг от друга – собаки лают, рыбки беззвучно плавают, попугайчики разговаривают лучше чем собаки. И все-же, все они спят, едят, и имеют рост и вес. Поэтому гораздо легче создать класс Fish (рыба) так, чтобы он унаследовал общие черты и поведение у класса Pet, чем каждый раз создавать с начала классы для собак, попугаев и рыб.

Для этого и существует специальное ключевое слово `extends`:

```
class Fish extends Pet{

}
```

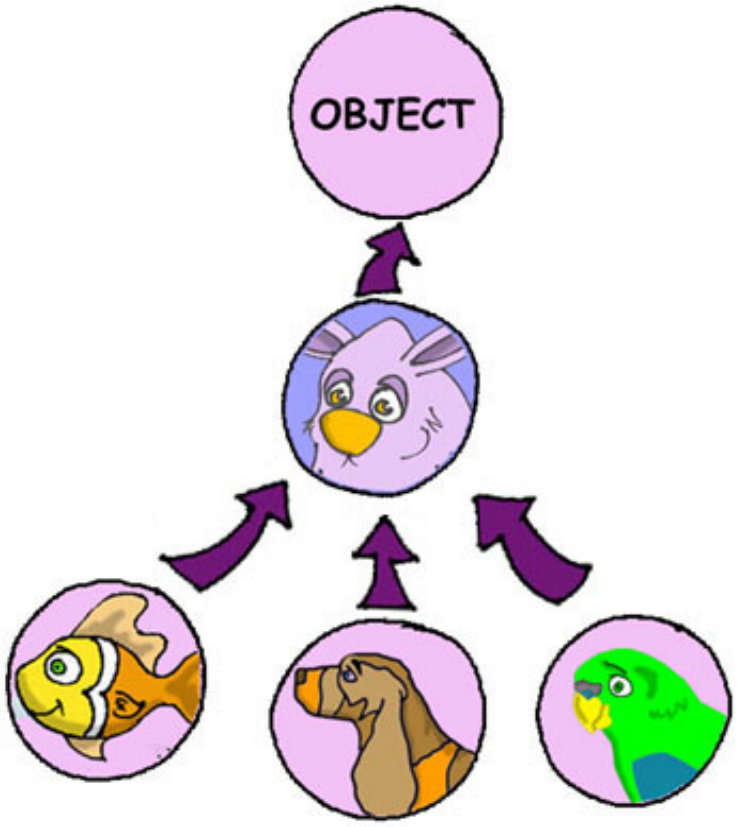
Теперь вы имеете полное право сказать что `Fish` - это *под-класс* класса `Pet`, а класс `Pet` – это *супер-класс* класса `Fish`. Мы использовали класс `Pet` как свообразный шаблон для создания класса `Fish`.

Даже если вы оставите класс `Fish` таким как он есть сейчас, всё равно уже можно использовать каждый метод и аттрибут унаследованный из класса `Pet`. Посмотрите:

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

Хоть мы ещё и не объявляли никаких методов в классе `Fish`, уже можно вызывать метод `sleep()` находящийся в его супер-классе!

Нет ничего легче создания под-классов в приложении Eclipse! Выберите меню *File*, *New*, *Class* и напечатайте слово `Fish` как имя класса. Замените в поле супер-класс `java.lang.Object` на слово `Pet`.



Не забывайте, что мы создаем под-класс класса `Pet` чтобы добавить то, что присуще только рыбам, а общий для всех животных код, объявленнуй в супер-классе, мы просто используем.

Пора расскать маленький секрет – все классы в языке Java унаследованы из супер-дупер класса `Object`, даже если вы и не использовали ключевое слово `extends`.

В отличие от людей, Java-классы не могут иметь двух родителей. А если бы у нас это было как в языке Java , дети не были-бы “под-классами” своих родителей, а все мальчики происходили-бы от Адама, а девочки от Евы ☺.

Не все домашние животные могут нырять, но рыбки, конечно-же могут. Давайте добавим к классу `Fish` метод `dive()` - ныряй.

```
public class Fish extends Pet {

   int currentDepth=0;

   public int dive(int howDeep){
      currentDepth=currentDepth + howDeep;
      System.out.println("Ныряю на глубину "
                              + howDeep + " футов");
      System.out.println("Я на глубине  "
         + currentDepth + " футов ниже уровня моря");
      return currentDepth;
   }
}
```

У метода `dive()` есть параметр `howDeep`, который “говорит” рыбке как глубоко она должна нырнуть. Мы ещё объявили переменную `currentDepth`, куда будем помещать текущее значение глубины при каждом вызове метода `dive()`. Этот метод возвращает значение переменной `currentDepth` вызывающему классу.

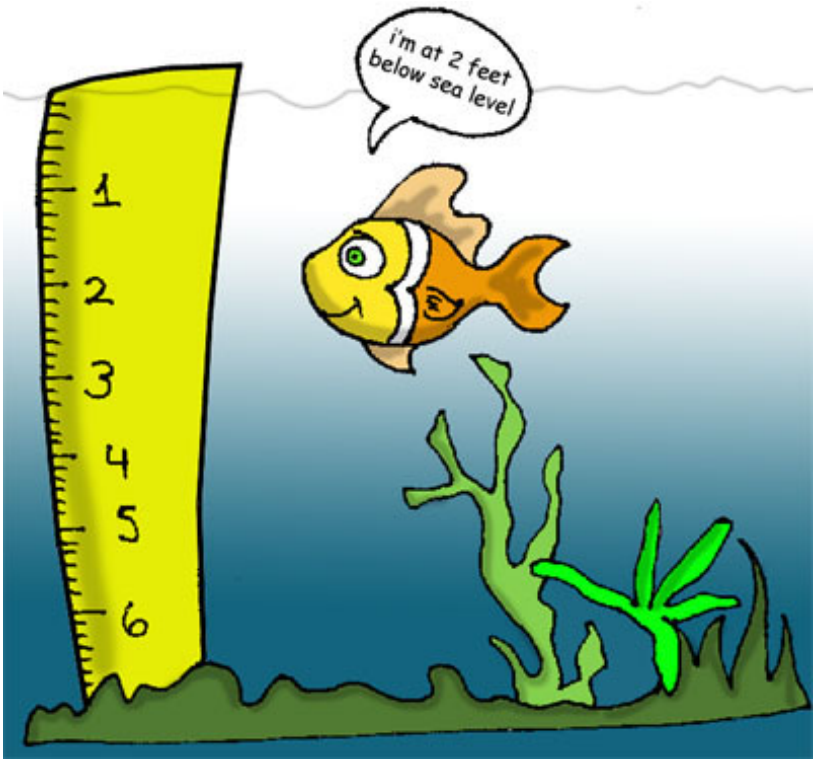
Сделайте, пожалуйста, вот такой класс `FishMaster`:

```
public class FishMaster {

      public static void main(String[] args) {

            Fish myFish = new Fish();

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

            myFish.sleep();
      }
}
```

Метод `main()` создает экземпляр объекта Fish и дважды вызывает его метод `dive()` с разными параметрами. После этого он вызывает метод `sleep()`. Во время выполнения, программа `FishMaster`, напечатает следующее:

```
Ныряю на глубину 2 футов
Я на глубине 2 футов ниже уровня моря
Ныряю на глубину 3 футов
Я на глубине 5 футов ниже уровня моря
Спокойной ночи, до завтра
```

Вы заметили, что `FishMaster` вызывает не только методы объявленные в классе `Fish`, но также и методы его супер-класса `Pet`? То-то! В этом и есть вся прелесть наследования – вам не нужно копировать код из класса `Pet`. Просто напишите слово `extends` и класс `Fish` сможет пользоваться методами класса `Pet`!



Да, вот ещё что, хотя метод `dive()` и возвращает значение переменной `currentDepth`, наш `FishMaster` им не пользуется. Это не беда, просто нашему классу `FishMaster` оно не нужно. Но каким-нибудь другим классам, которые тоже могут работать с классом `Fish`, это значение может быть очень даже полезно. Представьте, например класс `FishTrafficDispatcher` (регулировщик движения рыб), который должен знать положения других рыб в море прежде чем разрешить ныряние во избежание дорожно-транспортных происшествий ☺.

Переопределение Методов

Вы конечно знаете, что рыбы не говорят (по крайней мере он не делают это громко). Но наш класс `Fish` был унаследован из класса `Pet`, у которого есть метод `say()`. Это значит, что вы беспрепятственно можете написать что-то в этом роде:

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

Ну и ну, наши рыбки заговорили… Чтобы избежать этого, в классе `Fish` нужно переопределить (override) метод `say()`, обьявленный в классе `Pet`. Это работает так: если вы объявляете в под-классе метод имеющий точно такой-же заголовок как в его-же супер-классе, Java выполнит метод под-класса, вместо метода супер-класса. Давайте добавим к классу `Fish` метод `say()`.

```
public String say(String something){
 return "Ты чё не знаешь, что рыбы не разговаривают?";
}
```

А теперь вызовем метод `say()` из метода `main()` класса `FishMaster`:

```
myFish.say("Привет");
```

Выполните эту программу и оне напечатает следующее:

```
Ты чё не знаешь, что рыбы не разговаривают?
```

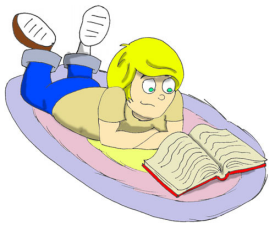
Это подтверждает, что метод `say()` класса `Pet` был переопределен.

Если заголовок метода включает ключевое слово `final`, такой метод переопределить нельзя, например:

```
final public void sleep(){…}
```

Вот это да! Мы изучили много нового в этой главе – давайте передохнём.

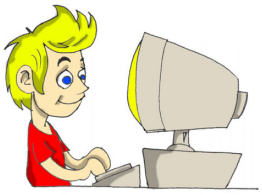
Дополнительное Чтение



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

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

Практика



1. Создайте новый класс `Car` (автомобиль) и включите в него следующие методы:

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

Метод `drive()` (едь) должен возвращать общее расстояние пройденное автомобилем за заданное время. Используйте следующую формулу дла рассчёта расстояния:

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

2. Создайте ещё один класс `CarOwner` (хозяин автомобиля) который будет создавать экземпляр объекта `Car` и вызывать его методы. Результат каждого такого вызова должен быть напечатан с помощью `System.out.println()`.

Практика для Умников

Сделайте под-класс класса `Car`, назовите ее `JamesBondCar` (автомобиль Джемса Бонда) и переопределите в нем метод `drive()`. Используйте следующую формулу дла рассчёта расстояния:

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

Будьте изобретательны! Печатайте смешные сообщения.