对象和类

英文标题【Object and Classes】

Java 是一种面向对象的语言。作为一个面向的语言，Java 具有面向对象的特性，Java 能够支持下面的一些基本概念

- 多态（Polymorphism）
- 继承（Inheritance）
- 封装（Encapsulation）
- 抽象（Abstraction）
- 类（Classes）
- 对象（Objects）
- 实例（Instance）
- 方法（Method）
- 消息传递（Message Passing）

在这个章节中，我们将会重点进入 Java 的一个概念 - 类和对象。

- 对象（Object） 对象具有状态和行为。例如：一条狗具有状态 - 颜色，名字，品种；同时还具有行为 - 摇动尾巴，叫唤，吃东西。一个对象就是一个类的实例。
- 类（Class） 一个类可以定义一些模板或蓝图，这些用来描述一类对象的状态和行为。

本页中的内容：

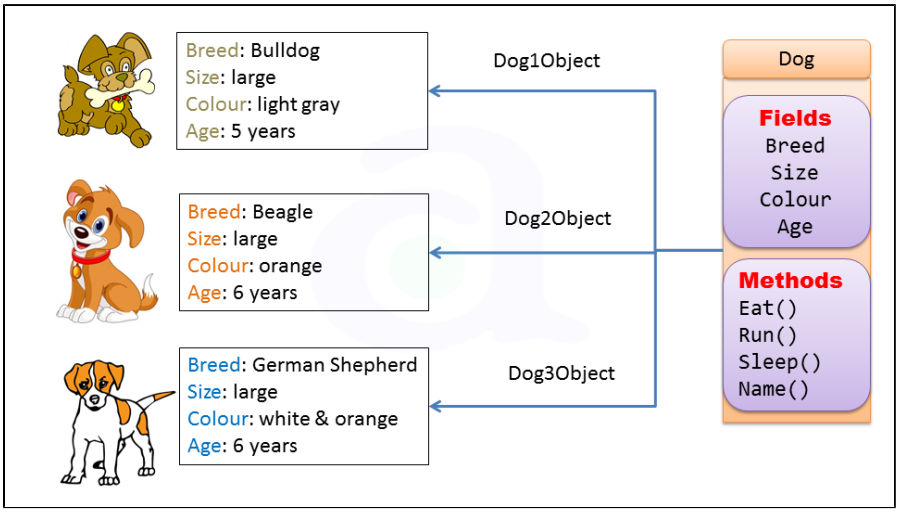
相关页面：

- Basic Syntax

我们以狗为例子，在这个例子中。我们定义了一个狗的类，这个用于描述狗的定义。

在实际中，我们可能会遇到各种不同类型的狗，这些类型的狗都会有自己的状态和行为，我们成为对象。

例如，我们现在要描述一条拉布拉多犬，那么我们可以用狗这个类创建拉布拉多犬这个对象，在 Java 中使用关键字 new 进行创建。



Java 中的对象

现在让我们深入了解什么是对象。看看周围真实的世界，会发现身边有很多对象，车，狗，人等等。所有这些对象都有自己的状态和行为。

拿一条狗来举例，它的状态有：名字、品种、颜色，行为有：叫、摇尾巴和跑。

如果你拿软件对象来对比实际世界中的对象，你会发现他们直接有非常大的相似性。

软件对象也有状态和行为。软件对象的状态就是属性，行为通过方法体现。

在软件开发中，方法操作对象内部状态的改变，同时对象和对象直接的通信也是通过方法来实现的。

Java 中的类

一个类可以被看做是从一个对象中创建出来的蓝图。

请参考下面一个方法的示例。

例如

```
public class Dog{
        String breed;
        int age;
        String color;

        void barking() {
        }

        void hungry() {
        }

        void sleeping() {
        }
}
```

一个对象可以包含有下面的一些变量类型。

- 局部变量（Local variables） 变量被定义在方法或者构造方法中，这种变量被称为局部变量。变量声明和初始化都是在方法中，方法结束后，变量就会自动销毁。
- 实例变量（Instance variables）— 这个变量也被称为成员变量。这个变量定义在类中，方法体之外的变量。这种变量在创建对象的时候实例化。成员变量可以被类中方法、构造方法和特定类的语句块访问。
- 类变量（Class variables）— 这个变量也被称为静态变量。类变量也声明在类中，方法体之外，但必须声明为static类型。

一个类可以拥有多个方法，在上面的例子中：barking()、hungry() 和 sleeping() 都是Dog类的方法。

下面是 Java 语言中有关类的更多一些重要概念。

实例变量	静态变量（类变量）
实例变量声明在一个类中，但在方法、构造方法和语句块之外； 当一个对象被实例化之后，每个实例变量的值就跟着确定； 实例变量在对象创建的时候创建，在对象被销毁的时候销毁； 实例变量的值应该至少被一个方法、构造方法或者语句块引用，使得外部能够通过这些方式获取实例变量信息； 实例变量对于类中的方法、构造方法或者语句块是可见的。一般情况下应该把实例变量设为私有。通过使用访问修饰符可以使实例变量对子类可见； 实例变量具有默认值。数值型变量的默认值是0，布尔型变量的默认值是false，引用类型变量的默认值是null。变量的值可以在声明时指定，也可以在构造方法中指定； 实例变量可以直接通过变量名访问。但在静态方法以及其他类中，就应该使用完全限定名：ObejectReference.VariableName。	类变量也称为静态变量，在类中以static关键字声明，但必须在方法构造方法和语句块之外。 无论一个类创建了多少个对象，类只拥有类变量的一份拷贝。 静态变量除了被声明为常量外很少使用。常量是指声明为public/private，final和static类型的变量。常量初始化后不可改变。 静态变量储存在静态存储区。经常被声明为常量，很少单独使用static声明变量。 静态变量在程序开始时创建，在程序结束时销毁。 与实例变量具有相似的可见性。但为了对类的使用者可见，大多数静态变量声明为public类型。 默认值和实例变量相似。数值型变量默认值是0，布尔型默认值是false，引用类型默认值是null。变量的值可以在声明的时候指定，也可以在构造方法中指定。此外，静态变量还可以在静态语句块中初始化。 静态变量可以通过：ClassName.VariableName的方式访问。 类变量被声明为public static final类型时，类变量名称最好使用大写字母。如果静态变量不是public和final类型，其命名方式与实例变量以及局部变量的命名方式一致。

请参考下面的示例代码：

示例代码 （GitHub）

```
package com.ossez.lang.tutorial.tests;

import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import junit.framework.TestCase;

/**
 * Object of VariableOssez
 *
 * @author YuCheng
 *
 */
class OssezVariable {
        int s1, s2;
        static int s3;

        OssezVariable(int x, int y, int z) {
                s1 = x;
                s2 = y;
                s3 = z;
        }
}

/**
 *
 * @author YuCheng
 *
 */
public class VariableTest extends TestCase {

        private final static Logger logger = LoggerFactory.getLogger(VariableTest.class);

        /**
         * Do RetsServerConnection Test
         */
        @Test
        public void testStaticVariableChange() {

                OssezVariable objA = new OssezVariable(1, 2, 3);
                logger.debug("s1/s2/s3 - [{}]/[{}]/[{}]", objA.s1, objA.s2, OssezVariable.s3);

                OssezVariable objB = new OssezVariable(4, 5, 6);
                logger.debug("s1/s2/s3 - [{}]/[{}]/[{}]", objA.s1, objA.s2, OssezVariable.s3);
                logger.debug("s1/s2/s3 - [{}]/[{}]/[{}]", objB.s1, objB.s2, OssezVariable.s3);
        }

}
```

运行的结果为

```
2018/11/21 14:03:18 DEBUG [com.ossez.lang.tutorial.tests.VariableTest] - s1/s2/s3 - [1]/[2]/[3]
2018/11/21 14:03:18 DEBUG [com.ossez.lang.tutorial.tests.VariableTest] - s1/s2/s3 - [1]/[2]/[6]
2018/11/21 14:03:18 DEBUG [com.ossez.lang.tutorial.tests.VariableTest] - s1/s2/s3 - [4]/[5]/[6]
```

在这个代码中，在对objB 类实例化的时候，我们修改了静态变量 s3 的值。

因为静态变量是类变量，因此在objB 对象实例化的时候，前期实例化的对象objA 中的 s3 的值也一同再次实例化了。

从这个例子我们可以看出，静态变量在对象实例化的时候如果修改了值，那么这个类已经实例化的所有对象都会被修改。

构造方法

在我们讨论类的时候，一个重要的概念就是有关构造方法。每一个类都会有一个构造方法。

尽管你没有显式的定义构造方法，Java 的编译器也会默认为该类创建一个构造方法。

在每一个新对象创建的时候，最少都会调用一次构造方法。构造方法的名字必须与类的名字相同，在一个类中，可以允许有多个构造方法。

下面是有关构造方法的的示例：

示例

```
public class VariableTest extends TestCase {

        private final static Logger logger = LoggerFactory.getLogger(VariableTest.class);

        public VariableTest() {
        }

        public VariableTest(String name) {
                //  name
        }
}
```

Java 同时还支持单例模式，这种模式能够让你针对一个类只能创建一个实例。

在单例模式中有 2 种类型的的构造方法，有关单例模式的 2 种构造方法，请参考 单例模式页面中的内容。

创建一个对象

如我们上面提到的，一个类为对象提供了蓝图。因此基本上来说一个对象的创建是从类来创建的。在 Java 中关键字 new 被用来创建一个新的对象。

从一个类中来创建一个对象有下面 3 个步骤：

- 申明（Declaration）声明一个对象，包括对象名称和对象类型。
- 实例化（Instantiation） new 关键字被用来创建这个对象。
- 初始化（Initialization）new 关键字被用来创建对象的时候，同时也会调用构造方法来对对象进行初始化。

下面吃创建对象的一个示例：

示例（GitHub）

```
package com.ossez.lang.tutorial.objplusclass;

/**
 *
 * @author YuCheng
 *
 */
public class CreateObject {
        public CreateObject(String name) {
                // This constructor has one parameter, name
                System.out.println(": " + name);
        }

        public static void main(String[] args) {
                // Following statement would create an object myPuppy
                CreateObject myPuppy = new CreateObject("Tomcat");
        }
}
```

我们将上面的程序编译后输出的结果如下

运行输出

小狗的名字是: Tomcat

访问实例变量和方法

实例变量和方法可以通过创建对象来访问。访问实例变量请参考下面完整的路径：

```
//
ObjectReference = new Constructor();

//
ObjectReference.variableName;

//
ObjectReference.methodName();
```

示例

下面的示例展示了如何访问一个类的实例变量和方法。

```
public class Puppy {
    int puppyAge;

    public Puppy(String name) {
        // , name.
        System.out.println("Name chosen is :" + name);
    }

    public void setAge(int age) {
        puppyAge = age;
    }

    public int getAge() {
        System.out.println("Puppy's age is :" + puppyAge);
        return puppyAge;
    }

    public static void main(String[] args) {
        /*  */
        Puppy myPuppy = new Puppy("tommy");

        /*  puppy  */
        myPuppy.setAge(2);

        /*  puppy  */
        myPuppy.getAge();

        /*  */
        System.out.println("Variable Value :" + myPuppy.puppyAge);
    }
}
```

如果我们对上面的代码进行编译后运行的结果如下：

输出

```
Name chosen is :tommy
Puppy's age is :2
Variable Value :2
```

源文件申明规则

在本节的最后部分，我们将学习源文件的声明规则。当在一个源文件中定义多个类，并且还有import语句和package语句时，要特别注意这些规则。

- 一个源文件中只能有一个public类
- 一个源文件可以有多个非public类
- 源文件的名称应该和public类的类名保持一致。例如：源文件中public类的类名是Employee，那么源文件应该命名为Employee.java。
- 如果一个类定义在某个包中，那么package语句应该在源文件的首行。
- 如果源文件包含import语句，那么应该放在package语句和类定义之间。如果没有package语句，那么import语句应该在源文件中最前面。
- import语句和package语句对源文件中定义的所有类都有效。在同一源文件中，不能给不同的类不同的包声明。

类有若干种访问级别，并且类也分不同的类型：抽象类和final类等。这些将在访问控制章节介绍。

除了上面提到的几种类型，Java还有一些特殊的类，如：内部类、匿名类。

Java 包

包主要用来对类和接口进行分类。当开发Java程序时，可能编写成百上千的类，因此很有必要对类和接口进行分类。

Import 语句

在Java中，如果给出一个完整的限定名，包括包名、类名，那么 Java 编译器就可以很容易地定位到源代码或者类。

Import 语句总是给编译器如何找到需要的类的正确路径。

例如，import java.io.*; 的包的引入命令行将会让编译器载入 java_installation/java/io 路径下的所有类

一个简单的例子

在下面的例子中，我们创建两个类：**Employee**和**EmployeeTest。**

首先你可以打开你习惯使用的记事本，然后输入下面的代码。请注意 Employee 类是一个 public 类。然后将你输入的源代码保存为 Employee.java 文件。

Employee 类有 4 个实体变量：name, age, designation 和 salary。这个类只有一个确定的构造函数，这个构造函数被用来对参数进行初始化。

示例

```
import java.io.*;
public class Employee {

    String name;
    int age;
    String designation;
    double salary;

    // This is the constructor of the class Employee
    public Employee(String name) {
        this.name = name;
    }

    // Assign the age of the Employee  to the variable age.
    public void empAge(int empAge) {
        age = empAge;
    }

    /* Assign the designation to the variable designation.*/
    public void empDesignation(String empDesig) {
        designation = empDesig;
    }

    /* Assign the salary to the variable        salary.*/
    public void empSalary(double empSalary) {
        salary = empSalary;
    }

    /* Print the Employee details */
    public void printEmployee() {
        System.out.println("Name:" + name);
        System.out.println("Age:" + age);
        System.out.println("Designation:" + designation);
        System.out.println("Salary:" + salary);
    }
}
```

正如我们前面所提到的，程序的开始运行时通过 main 方法开始的。为了让 Employee 类能够运行，我们需要为我们的程序创建一个 main 方法，这个方法我们将在不同的类中进行创建。

我们需要创建的类的名字为*EmployeeTest*，该类实例化 2 个 Employee 类，并调用方法设置变量的值。

将下面的代码保存为EmployeeTest.java 文件。

```
import java.io.*;
public class EmployeeTest {

    public static void main(String args[]) {
        /* Create two objects using constructor */
        Employee empOne = new Employee("James Smith");
        Employee empTwo = new Employee("Mary Anne");

        // Invoking methods for each object created
        empOne.empAge(26);
        empOne.empDesignation("Senior Software Engineer");
        empOne.empSalary(1000);
        empOne.printEmployee();

        empTwo.empAge(21);
        empTwo.empDesignation("Software Engineer");
        empTwo.empSalary(500);
        empTwo.printEmployee();
    }
}
```

现在你可以对程序进行编译后，运行*EmployeeTest*，你应该能够看到下面的输出：

输出

```
C:\> javac Employee.java
C:\> javac EmployeeTest.java
C:\> java EmployeeTest
Name:James Smith
Age:26
Designation:Senior Software Engineer
Salary:1000.0
Name:Mary Anne
Age:21
Designation:Software Engineer
Salary:500.0
```