Fundamental Concepts of OOP

Adapted from a recommended tutorial on OOP using Java

1. Class
2. Object
3. Inheritance
4. Interface
5. Package

Java Platform

from IBM Developer Works Java Tutorials

Java Language

The Java language's programming paradigm

- –is based on the concept of object-oriented programming (OOP);
- –is a C-language derivative, so its syntax rules look much like C's;
- –is organized into *packages*, within which are classes, within which are methods, variables, constants, and so on.

The Java compiler

When you program for the Java platform, you write JAVA source code that are in *.java* files and then compile them. The compiler checks your code against the language's syntax rules, then writes out *bytecodes* in .class files. Bytecodes are standard instructions targeted to run on a Java virtual machine (JVM), rather than a specific Chipset/

The JVM

The JVM reads and interprets .class files and executes the program's instructions on the native hardware platform for which the JVM was written. The JVM is a piece of software for a particular platform that *interprets* the bytecodes just as a CPU would interpret assembly-language instructions. JVMs are available for Linux and Windows, and subsets of the Java are available in JVMs for mobile phones and hobbyist chips.

[The idea of compiling a source language into byte code for a virtual machine came from the University of San Diego's PASCAL.]

Java Platform

from IBM Developer Works Java Tutorials

Compile and Run a Java Program from a Console window [Command prompt]

CD myProgram - change to directory where your program is located

javac myProgram.java - compiles to myProgram.class in the same directory

java myProgram - executes the myPogram.class bytecode

The garbage collector

When your Java application creates an object instance at run time, the JVM automatically allocates memory space for that object from the *heap*, a pool of memory set aside for your program to use. The Java *garbage collector* runs in the background, keeping track of which objects the application no longer needs and reclaiming memory from them.

The Java Development Kit, (JDK) and The Java Runtime Environment

When you download a Java Development Kit (JDK), you get — in addition to the compiler and other tools — a complete class library of prebuilt utilities that help you accomplish just about any task common to application development. (see Resources). The Java Runtime Environment (JRE;) includes the JVM, code libraries, and components for running Java programs. JAR stands for Java Archive files: it is a single, zipped file, and many Java programs are stored there. You can freely redistribute the JRE with your applications.

What Is a Class?

A class is a blueprint or prototype from which objects are created.

A class models the state and behavior of a objects of the same type.

Class

A class is a blueprint for a type of entity

Objects are instantiated (constructed, created) from the class, using the blueprint

*UML - Unified Modeling Language, a language used in object-oriented design

Class Diagram (UML)*

What Is an Object?

An object is a software bundle of related *state* and *behavior*.

Objects are often used to model real-world objects.

State is represented within an object using private *instance variables.*

Behaviour is represented by an object's *public* methods

Hiding an object's state and allowing other classes to access it's state only through the object's public methods is called *encapsulation.*

Object

When an object is created from the class blue-print, it is a *reference* (points) to an account object.

The account object mirrors the structure of the class blue-print.

An object is a *reference* to an instance of a class

Here is how an object is *constructed:*

Bank Account myAccount = *new BankAccount();*

myAccount ----->

myAccount references the newly instantiated object.

What is inheritance?

Different kinds of objects often have a certain amount in common with each other.

Mountain bikes, road bikes, and tandem bikes, for example, all share the characteristics of bicycles (current speed, current pedal cadence, current gear).

Yet each also defines additional features that make them different:

- tandem bicycles have two seats and two sets of handlebars;
- road bikes have drop handlebars;
- some mountain bikes have an additional chain ring, giving them a lower gear ratio.

What is Inheritance?

Different kinds of objects often have a certain amount in common with each other.

Mountain bikes, road bikes, and tandem bikes, for example, all share the characteristics of bicycles (current speed, current pedal cadence, current gear).

Yet each also defines additional features that make them different:

- tandem bicycles have two seats and two sets of handlebars;
- road bikes have drop handlebars;
- some mountain bikes have an additional chain ring, giving them a lower gear ratio.

Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.

In this example, Bicycle becomes the superclass of MountainBike, RoadBike, and TandemBike.

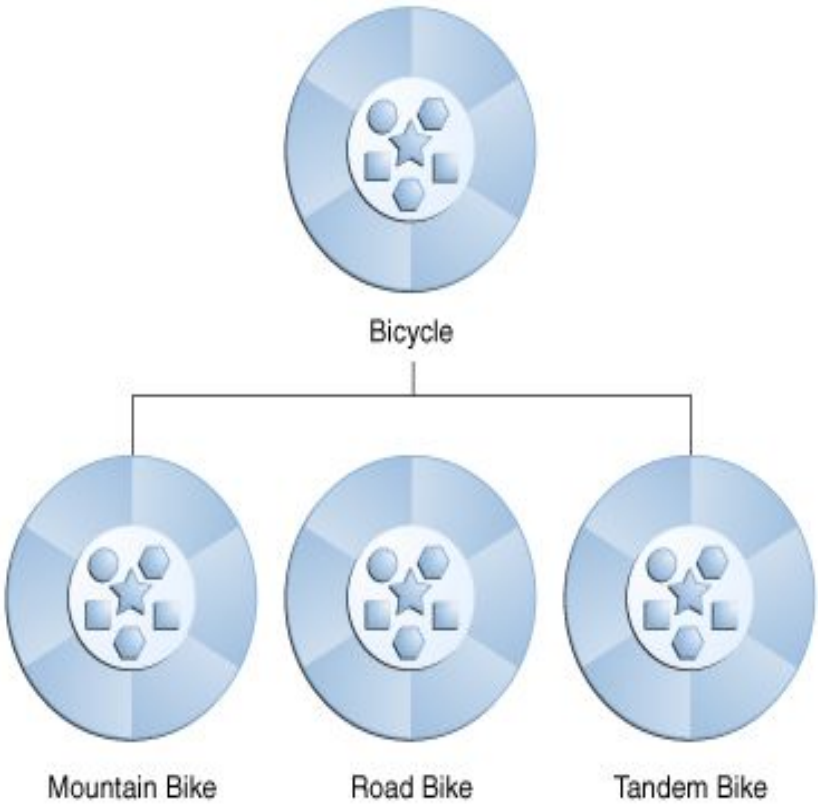
Each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of subclasses.

How inheritance is expressed

```
class MountainBike extends Bicycle {

    // new fields and methods defining a
mountain bike go here

}
```



What Is an Interface?

An interface is a contract between a class and the outside world. When a class *implements* an interface, it promises to provide the behavior published by that interface

Objects define their interaction with the outside world through their *public* methods - they are the object's interface with the outside world.

In the real world, the buttons on the front of your television set are the interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the television on and off. You don't care how this happens as long as it does.

Declaring an interface

In its most common form, an *interface* is a group of related methods with *empty bodies*.

Here is a bicycle's behavior, specified as an interface:

```
public interface Bicycle {

    void changeCadence(int newValue);      //cadence is the pedalling rate

    void changeGear(int newValue);

    void speedUp(int increment);

    void applyBrakes(int decrement);
}
```

Implementing an Interface

We can implement this interface, using a particular brand of bicycle, for example, an ACMEBicycle:

```
class ACMEBicycle implements Bicycle {

        int cadence = 0;
        int speed = 0;
        int gear = 1;

        public void changeCadence(int newValue) {
             cadence = newValue;    }
        public void changeGear(int newValue) {
              gear = newValue;    }
        public void speedUp(int increment) {
              speed = speed + increment;       }
        public void applyBrakes(int decrement) {
              speed = speed - decrement;    }

        public void printStates() {
             System.out.println("cadence:" +  cadence + " speed:" +  speed + " gear:" +
           gear);    }
}
```

All methods defined by that interface must appear in a class that implements the interface before the class will successfully compile.

What is a Package?

A package is a *name space* that organizes a set of related classes and interfaces.

The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications.

This library is known as the "Application Programming Interface", or "API" for short.

The Java API

The Java Platform API Specification
contains the complete listing for all packages, interfaces, classes, fields, and methods supplied by the Java SE platform.

Bookmark this link!

It is a programmer's single most important piece of reference documentation for Java.