Chapter 10: Designing classes

Object-oriented programming

- What is Object-oriented programming?
  - Object-oriented programming (OOP) is a programming paradigm using "objects" – usually instances of a class – consisting of data fields (attributes) and methods (behaviors), together with their interactions – to design applications and computer programs.

(Source: Wikipedia (http://en.wikipedia.org/wiki/Object-oriented_programming)

Recap: classes and objects

- A class is a blueprint or prototype from which objects are created.
  - Class header
  - Class body
- An object is an instance of a class.
- An object is a software bundle of related behavior (methods) and state (attributes).
- Ask yourself two questions:
  - What possible states (fields) can this object be in?
  - What possible behavior (methods) can this object perform?

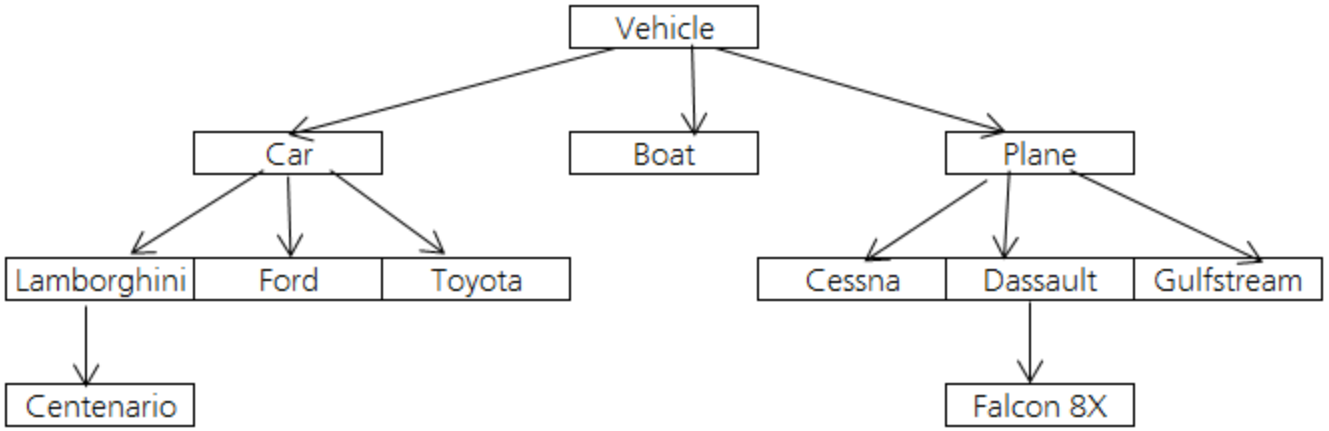
Object-oriented Concepts

- Encapsulation
  - Information hiding. The internal representation of an object is generally hidden from view outside of the object's definition.
- Inheritance
  - A way to reuse code of existing objects or to establish a subtype from an existing object. *See next slide.*
- Polymorphism
  - (in Greek, means many forms) The ability to create a variable, a function, or an object that has more than one form.

(Source: Wikipedia (http://en.wikipedia.org/wiki/Object-oriented_programming)

Inheritance

The "family tree" of classes, also called the class hierarchy.



Inheritance / hiearchy

```
public class Lamborghini extends Car {
// inherits the attributes/state and behavior of the parent 'Car'
class


    // additional attribute:
    double groundClearance; // how high above the ground is it?

    // additional method, to change ground clearance:
    public void setGroundClearance(double newGC) {
        groundClearance = newGC;
    }
}
```

The class hierarchy

- In Java, all classes extend some other class.
- The most basic class is called **Object**.
  - Contains no instance variables.
  - Provides the methods `equals()` and `toString()`, among others.
- Any class that does not explicitly name a parent, inherits from **Object** by default.

Java programs

- Java programs are object-oriented, which means that the focus is on objects and their interactions.
  - Objects often represent entities in the real world, e.g., Car class
  - The majority of methods are object methods (like the methods you invoke on Strings) rather than class methods (like the Math methods).
  - Objects are isolated from each other by limiting the ways they interact, especially by preventing them from accessing instance variables without invoking methods.
  - Classes are organized in family trees where new classes extend existing classes, adding new methods and replacing others.

The toString() method

- Every object type has a method called *toString* that returns a string representation of the object.
- When you print an object using *print* or *println*, Java invokes the object's toString method.
- The default version of toString returns a string that contains the type of the object and a unique identifier.

```
Color color = Color.RED;
System.out.println("Color is: " + color); // Color is: java.awt.Color[r=255,g=0,b=0]
```

The toString method

When you define a new object type, you can override the default behavior by providing a new method with the behavior you want.

```
Car car = new Car();
System.out.println(car);
// Might print something like: Test$Car@135fbaa4

// Now implement the toString() method for the Car class:
public String toString() {
    return ("I am a " + color + " car!");
}
System.out.println(car); // assume 'color' is set to "red"
// Prints: I am a red car!
```

The equals method

- Review two notions of equality:
  - identity: two variables refer to the same object
    - The == operator tests identity.
  - equivalence: they have the same value
    - no **operator** tests equivalence because what "equivalence" means depends on the type of the objects.
    - Java classes provide *equals* methods that define equivalence.

Summary & Exercises

Appendix

Naming Conventions

http://en.wikipedia.org/wiki/Naming_convention_(programming)#Java

Naming Conventions - Classes

- Class names should be nouns in Upper CamelCase, with the first letter of every word capitalized.
- Use whole words — avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).

```
class Bicycle {
}
```

Naming Conventions - Methods

- Methods should be verbs in lower CamelCase; that is, with the first letter lowercase and the first letters of subsequent words in uppercase.

```
public int getGear() {
}
```

Naming Conventions - Variables

- Local variables, instance variables, and class variables are also written in lower CamelCase.
- Variable names should not start with underscore (_) or dollar sign ($) characters, even though both are allowed.
- Certain coding conventions state that underscores should be used to prefix all instance variables, for improved reading and program understanding.

```
public int gear;
```

Naming Conventions - Variables

- Variable names should be short yet meaningful.
- The choice of a variable name should be mnemonic — that is, designed to indicate to the casual observer the intent of its use.
- The convention is to always begin the variable names with a letter, not "$" or "_".
- Subsequent characters may be letters, digits, dollar signs, or underscore characters.
- The name must not be a keyword or reserved word.
- One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.

```
public int speed;
```

Naming Conventions - Constants

- Constants should be written in uppercase characters separated by underscores. Constant names may also contain digits if appropriate, but not as the first character.

```
public final static int MAXIMUM_NUM_OF_SPEEDS =
10;
```

Access Modifiers

- Two levels of access control:
  - At the top level: public, or package-private(no explicit modifier).
  - At the member level: public, private, protected, or package-private(no explicit modifier).
- public: accessible from all classes
- private: accessible only within its own class
- protected: accessible from within its own package and by a subclass of its class in another package

Access Level

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
No modifier	Y	Y	N	N
private	Y	N	N	N

Choosing an Access Level

- Use the most restrictive access level that makes sense for a particular member.
- Use private unless you have a good reason not to.
- Avoid public fields except for constants.
- Public fields tend to link you to a particular implementation and limit your flexibility in changing your code.

Inheritance

- Inheritance is the ability to define a new class that is a modified version of an existing class.
- The existing class is called the **parent** class and the new class is called the **child**.
- Advantage: you can add methods and instance variables without modifying the parent.

Inheritance

- Inheritance is a powerful feature.
- Some programs that would be complicated without it can be written concisely and simple with it.
- Inheritance can facilitate code reuse, since you can customize the behavior of existing classes without having to modify them.

Object methods and class methods

- Class methods are identified by the keyword static in the first line.
- Any method that does not have the keyword static is an object method.
- Whenever you invoke a method “on” an object, it’s an object method, String.charAt.
- Anything that can be written as a class method can also be written as an object method, and vice versa.

Object methods and class methods

- printCard as a class method

```
public static void printCard(Card c) {
  System.out.println("This card is " + c.type);
}
```

- print as an object method

```
public void print() {
  System.out.println("This card is " + type);
}
```

Bicycle in action

- Bicycle
- MountainBike

Interfaces

- 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.
- An interface declaration consists of modifiers, the keyword interface, the interface name, a comma-separated list of parent interfaces (if any), and the interface body.

Interfaces

- Interface can contain only constants, method signatures, and nested types. There are no method bodies.
- All methods declared in an interface are implicitly **public**.
- All constants defined in an interface are implicitly **public**, **static**, and **final**.
- Interface cannot be instantiated—they can only be implemented by classes or extended by other interfaces.

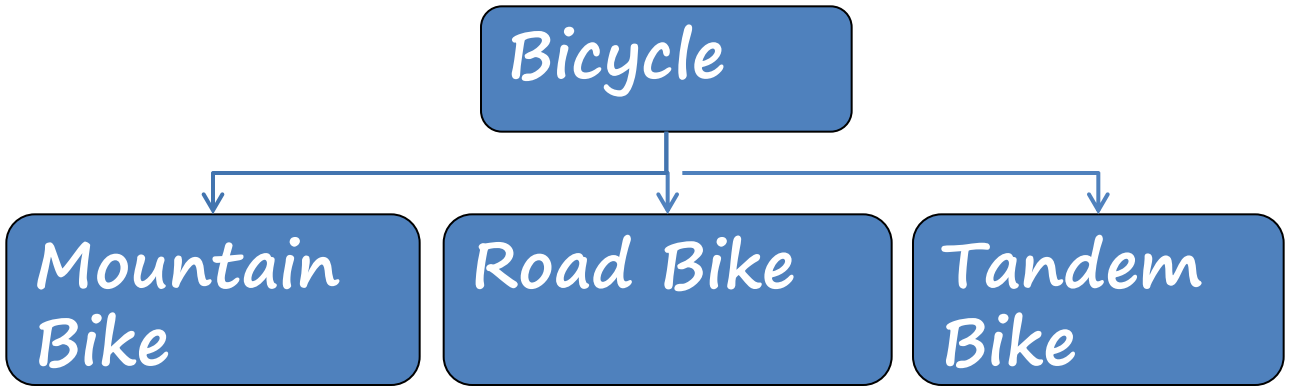
Define Interface

```
public interface Bicycle {
    // constant
    int MAX_GEARS = 20;

    //  wheel revolutions per minute
    void changeGear(int newValue);
    void speedUp(int increment);
    void applyBrakes(int decrement);
    void changeCadence(int newValue);
}
```

Implements Interface

```
public interface Bicycle {
    void changeGear(int newValue);
}
class MountainBike implements Bicycle {
    void changeGear(int newValue)
        gear = newValue;
    }
}
```



Rewriting Interfaces

- Developed An Interface

```
Public interface DoIt {
    void doSomething(int i);
}
```

- Want to Add A Method

```
Public interface DoIt {
    void doSomething(int i);
    int doSomethingElse(String s);
}
```

- Add A Method

```
Public interface DoItPlus extends DoIt {
    boolean doSomethingElse(String s);
}
```

Interfaces & Multiple Inheritance

- The Java programming language does not permit multiple inheritance, but interfaces provide an alternative.
- In Java, a class can inherit from only one class but it can implement more than one interface. Therefore, objects can have multiple types: the type of their own class and the types of all the interfaces that they implement. This means that if a variable is declared to be the type of an interface, its value can reference any object that is instantiated from any class that implements the interface.

Multiple Interface

```
public interface GroupedInterface extends
Interface1, Interface2, Interface3 {
    // constant declarations
    // base of natural logarithms
    double E = 2.718282;

    // method signatures
    void doSomething (int i, double x);
    int doSomethingElse(String s);
}
```

Summary of Interfaces

- A protocol of communication between two objects
- Contains signatures and constant but not method implementation
- A class implements an interface must implement all methods
- An interface name can be used anywhere a type can be used

What do you mean by static in Java?

- When a number of objects are created from the same class blueprint, they each have their own distinct copies of ***instance (object) variables*** and ***instance (object) methods***.

What do you mean by static in Java? (continued)

- Sometimes, you want to have variables and methods that are common to all objects.
  - This is accomplished with the **static** modifier.
    - static fields or ***class variables***
    - static methods or ***class methods***
- They are associated with the class, rather than with any object.
- Every instance of the class shares a class variable, which is in one fixed location in memory.
- Any object can change the value of a class variable.

What do you mean by static in Java? (continued)

- The static keyword can be used in:
  - static variables
  - static methods
  - Constants
  - static blocks of code

static variable

- Belongs to the class and not to object (instance)
- Is initialized only once, at the start of the execution.
- A single copy is shared by all instances of the class.
- Is accessed directly by the class name and doesn't need any object
  - ClassName.variableName
  - **`Bicycle.numberOfBicycles`**

static method

- Belongs to the class and not to the object (instance)
- Can access only static data. It can not access non-static data (instance variables)
- Can call only other static methods and can not call a non-static method from it.
- Can be accessed directly by the class name and doesn't need any object.
- Can not refer to "**this**" or "**super**" keywords in anyway.
  - ClassName.methodName(args)
  - Bicycle.getNumberOfBicycles()

Instance and class variables and methods

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access class variables and class methods directly.
- Class methods can access class variables and class methods directly.
- Class methods cannot access instance variables or instance methods directly—they must use an object reference.
- Class methods **cannot** use the this keyword as there is no instance for **this** to refer to.

Constants

- The **static** modifier, in combination with the **final** modifier, is used to define constants. The **final** modifier indicates that the value of this field cannot change.
- static final double PI = 3. 141592653589793;

static blocks of code

- The static initialization block is a normal block of code enclosed in braces, {}, and preceded by the **static** keyword that will be executed when a class is first loaded into the JVM.

```
static {
  // whatever code is needed for
initialization goes here
}
```