



Object Oriented Programming Concepts

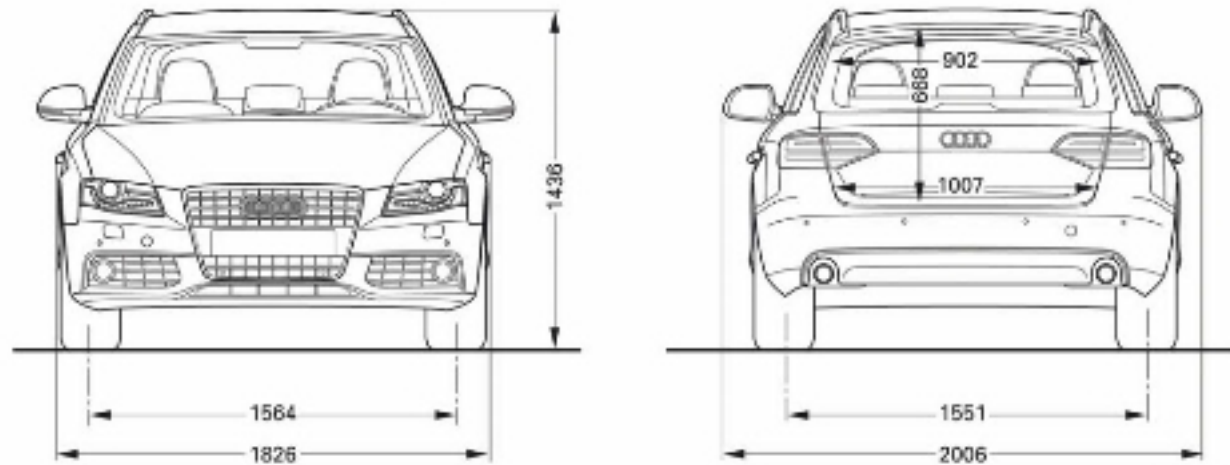
Sassan Maleki
Winter 2009

What is an Object?

- A software bundle of related
 - States
 - Behaviors
- Used to model real-world objects
 - e.g. car, book, human

What is a Class?

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



```
class Bicycle {  
  
    int cadence = 0;  
    int speed = 0;  
    int gear = 1;  
  
    void changeCadence(int newValue) {  
        cadence = newValue;  
    }  
  
    void changeGear(int newValue) {  
        gear = newValue;  
    }  
  
    void speedUp(int increment) {  
        speed = speed + increment;  
    }  
  
    void applyBrakes(int decrement) {  
        speed = speed - decrement;  
    }  
  
    void printStates() {  
        System.out.println("cadence:" + cadence + " speed:" + speed + " gear:" + gear);  
    }  
}
```

What is a Constructor?

- Constructors are invoked to create objects from the class blueprint

```
public Bicycle(int startCadence, int startSpeed,  
int startGear) {  
    gear = startGear;  
    cadence = startCadence;  
    speed = startSpeed;  
}
```

```
Bicycle myBike = new Bicycle(30, 0, 8);
```

Abstract Methods and Classes

- An *abstract class* is a class that is declared *abstract*
 - It may or may not include abstract methods.
 - Cannot be instantiated, but can be subclassed.
- An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract void moveTo(double deltaX, double deltaY);
```

What is Inheritance?

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

```
class MountainBike extends Bicycle {  
  
    // new fields and methods defining a mountain  
    bike would go here  
  
}
```

What is an Interface?

- Objects define their interaction with the outside world through the methods that they expose
- Methods form the object's interface with the outside world
 - e.g. 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
- Cannot be instantiated

```
interface Bicycle {  
  
    void changeCadence(int newValue);  
  
    void changeGear(int newValue);  
  
    void speedUp(int increment);  
  
    void applyBrakes(int decrement);  
  
}
```

```
class ACMEBicycle implements Bicycle {  
  
    // remainder of this class implemented as before  
  
}
```

Abstract Classes vs Interfaces

- Abstract classes can contain implemented methods
- Abstract classes are similar to interfaces, except that they provide a partial implementation, leaving it to subclasses to complete the implementation

What is a Package?

- A namespace that organizes a set of related classes and interfaces
- Similar to different folders on your computer

What is an Exception?

- An event that occurs during the execution of a program that disrupts the normal flow of instructions.

```
public void writeList() {
    PrintWriter out = null;

    try {
        System.out.println("Entering try statement");
        out = new PrintWriter(
            new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++)
            out.println("Value at: " + i + " = "
                + vector.elementAt(i));

    } catch (ArrayIndexOutOfBoundsException e) {
        System.err.println("Caught "
            + "ArrayIndexOutOfBoundsException: "
            + e.getMessage());

    } catch (IOException e) {
        System.err.println("Caught IOException: "
            + e.getMessage());

    } finally {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();

        }
        else {
            System.out.println("PrintWriter not open");
        }
    }
}
```

```
}
```

```
public class MyException extends Exception{  
    public MyException(String msg){  
        super(msg);  
    }  
  
    public MyException(String msg, Throwable t){  
        super(msg, t);  
    }  
}
```

```
public static void example(Object input) throws MyException{  
    if(input != valid){  
        throw new MyException("You didn't give me the info I wanted!");  
    }  
}
```

```
public static void example(Object input){  
    if(input != whatsExpected)  
        throw new Exception();  
}
```

Field Encapsulation

Setter and Getter Methods

- Putting restrictions on access to fields from outside of the class
- Steps:
 1. Make the field private (e.g private name)
 2. Create a *setter* method for allowing the field to be set to a value (e.g setName)
 3. Create a *getter* method for returning the value of the field (e.g getName)

Polymorphism

- Defining methods that take other objects as parameters. One can get more cooperation and efficiency when the objects are united by a common superclass.
- If you define a method *examine* that takes an object *man* as a parameter it can only accept a man however if you define it to take a *human* it can accept both *man* and *woman* objects.