

# Advanced Programming-- Java

The Object Class, Abstract Classes and Interfaces



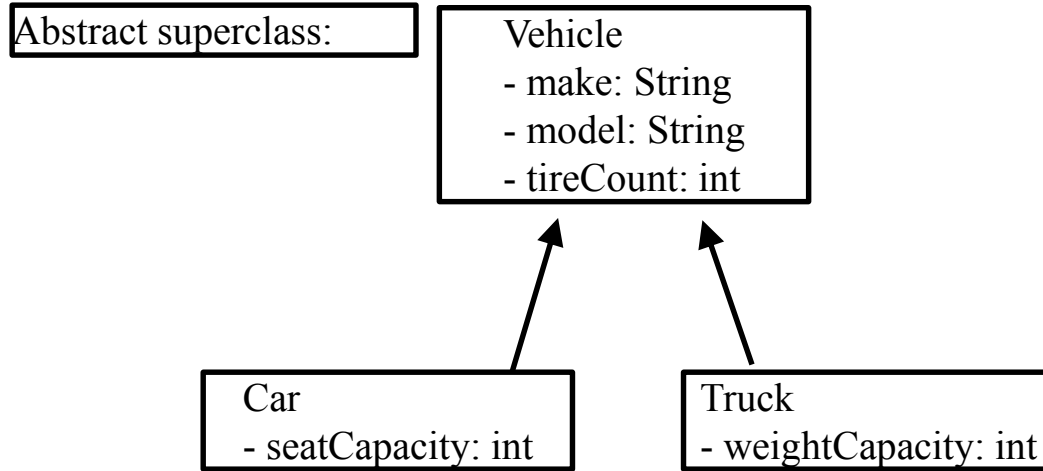
# The Java Object Class

- Sits at the top of Java development environment.
- Every class is directly or indirectly a descendant of the Object class.
- When a class does not extend any other class, then it is a subclass of the object class.
- Defines the basic state and behavior of objects.

# What is an Abstract class?

- Superclasses are created through the process called "generalization"
  - Common features (methods or variables) are factored out of object classifications (i.e. classes).
  - Those features are formalized in a class. This becomes the superclass
  - The classes from which the common features were taken become subclasses to the newly created super class
- Often, the superclass does not have a "meaning" or does not directly relate to a "thing" in the real world
- Because of this, abstract classes cannot be instantiated

# Abstract Class Example



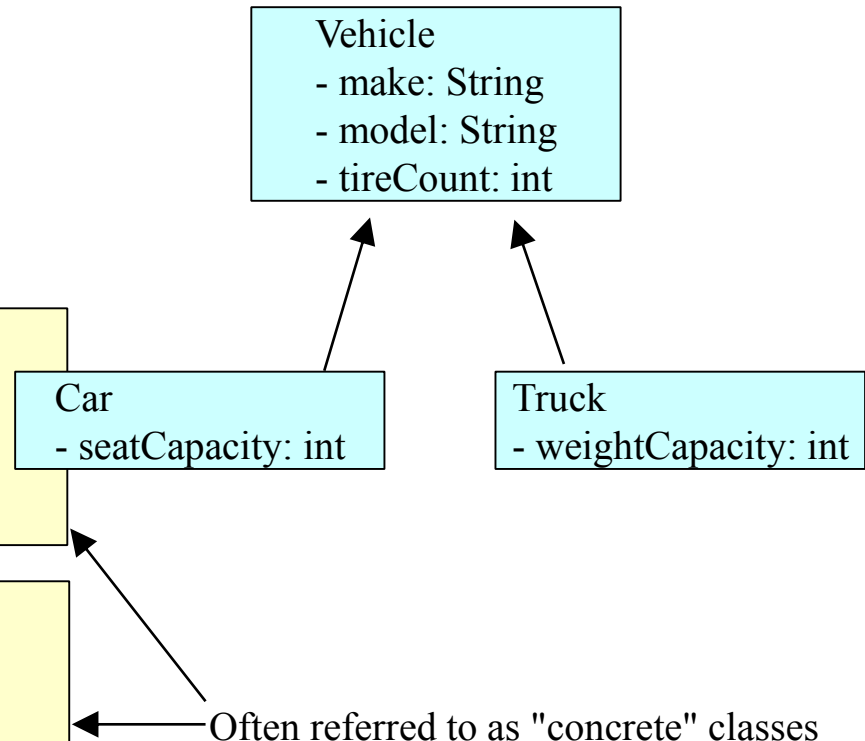
# Defining Abstract Classes

- Inheritance is declared using the "extends" keyword
  - If inheritance is not defined, the class extends a class called Object

```
public abstract class Vehicle
{
    private String make;
    private String model;
    private int tireCount;
    [...]
```

```
public class Car extends Vehicle
{
    private int weightCapacity;
    [...]
```

```
public class Truck extends Vehicle
{
    private int seatCapacity;
    [...]
```



# Abstract Methods

- **Methods can also be abstracted**
  - An abstract method is one to which a signature has been provided, but no implementation for that method is given.
  - An Abstract method is a placeholder. It means that we declare that a method must exist, but there is no meaningful implementation for that methods within this class
- **Any class which contains an abstract method MUST also be abstract**
  - Any class which has an incomplete method definition cannot be instantiated (i.e. it is abstract)
- **Abstract classes can contain both concrete and abstract methods.**
  - If a method can be implemented within an abstract class, and implementation should be provided.

# What is an Interface?

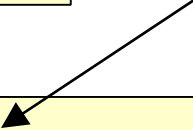
- An interface is similar to an abstract class with the following exceptions:
  - All methods defined in an interface are abstract. Interfaces can contain no implementation
  - Interfaces cannot contain instance variables. However, they can contain public static final variables (i.e. constant class variables)
- Interfaces are declared using the "interface" keyword
  - If an interface is public, it must be contained in a file which has the same name.
- Interfaces are implemented by classes using the "implements" keyword.

# Declaring an Interface

In Steerable.java:

```
public interface Steerable
{
    public void turnLeft(int degrees);
    public void turnRight(int degrees);
}
```

When a class "implements" an interface, the compiler ensures that it provides an implementation for all methods defined within the interface.



In Car.java:

```
public class Car extends Vehicle implements Steerable
{
    public int turnLeft(int degrees)
    {
        [...]
    }

    public int turnRight(int degrees)
    {
        [...]
    }
}
```



# Implementing Interfaces

- A Class can only inherit from one superclass. However, a class may implement several Interfaces
  - The interfaces that a class implements are separated by commas
- Any class which implements an interface must provide an implementation for all methods defined within the interface.
  - NOTE: if an abstract class implements an interface, it NEED NOT implement all methods defined in the interface. HOWEVER, each concrete subclass MUST implement the methods defined in the interface.

# Declaring an Interface

In Car.java:

```
public class Car extends Vehicle implements Steerable, Driveable
{
    public int turnLeft(int degrees)
    {
        [...]
    }

    public int turnRight(int degrees)
    {
        [...]
    }

    // implement methods defined within the Driveable interface
```