# CI6206
# Internet Programming

Abstract Classes and Interfaces

# What is an Abstract class?

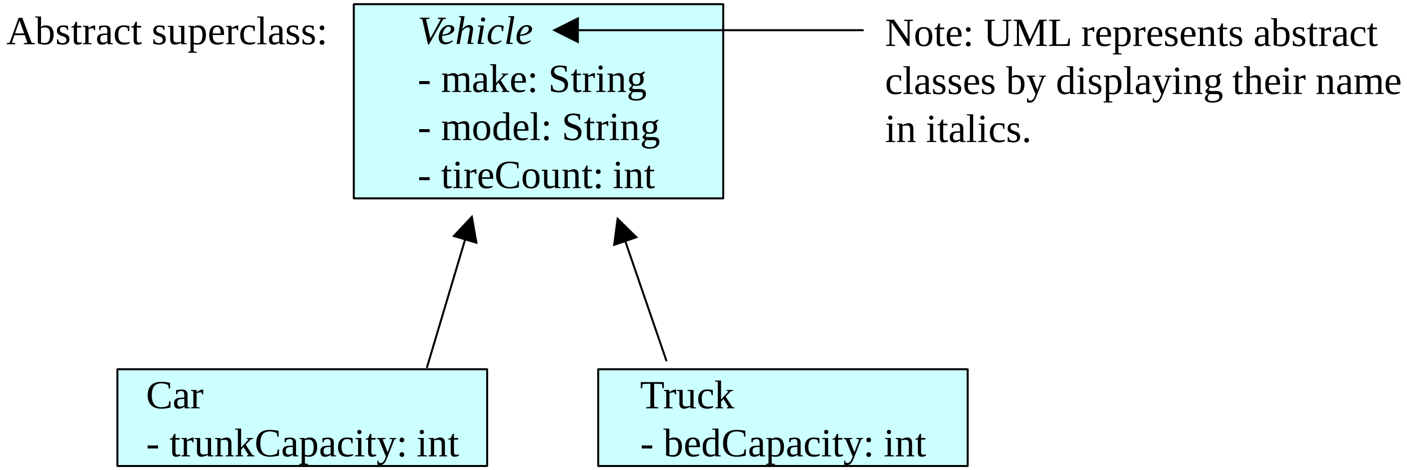
- Superclasses are created through the process called "generalization"
  - Common features (methods or variables) are factored out of object classifications (ie. classes).
  - The classes from which the common features were taken become subclasses to the newly created super class

- Often, the superclass does not have a "meaning" or does not directly relate to a "thing" in the real world
  - It is an artifact of the generalization process

- Because of this, abstract classes cannot be instantiated
  - They act as place holders for abstraction

# Abstract Class Example

- In the following example, the subclasses represent objects taken from the problem domain.
- The superclass represents an abstract concept that does not exist "as is" in the real world.



Abstract superclass:

*Vehicle*
- make: String
- model: String
- tireCount: int

Note: UML represents abstract classes by displaying their name in italics.

Car
- trunkCapacity: int

Truck
- bedCapacity: int

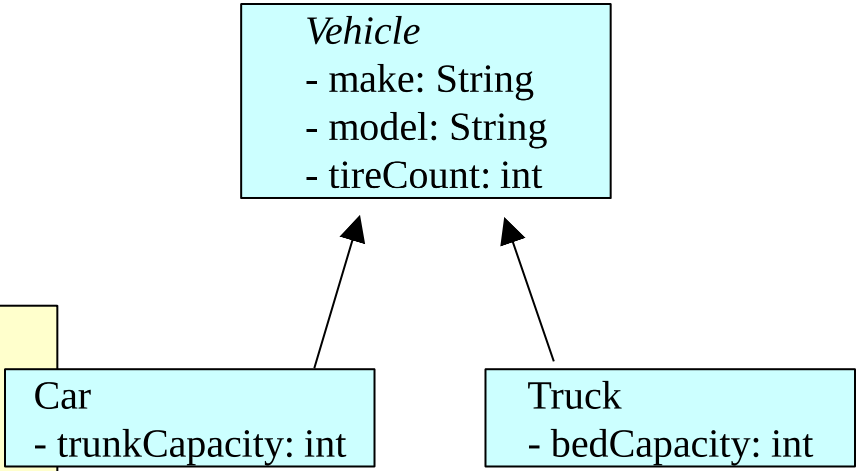
# Defining Abstract Classes

- 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 trunkCapacity;
    [...]
```

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



Often referred to as "concrete" classes

# Abstract Methods

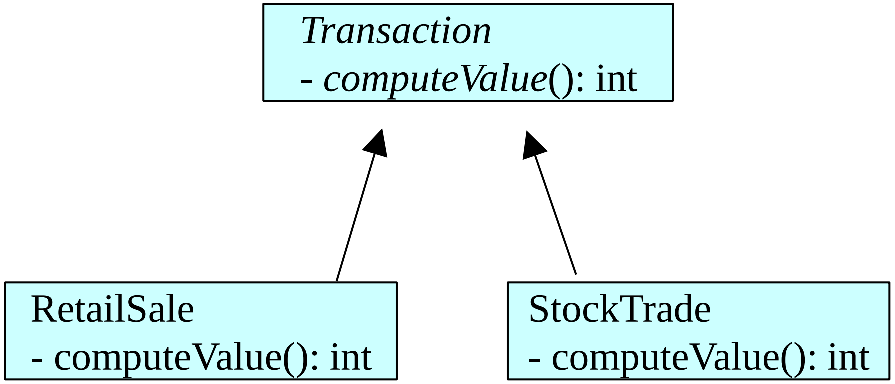
- Methods can also be abstracted
  - An abstract method is one to which a signature has been provided, but no implementation for that method is given.
  - An Abstract method is a placeholder. It means that we declare that a method must exist, but there is no meaningful implementation for that methods 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 (ie. 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.

# Abstract Method Example

- In the following example, a Transaction's value can be computed, but there is no meaningful implementation that can be defined within the Transaction class.
  - How a transaction is computed is dependent on the transaction's type
  - *Note: This is polymorphism.*



*Transaction*
- *computeValue*(): int

RetailSale
- computeValue(): int

StockTrade
- computeValue(): int

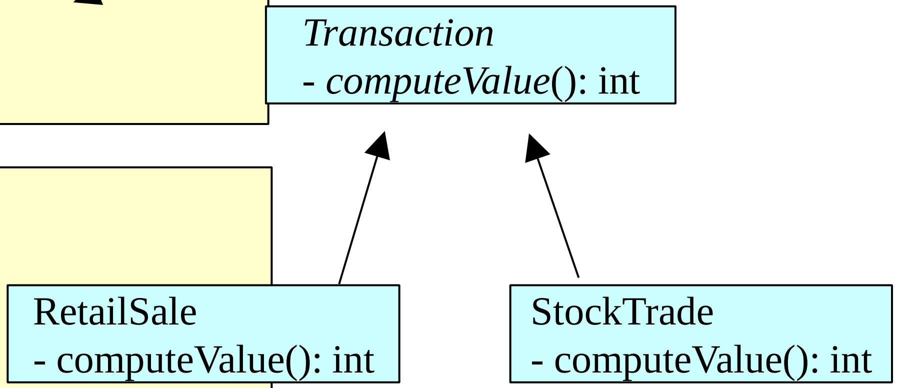
# Defining Abstract Methods

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

```
public abstract class Transaction
{
    public abstract int computeValue();
```

Note: no implementation

```
public class RetailSale extends Transaction
{
    public int computeValue()
    {
            [...]
```



```
public class StockTrade extends Transaction
{
    public int computeValue()
    {
            [...]
```

# 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 (ie. constant class variables)

- Interfaces are more abstract than abstract classes

- 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.
- Interfaces can inherit method signatures from other interfaces.

# 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
```

# Java Servelets

- A servlet is an instance of a class that implements the java.servlet.Servlet interface.
- The **javax.servlet** and **javax.servlet.http** packages provide interfaces and classes for writing servlets.
- All servlets must implement the **Servlet** interface, which defines life-cycle methods.
- When implementing a generic service, you can use or extend the **GenericServlet** class provided with the Java Servlet API.
- The **HttpServlet** class provides methods, such as doGet and doPost, for handling HTTP-specific services.

- **javax.servlet.http.HttpServlet**
  Signature: public abstract class HttpServlet extends GenericServlet implements java.io.Serializable
- HttpServlet defines a HTTP protocol specific servlet.
- HttpServlet gives a blueprint for Http servlet and makes writing them easier.
- HttpServlet extends the GenericServlet and hence inherits the properties GenericServlet.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SomeServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
// Use "request" to read incoming HTTP headers (e.g. cookies)
// and HTML form data (e.g. data the user entered and submitted)
// Use "response" to specify the HTTP response line and headers
// (e.g. specifying the content type, setting cookies).
 PrintWriter out = response.getWriter();
// Use "out" to send content to browser } }
```

- http://docs.oracle.com/javaee/1.3/api/javax/servlet/http/HttpServlet.html