Module Extra

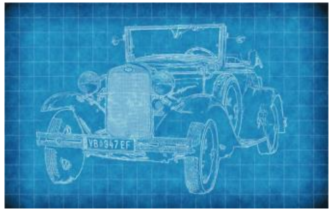
Introduction to Classes and Objects

Objectives

- Should be able to explain the concept of classes and their use in Object Oriented Programming
- Should be able to create a proper class definition
- Should be able to effectively use class variables, methods and properties
- Should be able to create and call Constructors
- Should be able to describe the difference with public and private access modifiers
- Should be able to create instances of a class
- Should be able to explain the concept of properties and correctly define and use them in a class
- Should be able to define member methods in a class and use them in an object of that class
- Should be able to state the concept of overloading as it pertains to classes

Classes

Classes are blueprints to create objects.



A **class** is like a blueprint, it is not the thing you’re building, but it describes what you’re building

From a class, we can create as many objects of that class we need.

Objects: Properties and Methods

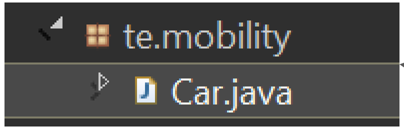
Objects have properties (also called members, or data members) and methods.

Consider these vehicles, they were all created from the same blueprint. The blueprint specifies that each vehicle should have a color, **color is therefore a property of the object**.

Objects also have methods. Again, consider some of the things a vehicle can or needs to do: start the engine, go in reverse, check how much fuel it has left. **These are examples of methods a vehicle object might have.**

Class Declaration

Here are the basics on how to declare a custom class:



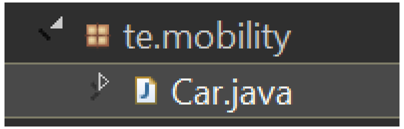
```
package te.mobility;

public class Car {
        // most basic class definition.
}
```

On your file system, the name of your class must match the name of the file and the specified package.

Classes contain curly braces. All class related code will be enclosed within them.

Class Declaration: Packages

Packages are Java's ways to organize classes. In the file system they will correspond to folder names:

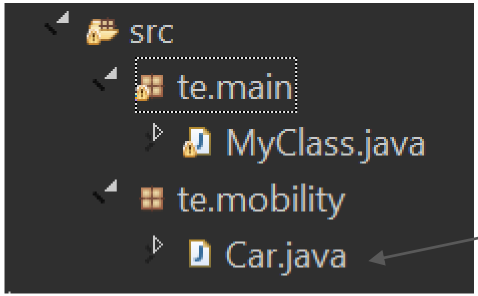


```
package te.mobility;
...
```

```
$ pwd
/c/Users/Student/workspace/JohnSmith/FirstJavaApp/src/te/mobility
```

Class Declaration: Packages & Access Modifiers

Consider the following Java application, note that there are 2 packages: **te.mobility** and **te.main**.



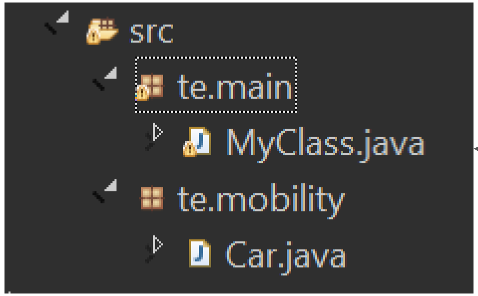
```
package te.mobility;

public class Car {
        // most basic class definition.
}
```

The **Car** class is part of the te.mobility package.

Note that the class has an access modifier of **public**.

Class Declaration: Packages & Access Modifiers

Because the Car class is public it can be instantiated from any other package in the Java application.



```
package te.main;

import te.mobility.Car;

public class MyClass {

   public static void main(String args[]) {
                 Car thisCar = new Car();
         }
}
```

Note that MyClass is in a different package, yet I'm able to reference the Car class. (We will go over the new Car() syntax in future slides)

Class Declaration: Packages & Access Modifiers

If we change the access modifier from public to default (by not specifying anything), then the Car class becomes invisible to all other packages.

Note that the the class is not public anymore, it's default. There is no **public** keyword!

```
package te.mobility;

class Car {
          // most basic class definition.
}
```

The following lines will now cause errors.

```
package te.main;

import te.mobility.Car;

public class MyClass {

   public static void main(String args[]) {
      Car thisCar = new Car();
   }
}
```

Class Declaration: Packages & Access Modifiers

Classes have two types of access modifiers:

- **public**: The class will be visible to all packages.
  - *public class MyClass {...}*
- **default**: The class is only visible within the package it’s on.
  - *class MyClass {...}*

Note that there is no “default” keyword, default is just not specifying anything!

Data Members: Declaration

Classes have properties or data members. Let’s consider the Car class.

We have declared some properties but not initialized them to anything, they will have default values.

```
class Car {
   private String color;
   private Double engineSize;
   private int numberOfDoors;
}
```

We have declared some properties and initialized them to some values.

```
class Car {
   private String color = "green";
   private double engineSize = 1.5;
   private int numberOfDoors = 2;

}
```

Data Members: Default Values

Data members have access modifiers as well

Data Type	**Default Value**
int	0
double / float	0.0
boolean	false
String	null

Data Members: Access Modifiers

Data members have access modifiers as well

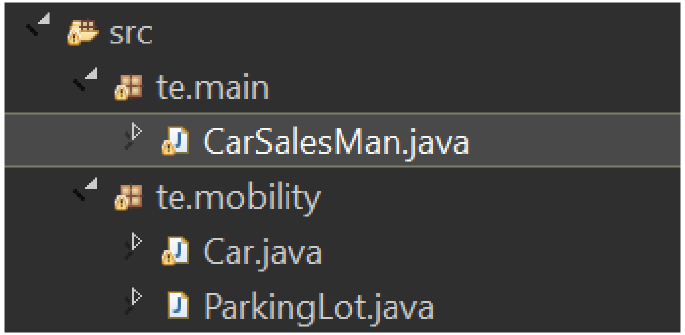
Access Modifier	**Implication**
public	Accessible to any class.
private	Only accessible within the same class.
protected	Default access, but can also be used with sub-classes regardless of package.
default	Accessible to other classes, but must be within the same package.

Data Members: Access Modifiers Example

Consider the following scenario:

- I have 1 class called CarSalesMan in the te.main package.
- I have 2 classes in te.mobility: Car and ParkingLot.

This is what the package structure looks like on the IDE:



```
src
  te.main
    CarSalesMan.java
  te.mobility
    Car.java
    ParkingLot.java
```

This is what the Car class looks like, we will analyze what happens as we toggle the access modifier.

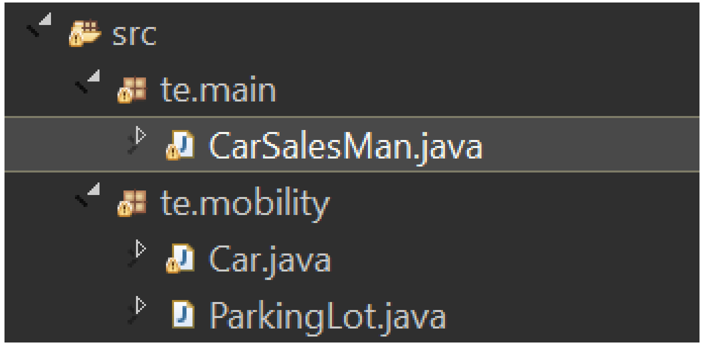
```
package te.mobility;

public class Car {
   private String color = "green";
}
```

Data Members: Access Modifiers Example

If the color data member is **private**:

Car.java

```
package te.mobility;

public class Car {
   private String color = "green";
}
```



```
src
  te.main
    CarSalesMan.java
  te.mobility
    Car.java
    ParkingLot.java
```

CarSalesMan.java

```
// within some method:
Car myCar = new Car();
System.out.println(myCar.color);
// This is an illegal declaration,
// color is private.
```

ParkingLot.java

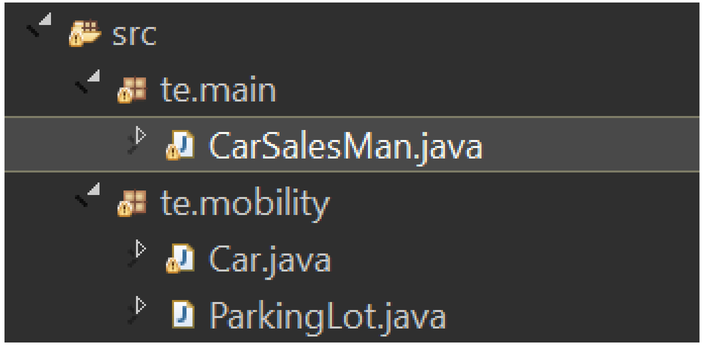
```
/ within some method:
Car myCar = new Car();
System.out.println(myCar.color);
// This is an illegal declaration,
// color is private.
```

Data Members: Access Modifiers Example

If the color data member is **public**: (NOTE: data members should **NEVER** be public!)

Car.java

```
package te.mobility;

public class Car {
   public String color = "green";
}
```



CarSalesMan.java

```
// within some method:
Car myCar = new Car();
System.out.println(myCar.color);
// This is fine.
```

ParkingLot.java

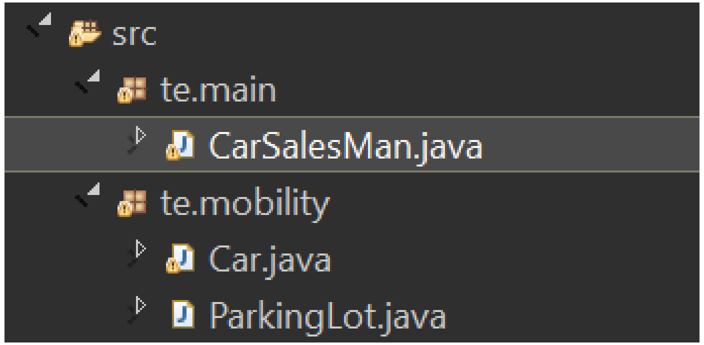
```
/ within some method:
Car myCar = new Car();
System.out.println(myCar.color);
// This is fine.
```

Data Members: Access Modifiers Example

If the color data member is **default**:

Car.java

```
package te.mobility;

public class Car {
   String color = "green";
}
```



```
src
  te.main
    CarSalesMan.java
  te.mobility
    Car.java
    ParkingLot.java
```

CarSalesMan.java

```
// within some method:
Car myCar = new Car();
System.out.println(myCar.color);
// This is invalid now,
CarSalesMan // is on a different
package.
```

ParkingLot.java

```
// within some method:
Car myCar = new Car();
System.out.println(myCar.color);
// This is fine! Default allows
// access from the same package.
```

Data Members: Getters and Setters

Data members should **always be private**.

- Access to data members will be provided via getter and setter methods.
- Getter methods allow the outside world to retrieve the value of the data member.
- Setter methods allow the outside world to set the value of the data member.

Data Members: Getters and Setters

Here, a getter and setter have been created for the color data member:

```
package te.mobility;

public class Car {
   private String color = "green";

   public String getColor() {
     return color;
   }

   public void setColor(String color) {
     this.color = color;
   }
}
```

This is a getter, it simply returns the value of the data member.

This is a setter, it takes 1 parameter, which will be used to update the data member

“this” is used to differentiate the data member from the parameter passed in.

Data Members: Getters and Setters

Consider the CarSalesMan class. It can now call the getter method to obtain the color,and the setter method to change the car’s color.

Car.java

```java
package te.mobility;

public class Car {
   private String color = "green";

   public String getColor() {
      return color;
   }

   public void setColor(String color) {
      this.color = color;
   }
}
```

CarSalesMan.java

```java
package te.main;

import te.mobility.Car;

public class CarSalesMan {

   public static void main(String args[]) {
      Car thisCar = new Car();
      System.out.println(thisCar.getColor());
        // green
      thisCar.setColor(“blue”);

      System.out.println(thisCar.getColor());
        // blue
   }
}
```

Methods: Declaring

Refer to the notes in the Inputs / Outputs lecture on declaring methods, here will just emphasize how methods are called.

Car.java

```
package te.mobility;

public class Car {
   private String color = "green";
   private boolean engineOn = false;

   public String getColor() {
      return color;
   }

   public void setColor(String color) {
      this.color = color;
   }

   public void goInReverse() {
      System.out.println("going backwards.");
   }
}
```

Driver.java

```
package te.main;

import te.mobility.Car;

public class Driver {

   public static void main (String args[]) {

      Car shinyNewCar = new Car();
      shinyNewCar.goInReverse();
   }
}
```

Methods: Access Modifiers

Methods have the same access modifiers as data members and obey the same rules in terms of visibility to the outside world.

Methods: Constructors

Constructors are special methods designed to help initialize an object of a class.

Consider the following declaration:

```
Car shinyNewCar = new Car();
```

Every class has a default constructor that takes no arguments, in this case it's Car().

The new keyword instantiates an object and creates space for it in memory.

Methods: Constructors Declaration

Custom constructors can be declared following this pattern:

<Name of The Class> (**parameter type & name 1**, **parameter type & name 2**) {

… // body of constructor}

Some rules must be followed:

- The constructor has no return type.
- The constructor’s name must be identical to the class name.
- The constructor can have access modifiers if needed.

Methods: Constructors Declaration Example

A custom constructor with 2 parameters has been created for Car:

```
package te.mobility;

public class Car {
   private String color = "green";
   private int numOfDoors = 4;

   public Car(String color, int numberOfDoors) {
      this.color = color;
      this.numOfDoors = numberOfDoors;
   }
}
```

Methods: Constructors Declaration Example

Having defined a constructor in this manner allows for car to be instantiated by providing two parameters.

```
package te.mobility;

public class Car {
   private String color = "green";
   private int numOfDoors = 4;

   public Car(String color, int numberOfDoors) {
      this.color = color;
      this.numOfDoors = numberOfDoors;
   }
}
```

```
package te.main;

import te.mobility.Car;

public class CarSalesMan {

   public static void main(String args[]) {
      Car thisCar = new Car("blue", 4);
   }
}
```

We have now instantiated a blue car with 4 doors.

Summary of Class Components

```java
package te.mobility;

public class Car {
      private String color = "green";
      private int numOfDoors = 4;
      private int fuelRemaining = 5;
      private int totalFuelCapacity = 10;

      public Car(String color, int numberOfDoors) {
         this.setColor(color);
         this.setNumOfDoors(numberOfDoors);
      }

      public void goForward() {
         System.out.println("going forward");
      }

      public double fuelRemaining() {
        return fuelRemaining/totalFuelCapacity * 100;
      }

      public String getColor() {
        return color;
      }

      public void setColor(String color) {
         this.color = color;
      }

      public int getNumOfDoors() {
        return numOfDoors;
      }

      public void setNumOfDoors(int numOfDoors) {
        this.numOfDoors = numOfDoors;
      }
}
```

These are the data members for the class.

This is a constructor that takes two arguments.

These are methods of the class that perform a task.

These are getters and setters for the two of the data members.