Module 1-9

Classes and Objects

Object Oriented Programming

Object Oriented Programming is a style of programming where we define a blueprint for things we want to model (**classes**). We can proceed to create instances (**objects**) of those blueprints. These objects can contain data in the form of fields, or perform certain actions (**methods**).

Object Oriented Programming: Key Ideas

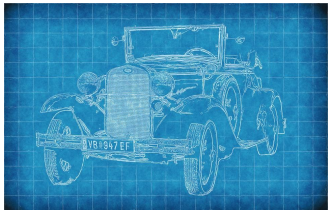
There are three underlying OOP principles:

- Encapsulation
- Inheritance
- Polymorphism

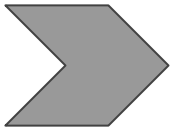
By the end of today, you should be able to define encapsulation.

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. **These objects are instances of the class.**

Objects: Properties and Methods

Objects have properties 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. Consider some of the things a vehicle can 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 class:

```
package te.mobility;

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

By convention, class names always start with a capital letter.

Properties: Declaration

Classes have properties. 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;
}
```

Properties: Default Values

Properties have default values, when none are provided by the programmer:

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

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: Getters and Setters

Getters and Setters are special types of methods.

- Data members should **always be private**.
- Access to properties 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.

This practice is known as encapsulation.

Methods: 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 property.

This is a setter, it takes 1 parameter, which will be used to update the property's value

"this" is used to differentiate the property from the parameter passed in.

Methods: 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

```
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

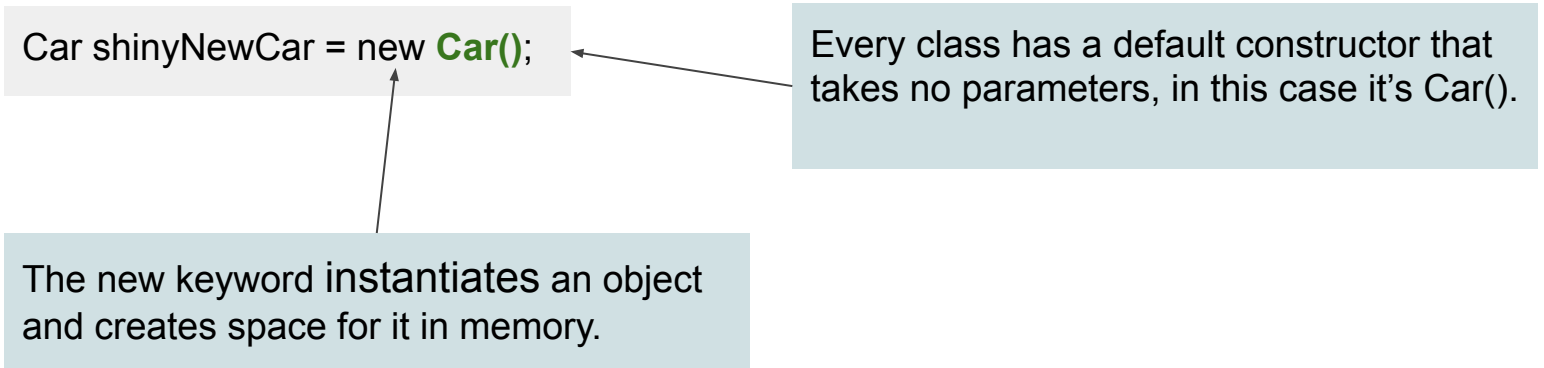
```
package te.main;
import te.mobility.Car;

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
}}
```

Constructors

Constructors are “method like constructs” in Java designed to help instantiate an object of a class.Consider the following declaration:



Methods: Constructors Declaration

Custom constructors can be declared following this pattern:

```
Name of The Class (parameter1, parameter2) {

… // body of constructor }
```

Two rules to have in mind:

- The constructor has no return type.
- The constructor’s name must be identical to the class name.

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.

Properties & Methods: Access Modifiers

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.

Class 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 {...}*

There is no "default" keyword, default is just not specifying anything!

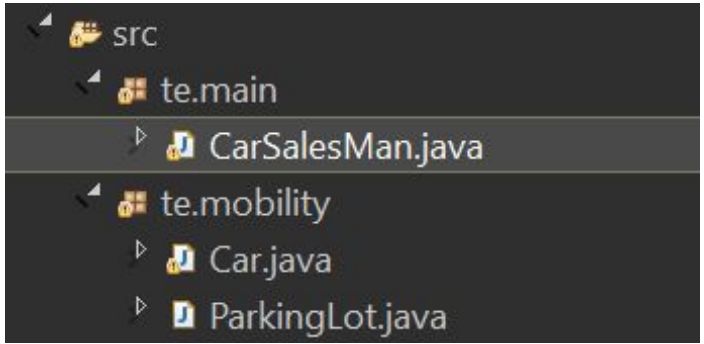
Props and Methods Access Modifiers Example

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

Car.java

```
package te.mobility;

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



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.
```

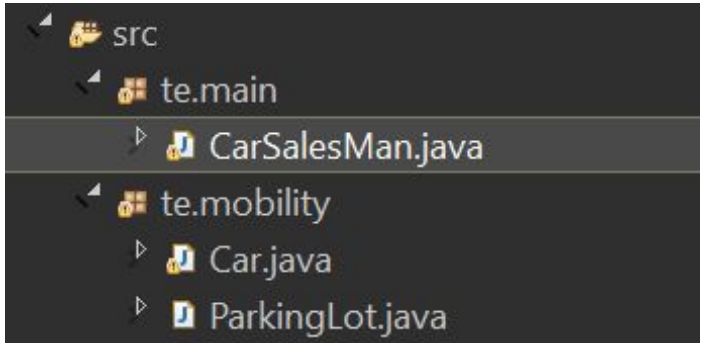
Props and Methods Access Modifiers Example

If the color data member is **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.
```

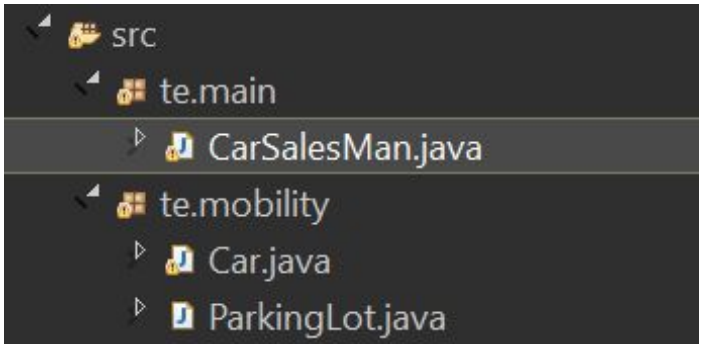
Props and Methods Access Modifiers Example

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

Car.java

```
package te.mobility;

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



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.
```

Summary of Class Components

```
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 this.numOfDoors;
        }

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

These are the properties for the class. (a.k.a instance variables.)

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.

Encapsulation

We have seen today how code is organized in the class to achieve encapsulation:

- Bundled the behavior and state of what we want to model into classes
- Manage internal logic and consistency
- Setting all