Module 1-13

Abstract Classes

Abstract Classes

Abstract Classes combine some of the features we've seen in interfaces and with inheriting from concrete classes.

Abstract Classes

Its features:

- Abstract methods can be extended by concrete classes.
- Abstract classes can have abstract methods
- Abstract classes can have concrete methods
- Abstract classes can have constructors
- Abstract classes, like Interfaces, cannot be instantiated

Abstract Classes : Declaration

We use the following pattern to declare abstract classes.

- The abstract class itself:

  public abstract class Name of the Abstract Class {...}

- The child class that inherits from the abstract class:

  public class Child Class extends Name of Abstract Class

Abstract Classes Example

```
package te.mobility;

public abstract class Vehicle {

        private int numberOfWheels;
        private double tankCapacity;
        private double fuelLeft;

        public Vehicle(int numberOfWheels) {
                this.numberOfWheels = numberOfWheels;
        }
        public double getTankCapacity() {
                return tankCapacity;
        }

        public abstract Double calculateFuelPercentage();


        public double getFuelLeft() {
                return fuelLeft;
        }
}
```

We need to implement the constructor

We need to implement the abstract method

extends, not implement, is used.

```
package te.mobility;

public class Car extends Vehicle {

        public Car(int numberOfWheels) {
                super(numberOfWheels);
        }

        @Override
        public Double calculateFuelPercentage() {

        return super.getFuelLeft() / super.getTankCapacity() * 100;

        }

}
```

Also note how we are able to call concrete methods within the Vehicle abstract class

Abstract Classes: final keyword

Declaring methods as final prevent them from being overriden by a child class.

```
package te.mobility;

public abstract class Vehicle {
...

        public final void refuelCar() {
                this.fuelLeft = tankCapacity;
...
}
```

```
package te.mobility;

public class Car extends Vehicle {

        @Override
        public void refuelCar() {


        }

}
```

This override will cause an error, as the method is marked as final.

Multiple Inheritance

- Java does not allow multiple inheritance of concrete classes or abstract classes. The following is not allowed:

```
public class Car extends Vehicle, MotorVehicles {...}
```

- Java does allow for the implementation of multiple interfaces:

```
public class Car implements IVehicle, IMotorVehicle
{...}
```

Where Vehicle and MotorVehicles are classes or abstract classes

Where IVehicle and IMotorVehicle are interfaces

Protected Access

- Properties and methods marked as protected are visible to other classes in the same package and to its subclasses.
  - The subclasses can be in other packages.

Protected Access Example

```
package packageA;

public class ParentClass {

        protected void doAction() {
                // Method Body
        }
}
```

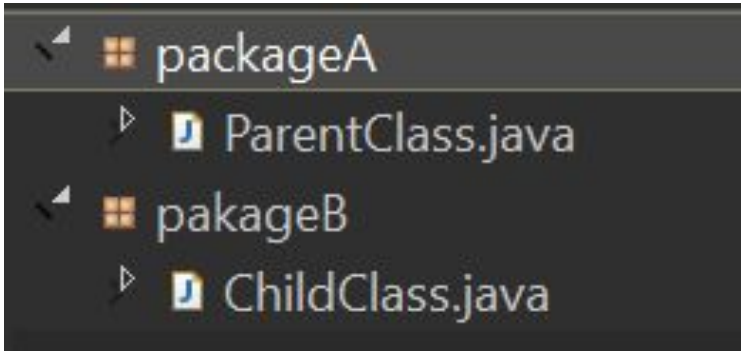
```
package pakageB;

import packageA.ParentClass;

public class ChildClass extends ParentClass {

        public void childMethod() {
                super.doAction();
        }

}
```



- Note that the parent and child classes exist in different packages.
- Since the parent method doAction is protected, it is visible to the child class.
- Omitting the access modifier (default) on doAction() or setting it to private will result in a compiler error.

Access Modifier Review

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Source:
https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html