

Module 1-13

Abstract Classes

If else if chain vs. switch statement

```
...
System.out.println("Movie ticket prices: ");
System.out.println("1. Adult - $14.00");
System.out.println("2. Child - $8.00");
System.out.println("3. Senior - $11.00");
System.out.print("Enter choice: ");
int choice = Integer.parseInt(input.nextLine());
    if (choice == 1) {
        total = quantity * 14;
    } else if (choice == 2) {
        total = quantity * 8;
    } else if (choice == 3) {
        total = quantity * 11;
    } else {
        System.out.println("Invalid entry");
    }

```

...

```
...
System.out.println("Movie ticket prices: ");
System.out.println("1. Adult - $14.00");
System.out.println("2. Child - $8.00");
System.out.println("3. Senior - $11.00");
System.out.print("Enter choice: ");
int choice = Integer.parseInt(input.nextLine());
    switch (choice) {
        case 1:
            total = quantity * 14;
            break;
        case 2:
            total = quantity * 8;
            break;
        case 3:
            total = quantity * 11;
            break;
        default:
            System.out.println("Invalid entry");
    }

```

...

Objectives

- Students should be able to define and use abstract in the context of a class and a method
- Students should be able to define and use final in the context of a class and a method
- Students should understand what a design pattern is and how to research them
- Students should be able to explain the differences between public, private, and protected access
- Students should understand that many keywords in Java are not for security, but for design and letting other developers know how to use your code

Abstract Classes

Abstract Classes combine some of the features we've seen in interfaces along with inheriting from a concrete class.

- 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 <<Name of 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

```
package te.mobility;
```

```
public class Car extends Vehicle {
```

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

```
    @Override  
    public Double calculateFuelPercentage() {
```

```
        super.getTankCapacity() * 100;  
    }
```

```
        return super.getFuelLeft() /
```

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

extends, not
implement, is
used.

Abstract Classes: final keyword

Declaring methods as final prevent them from being overridden 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 {...}
```

Where Vehicle and MotorVehicles are classes or abstract classes



- Java does allow for the implementation of multiple interfaces:

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

Where IVehicle and IMotorVehicle are interfaces

