CSE 215: Programming Language II Lab

Lab – 8 **Lab Officer: Tanzina Tazreen**

Abstract Class & Interface

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user. it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

There are two ways to achieve abstraction in java

1. Abstract class
2. Interface

Abstract class

To create an abstract class, you need to declare a class abstract, using the **abstract** keyword.

- Abstract class has to have at least one abstract method (abstract method is a method with the keyword abstract and without a body, only the header).
- An abstract class may or may not have all abstract methods. Some of them can be concrete methods
- If an abstract class cannot be instantiated (you cannot create object of that class).
- To use an abstract class, you have to inherit it from another class using the keyword "**extend**".
- If you inherit an abstract class, you have to implement all the abstract methods in it. thus making overriding compulsory
- An abstract class can have parameterized constructors and the default constructor is always present in an abstract class.
- It can have static methods also.

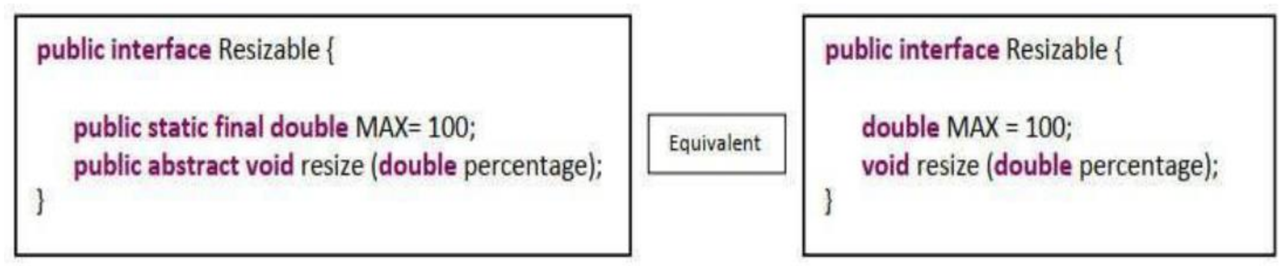
When to use abstract classes and abstract methods

There are situations in which we will want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method. Sometimes we will want to create a superclass that only defines a generalization form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.

```
// Abstract class
abstract class Animal
{
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep()
  {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal
{
  public void animalSound()
  {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal
{
  public void animalSound()
  {
    // The body of animalSound() is provided here
    System.out.println("The Dog says: vow vow");
  }
}


class Main {
  public static void main(String[] args)
  {
    //abstract class can be used to create an object
    // Animal an = new Animal();
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

An **interface** is a completely "**abstract class**" that is used to group related abstract methods (methods with empty bodies).

- An interface cannot have regular methods, only abstract methods. But you do not need to use the keyword “abstract” to declare a method abstract.
- Like abstract classes, interfaces cannot be used to create objects.
- To use an interface, you have to inherit it from another class using the keyword “**implements**”.
- If you inherit an interface, you have to implement all the abstract methods in it. thus, making overriding compulsory.
- Interface methods are by default abstract and public.
- Interface attributes are by default public, static and final.
- An interface cannot contain a constructor.



```
public interface Resizable {

    public static final double MAX= 100;
    public abstract void resize (double percentage);
}
```

Equivalent

```
public interface Resizable {

    double MAX = 100;
    void resize (double percentage);
}
```

Why And When To Use Interfaces?

1) To achieve security by complete abstraction - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces. **Note:** To implement multiple interfaces, separate them with a comma

```
interface Animal {
  public void animalSound(); // interface method (does not have a
body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal
{
  public void animalSound()
  {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep()
  {
    // The body of sleep() is provided here
    System.out.println("Pigs sleep all night long…..Zzzz");
  }
}

// Dog "implements" the Animal interface
class Pig implements Animal
{
  public void animalSound()
  {
    // The body of animalSound() is provided here
    System.out.println("The pig says: vow vow");
  }
  public void sleep()
  {
    // The body of sleep() is provided here
    System.out.println("Dogs hardly sleeps at night ");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

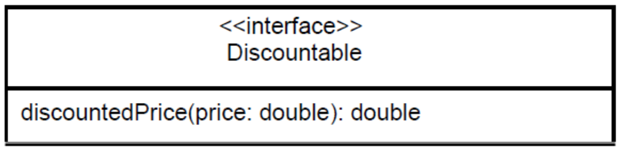
Difference between abstract class and interface

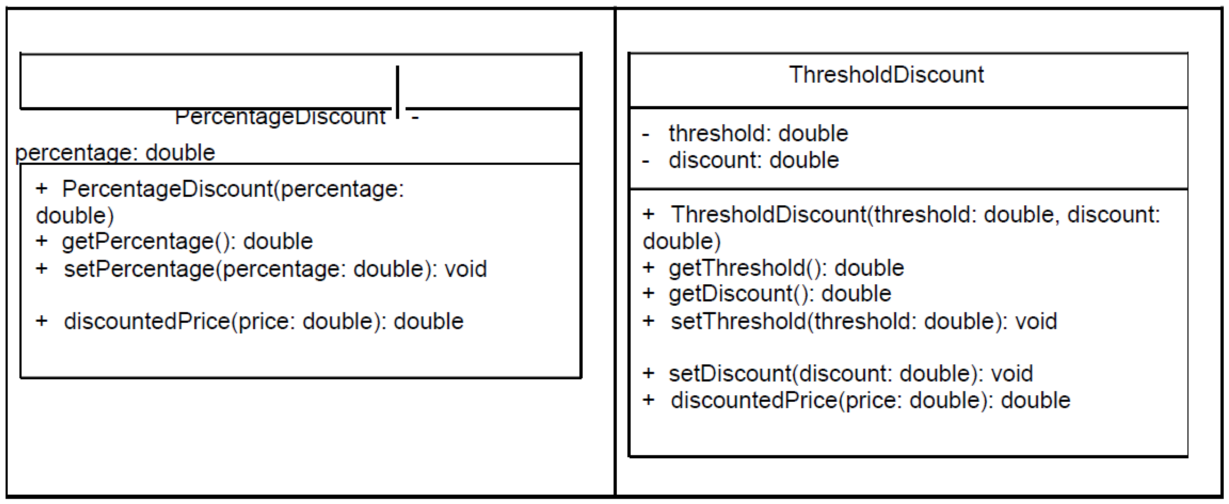
Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

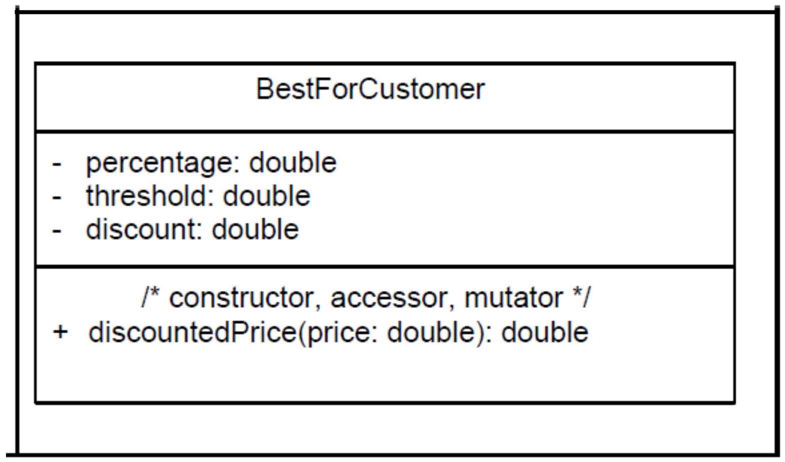
Tasks:

Implement the following classes and invoke discountedPrice() for object of each class.





PercentageDiscount
- percentage: double
+ PercentageDiscount(percentage: double)
+ getPercentage(): double
+ setPercentage(percentage: double): void
+ discountedPrice(price: double): double
ThresholdDiscount
- threshold: double
- discount: double
+ ThresholdDiscount(threshold: double, discount: double)
+ getThreshold(): double
+ getDiscount(): double
+ setThreshold(threshold: double): void
+ setDiscount(discount: double): void
+ discountedPrice(price: double): double



BestForCustomer
- percentage: double
- threshold: double
- discount: double
/* constructor, accessor, mutator */
+ discountedPrice(price: double): double

discountedPrice() from BestForCustomer class will consider both percentage and threshold discount and give the customer the best possible sales price.

For PercentageDiscount class:

discountedPrice = price – ((price *percentage)/100)

For ThresholdDiscount class:

If price< threshold, discount = 0
discountedPrice = price – discount