Abstract Classes in Java

Question 1
Run below coding one by one and do summary based on coding below.

Summary the output and make your own note **about abstract classes in Java**		
	Output	**Remark/Note**
1	**Derived fun() called**	An object of abstract class cannot be created directly, but the extends class of an abstract class can be created
2	Base Contructor Called Derived Constructor Called	Inside an abstract class, it can contain constructors. And the function inside the constructor will be called if an inherited class object was created.
3	Base fun() called	An abstract class, also can have 0 abstract methods and still can be inherited. But the classes created cannot be instantiated.
4	Derived fun() called	In an abstract class, there are final methods which cannot be overridden.
5	Error: Test is abstract; cannot be instantiated	It is impossible to instantiate or create an object of abstract class.
6	Lets have some fun!!	Abstract classes can be called independently without declaring or creating objects using the "define static methods" in the abstract class.

Following are some important observations about abstract classes in Java.

1) An instance of an abstract class cannot be created, we can have references to abstract class type though.

```
abstract class Base {
   abstract void fun();
}
class Derived extends Base {
   void fun()
   {
      System.out.println("Derived fun() called");
   }
}
class Main {
   public static void main(String args[])
   {

      // Uncommenting the following line will cause
      // compiler error as the line tries to create an
      // instance of abstract class.
      // Base b = new Base();

      // We can have references of Base type.
      Base b = new Derived();
      b.fun();
   }
}
```

2) An abstract class can contain constructors in Java. And a constructor of abstract class is called when an instance of an inherited class is created. For example, the following is a valid Java program.

```
// An abstract class with constructor
abstract class Base {
    Base()
    {
        System.out.println("Base Constructor Called");
    }
    abstract void fun();
}
class Derived extends Base {
    Derived()
    {
        System.out.println("Derived Constructor Called");
    }
    void fun()
    {
        System.out.println("Derived fun() called");
    }
}
class Main {
    public static void main(String args[])
    {
        Derived d = new Derived();
    }
}
```

3) We can have an abstract class without any abstract method. This allows us to create classes that cannot be instantiated but can only be inherited.

```
// An abstract class without any abstract method abstract
class Base {
    void fun() { System.out.println("Base fun() called"); }
}

class Derived extends Base {
}

class Main {
    public static void main(String args[])
    {
        Derived d = new Derived();
        d.fun();
    }
}
```

4) Abstract classes can also have final methods (methods that cannot be overridden). For example, the following program compiles and runs fine.

```
// An abstract class with a final method
abstract class Base {
    final void fun()
    {
        System.out.println("Derived fun() called");
    }
}

class Derived extends Base {
}

class Main {
    public static void main(String args[])
    {
        Base b = new Derived();
        b.fun();
    }
}
```

5) For any abstract java class we are not allowed to create an object i.e., for abstract class instantiation is not possible.

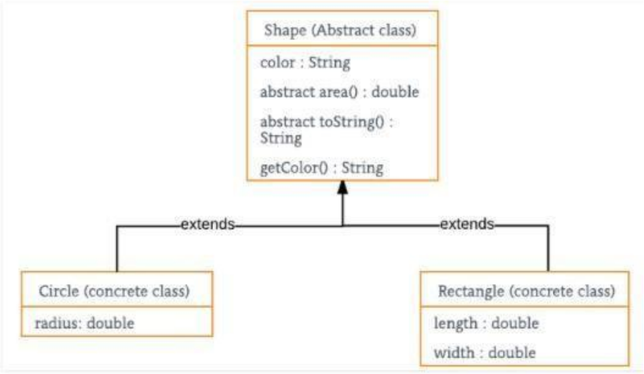
```
// An abstract class example
abstract class Test {
    public static void main(String args[])
    {
        // Try to create an object
        Test t = new Test();
    }
}
```

6) Similar to the interface we can define static methods in an abstract class that can be called independently without an object.

```
abstract class Party {
    static void doParty()
    {
        System.out.println("Lets have some fun!!");
    }
}

public class Main extends Party {
    public static void main(String[] args)
    {
        Party.doParty();
    }
}
```

Abstract classes and Abstract methods:

- **An abstract class is a class that is declared with abstract keyword.**
- **An abstract method is a method that is declared without an implementation.**
- **An abstract class may or may not have all abstract methods. Some of them can be concrete methods**
- **A method defined abstract must always be redefined in the subclass, thus making overriding compulsory OR either make subclass itself abstract.**
- **Any class that contains one or more abstract methods must also be declared with abstract keyword.**
- **There can be no object of an abstract class. That is, an abstract class can not be directly instantiated with the new operator.**
- **An abstract class can have parametrized constructors and default constructor is always present in an abstract class.**

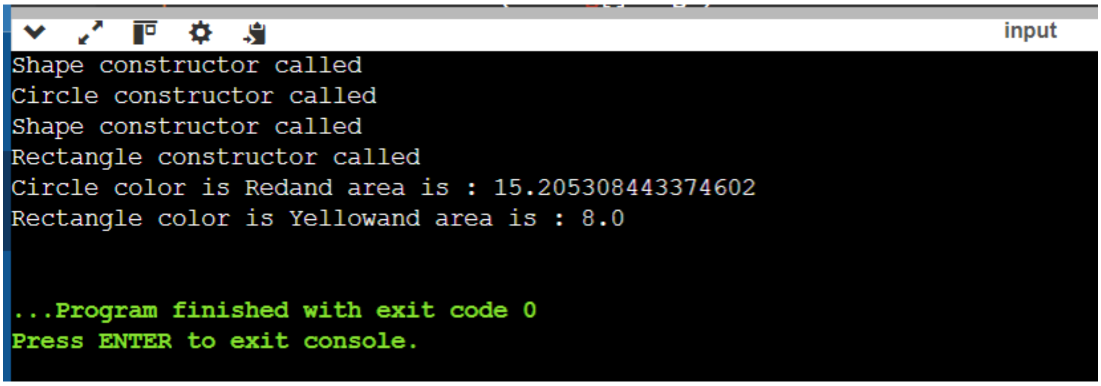


Question 2

(a) Run below coding and shown us the output.

```
// Java program to illustrate the
// concept of Abstraction
abstract class Shape {
    String color;

    // these are abstract methods abstract
    double area();
    public abstract String toString();

    // abstract class can have constructor public
    Shape(String color)
    {
        System.out.println("Shape constructor called");
        this.color = color;
    }

    // this is a concrete method
    public String getColor() { return color; }
}
class Circle extends Shape {
    double radius;
    public Circle(String color, double radius)
    {

        // calling Shape constructor
        super(color);
        System.out.println("Circle constructor called");
        this.radius = radius;
    }

    @Override double area()
    {
        return Math.PI * Math.pow(radius, 2);
    }

    @Override public String toString()
    {
        return "Circle color is " + super.getColor()
            + "and area is : " + area();
    }
}
class Rectangle extends Shape {

    double length;
    double width;

    public Rectangle(String color, double length,
                      double width)
    {
        // calling Shape constructor
        super(color);
        System.out.println("Rectangle constructor called");
        this.length = length;
        this.width = width;
    }

    @Override double area() { return length * width; }

    @Override public String toString()
    {
        return "Rectangle color is " + super.getColor()
            + "and area is : " + area();
    }
}
public class Test {
    public static void main(String[] args)
    {
        Shape s1 = new Circle("Red", 2.2);
        Shape s2 = new Rectangle("Yellow", 2, 4);

        System.out.println(s1.toString());
        System.out.println(s2.toString());
    }
}
```

OUTPUT:



```
Shape constructor called
Circle constructor called
Shape constructor called
Rectangle constructor called
Circle color is Redand area is : 15.205308443374602
Rectangle color is Yellowand area is : 8.0


...Program finished with exit code 0
Press ENTER to exit console.
```

(b) Summary in table **about abstract classes and abstract methods in Java** based on above yellow notes and coding output Q2 (a).

Abstract Classes	**Abstract Methods**
An abstract class is declared in the coding section using the “abstract” keyword. For example: *abstract class shape.*	An abstract method is declared without an implementation or a body. These abstract methods are meant to be overridden by subclasses, which will define their own coding implementation inside the method body.
An abstract class can have both abstract and concrete methods or one of them. This allows abstract classes to provide their own implementation details whether to customize or extend it.	
A class will have to be declared as an abstract class when have a common function, attributes with different implementation and want to be manipulated by other subclasses according to the system needs.	Abstract methods must be overridden in subclasses unless the subclass is also abstract.
No objects can be created from an abstract class using the “new” operator. Instead the objects must be created from subclasses that inherit from the abstract class.	
Abstract classes can have parameterized constructors, and a default constructor is always present	

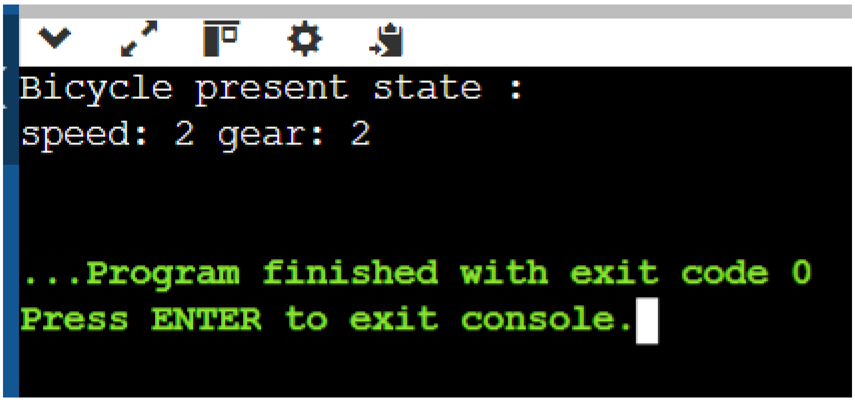
Interfaces in Java

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

Question 3.
Run and study coding below.

```
************
import java.io.*;

    interface Vehicle {

        // all are the abstract methods.
        void changeGear(int a);
        void speedUp(int a);
        void applyBrakes(int a);
    }

    class Bicycle implements Vehicle{

        int speed;
        int gear;

         // to change gear
        @Override
        public void changeGear(int newGear){

            gear = newGear;
        }

        // to increase speed
        @Override
        public void speedUp(int increment){

            speed = speed + increment;
        }

        // to decrease speed
        @Override
        public void applyBrakes(int decrement){

            speed = speed - decrement;
        }

        public void printStates() {
             System.out.println("speed: " + speed
                 + " gear: " + gear);
            }
        }

class GFG {

    public static void main (String[] args) {

        // creating an inatance of Bicycle
        // doing some operations
        Bicycle bicycle = new Bicycle();
        bicycle.changeGear(2);
        bicycle.speedUp(3);
        bicycle.applyBrakes(1);

        System.out.println("Bicycle present state :");
        bicycle.printStates();

        // creating instance of the bike.
        Bike bike = new Bike();
        bike.changeGear(1);
        bike.speedUp(4);
        bike.applyBrakes(3);

        System.out.println("Bike present state :");
        bike.printStates();
    }
}
```

a) What is the output for Q2?

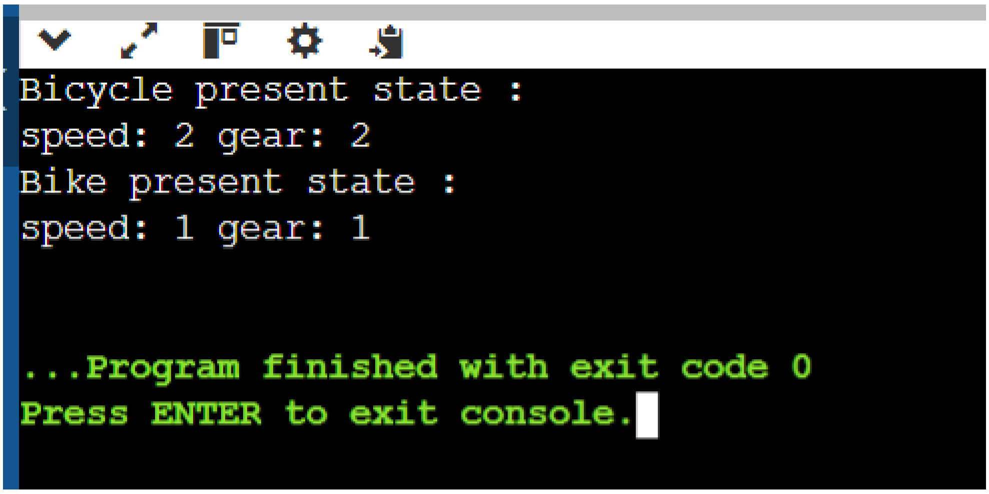


```
Bicycle present state :
speed: 2 gear: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

b) Extend the above coding by adding class Bike implements Vechicle.

```
class Bike implements Vehicle{

    int speed;
    int gear;

     // to change gear
    @Override
    public void changeGear(int newGear){

        gear = newGear;
    }

    // to increase speed
    @Override
    public void speedUp(int increment){

        speed = speed + increment;
    }

    // to decrease speed
    @Override
    public void applyBrakes(int decrement){

        speed = speed - decrement;
    }

    public void printStates() {
         System.out.println("speed: " + speed
              + " gear: " + gear);
              }
}
```

c) What is the output for Q2 (b)?



```
Bicycle present state :
speed: 2 gear: 2
Bike present state :
speed: 1 gear: 1


...Program finished with exit code 0
Press ENTER to exit console.
```