Java Training Questions - Phase 1

1. Why is an abstract class used in Java?

   An abstract class is used to achieve abstraction in java. Abstraction is the mechanism of hiding the implementation details and showing only the functionality to the user. This in turn helps to achieve security as we can hide certain details and only show the important details of an object.

2. Write a Java program to create an interface Shape with the *getArea()* method. Create three classes Rectangle, Circle, and Triangle that implement the Shape interface. Implement the getArea() method for each of the three classes.

```
import java.util.Scanner;

public interface Shape {
    void getArea();
}

class Rectangle implements Shape{
    double l,w;

    Rectangle(double l,double w){
        this.l = l;
        this.w = w;
    }

    public void getArea(){
        System.out.println("Area of rectangle = "+(l * w) );
    }
}

class Circle implements Shape{
    double r;

    Circle(double r){
        this.r = r;
    }

    public void getArea(){
```

```
        System.out.println("Area of circle = "+((Math.PI) * r * r));
    }
}

class Triangle implements Shape{
    double h,b;

    Triangle(double b,double h){
        this.b = b;
        this.h = h;
    }

    public void getArea(){
        System.out.println("Area of triangle = "+(0.5 * b * h));
    }
}

class Question2{
    public static void main(String[] args) {
        double length,width;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the length of the rectangle : ");
        length = sc.nextDouble();
        System.out.println("Enter the width of the rectangle : ");
        width = sc.nextDouble();
        Shape r1 = new Rectangle(length,width);
        r1.getArea();
        System.out.println("Enter the radius of the circle : ");
        double radius = sc.nextDouble();
        Shape c1 = new Circle(radius);
        c1.getArea();
        System.out.println("Enter the base of the triangle : ");
        double base = sc.nextDouble();
        System.out.println("Enter the height of the triangle : ");
        double height = sc.nextDouble();
        Shape t1 = new Triangle(base,height);
        t1.getArea();
    }
}
```

3. Write a Java program to create an interface Searchable with a method *search(String keyword) t*hat searches for a given keyword in a text document. Create two classes Document and WebPage that implement the Searchable interface and provide their own implementations of the search() method.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

interface Searchable {
    boolean search(String keyword);
}

class Document implements Searchable{
    public boolean search(String keyword){
        boolean flag = false;
        try{
            File f = new File("Files\\Doc.txt");
            Scanner sc = new Scanner(f);
            while(sc.hasNextLine()){
                String line = sc.nextLine();
                if(line.contains(keyword)){
                    flag=true;
                    break;
                }
            }
            sc.close();
        }
        catch(FileNotFoundException e){
            e.printStackTrace();
        }
        return flag;
    }
}

class WebPage implements Searchable {
    String url;
    String content;
```

```
    public WebPage(String url, String content) {
        this.url = url;
        this.content = content;
    }

    public boolean search(String keyword) {
        return content.contains(keyword);
    }
}

public class Question3{
    public static void main(String[] args){
        Document Obj1 = new Document();
        System.out.println("Enter the keyword : ");
        Scanner sc = new Scanner(System.in);
        String keyword = sc.nextLine();
        boolean res1 = Obj1.search(keyword);
        System.out.println(res1);
        WebPage Obj2 = new WebPage("http://example.com","This is a
sample web page. Welcome to it thank you");
        boolean res2 = Obj2.search(keyword);
        System.out.println(res2);
        sc.close();
    }
}
```

4. Why is the below code showing a compile time error?

Java

```
interface X
{
    void methodX();
}

class Y implements X
{
    void methodX()
    {
        System.out.println("Method X");
    }
```

```
}
```

The code is showing a compile-time error because the methodX in class Y does not have the correct access modifier. In the interface X, methodX is implicitly public, so when a class implements the interface X, a public implementation of the method must be provided.

5. Does the code below compile successfully? If not, why?

```Java
interface A
{
    int i = 111;
}

class B implements A
{
    void methodB()
    {
        i = 222;
    }
}
```

The code will not compile successfully. The reason behind the same is that the variable i declared within the interface A is implicitly final, static and public. Being a final variable its value will be a constant and cannot be reassigned.

6. Find Output of the below program

```Java
class Test
{
       void myMethod()
       {
              System.out.println("Cred.ai");
       }
}
public class Derived extends Test
{
       void myMethod()
```

```
        {
                System.out.println("caytm");
        }

        public static void main(String[] args)
        {
                Derived object = new Test();
                object.myMethod();
        }
}
```

The code provided will not compile due to an incompatible type error. In Java, a superclass object cannot be assigned directly to a subclass reference. The correct type assignment is given by : Test object = new Derived();

7. Find Output of the below program

Java

```
class Caytm
{
        protected void getData()
        {
                System.out.println("Inside caytm");
        }
}
class Credai extends Caytm
{
        protected void getData()
        {
                System.out.println("Inside Cred.ai");
        }

        protected void getValue()
        {
                System.out.println("Cred.ai");
        }
}
```

```java
public class Test
{
	public static void main(String[] args)
	{
		Caytm obj = new Credai();
		obj.getValue();
	}
}
```

The code provided will not compile because the method getValue() is not defined in the class Caytm. Therefore the compiler produces an error.

8. Can we overload below 2 methods? If not, why?

```java
class Demo {

  static void test(int ... vargs) {
    // method body
  }

  static void test(int n, int ... vargs) {
    // method body
  }
}
```

No, overloading is not possible here. Since both the methods can have up to n parameters, it raises ambiguity and hence method overloading can’t be achieved.

9. In the below class, is ‘method’ overloaded or duplicated?

```java
public class MainClass
{
    void method(int ... a)
    {
        System.out.println(1);
    }
```

```
    void method(int[] a)
    {
        System.out.println(2);
    }
}
```

Here the method is overloaded and not duplicated. Method overloading can be achieved by changing the datatype of the arguments or by changing the number of arguments. The first method accepts a variable length argument of type int. However the second method takes an array of type int as argument. Hence both methods have different signatures and hence method overloading is ensured.

10. In a class, one method has two overloaded forms. One form is defined as static and another form is defined as non-static. Is that method properly overloaded?

Java

```
class X
{
    void calculate(int a, int b)
    {
        System.out.println("Class X");
    }
}

class Y extends X
{
    @Override
    void calculate(int a, int b)
    {
        System.out.println("Class Y");
    }
}

class Z extends Y
{
    @Override
    void calculate(int a, int b)
    {
        System.out.println("Class Z");
    }
}
```

```java
public class MainClass
{
    public static void main(String[] args)
    {
        X x = new Y();

        x.calculate(10, 20);

        Y y = (Y) x;

        y.calculate(50, 100);

        Z z = (Z) y;

        z.calculate(100, 200);
    }
}
```

The line X x = new Y(); will run correctly since it creates an instance of Y and assigns it to a reference of type X. This is valid because Y is a subclass of X.
The line Y y = (Y) x; casts x back to Y. This is valid because x is actually an instance of Y.
On the other hand Z z = (Z) y; this line is not valid since y refers to an instance of Y and not Z. Hence it raises a ClassCastException.

11. Find out the error in the below code?

```java
class X
{
    static void methodOfX()
    {
        System.out.println("Class X");
    }
}

class Y extends X
{
```

```
    @Override
    static void methodOfX()
    {
        System.out.println("Class X");
    }
}
```

The code raises an error because the child class attempts to override a static method. However in java static methods can't be overridden and can only be hidden.

12. Final methods can be overridden but can't be overloaded? True or False?

False. Final methods can't be overridden. However they can be overloaded either by changing the number of arguments of the method or by changing the data type of the arguments.