abstract class and interface
++++++++++++++++++++++++++++

Given:

```
11. abstract class Vehicle { public int speed() { return 0; }
12. class Car extends Vehicle { public int speed() { return 60; }
13. class RaceCar extends Car { public int speed() { return 150; } ...
21.   RaceCar racer = new RaceCar();
22.   Car car = new RaceCar();
23    Vehicle vehicle = new RaceCar();
24    System.out.println(racer.speed() + ", " + car.speed() + ", " +
vehicle.speed());
```

What is the result?
A. 0, 0, 0
B. 150, 60, 0
C. Compilation fails.
D. 150, 150, 150
E. An exception is thrown at runtime.

Answer: D

Note:
 1. An abstract class can contain "concrete methods" also along with abstract methods.
 2. During inheritance, concrete methods can be "Overriden".
 3. An abstract class need not have any abstract methods also.
 4. For an abstract class object can't be created, only reference can be created.

eg#1.

```
abstract class Vehicle
{
      public abstract int getNoOfWheels();

      public void infoAboutVehicle()
      {
            System.out.println("Generic information...");
      }
}
class Bus extends Vehicle
{
      @Override
      public int getNoOfWheels()
      {
            return 7;
      }

      @Override
      public void infoAboutVehicle()
      {
            System.out.println("Volvo bus...");
      }
}
class Auto extends Vehicle
{
      @Override
      public int getNoOfWheels()
      {
            return 3;
```

```
	}
	@Override
	public void infoAboutVehicle()
	{
		System.out.println("Uber Auto...");
	}
}

public class Test
{
	public static void main(String[] args)
	{
		//Vehicle v = new Vehicle(); //instantiation of Vehicle is not possible
: abstract

		Vehicle v1 = new Bus();
		v1.infoAboutVehicle();
		System.out.println("No of wheels for Bus is :: "+v1.getNoOfWheels());

		Vehicle v2 = new Auto();
		v2.infoAboutVehicle();
		System.out.println("No of wheels for Auto is :: "+v2.getNoOfWheels());
	}
}
```

```
Output
Volvo bus...
No of wheels for Bus is :: 7

Uber Auto...
No of wheels for Auto is :: 3
```

eg#2.

```
import java.util.Scanner;

abstract class Shapee
{
	public float area;

	//abstract methods
	public abstract void input();
	public abstract void calcArea();

	//concrete methods
	public void disp()
	{
		System.out.println("Area is :: "+area);
	}
}

class Square extends Shapee
{
	private float length;

	@Override
	public void input()
	{
		Scanner scanner = new Scanner(System.in);
		System.out.print("Enter the length for Square object:: ");
```

```
            length = scanner.nextFloat();
      }

      @Override
      public void calcArea()
      {
            area=length*length;
      }
}

class Rectanglee extends Shapee
{
      private float length;
      private float breadth;

      @Override
      public void input()
      {
            Scanner scanner = new Scanner(System.in);

            System.out.print("Enter the length of Rectangleee object:: ");
            length = scanner.nextFloat();
            System.out.print("Enter the breadth of Rectanglee object:: ");
            breadth = scanner.nextFloat();

      }

      @Override
      public void calcArea()
      {
            area=length*breadth;
      }
}

class Circle extends Shapee
{
      private float radius;

      @Override
      public void input()
      {
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter the radius of circlee:: ");
            radius = scanner.nextFloat();
      }

      @Override
      public void calcArea()
      {
            area=3.1414f * radius *radius;
      }
}

public class Test
{
      public static void main(String[] args)
      {
            //Creating a reference variable for Shape
            Shapee s;
            //Working with Square Object
            s = new Square();
            s.input();
            s.calcArea();
            s.disp();//generic method

            System.out.println();

            //Working with Rectangle Object
            s = new Rectanglee();
            s.input();
            s.calcArea();
            s.disp();//generic method

            System.out.println();

            //Working with Circle Object
            s = new Circle();
            s.input();
            s.calcArea();
            s.disp();//generic method
      }
}
```

```
Output
Enter the length for Square object:: 3
Area is :: 9.0

Enter the length of Rectangleee object:: 5
Enter the breadth of Rectanglee object:: 6
Area is :: 30.0

Enter the radius of circlee:: 4
Area is :: 50.2624
```

eg#2.

```
import java.util.Scanner;

abstract class Bird
{
      public abstract void fly();
      public abstract void eat();
}
final class Sparrow extends Bird
{
      @Override
      public void fly()
      {
            System.out.println("Fly very fast...");
      }

      @Override
      public void eat()
      {
            System.out.println("Sparrows eats grains...");
      }
}
```

```
abstract class Eagle extends Bird
{
      @Override
      public void fly()
      {
            System.out.println("All Eagles Fly very very high...");
      }

      public abstract void eat();
}
final class SerpentEagle extends Eagle
{
      @Override
      public void eat(){
            System.out.println("SerpentEagles eats snakes...");
      }
}
final class GoldenEagle extends Eagle
{
      @Override
      public void eat(){
            System.out.println("GoldeEagles catches the prey over the ocean...");
      }
}
final class Crow extends Bird
{
      @Override
      public void fly()
      {
            System.out.println("Crow fly at Meidum Height...");
      }

      @Override
      public void eat()
      {
            System.out.println("Crow eat Flesh...");
      }
}

//HelperClass
abstract class Sky
{
      /*
            Polymorphism : Overriding(1:M)

                         = new Sparrow();
            Bird ref = new SerpentEagle();
                         = new goldenEagle();
                         = new Crow();
      */
      public static void allowBird(Bird ref)
      {
            System.out.println("Working with object 
called::"+ref.getClass().getName());

            ref.fly();
            ref.eat();

            System.out.println();
      }
}

public class Test
{
      public static void main(String[] args)
      {

            Sparrow sp = new Sparrow();

            Eagle e1;
            Eagle e2;
            e1 = new SerpentEagle();
            e2 = new GoldenEagle();

            Crow c = new Crow();

            Sky.allowBird(sp);
            Sky.allowBird(e1);
            Sky.allowBird(e2);
            Sky.allowBird(c);
      }
}
```

```
Output
Working with object called::Sparrow
Fly very fast...
Sparrows eats grains...

Working with object called::SerpentEagle
All Eagles Fly very very high...
SerpentEagles eats snakes...

Working with object called::GoldenEagle
All Eagles Fly very very high...
GoldeEagles catches the prey over the ocean...

Working with object called::Crow
Crow fly at Meidum Height...
Crow eat Flesh...
```

```
Note: Illegal combinations of access modifier at methods level
      a. abstract and final  -------> [Illegal]
      b. abstract and static -------> [Illegal]
```

Question
Consider the following snippet and predict the output

```
public class Test{
      public static void main(String... args){
            String language="java";

            while(language.equals("java"){
                  if(language.equals("java"))
                        language=language.toUpperCase();
                  if(language.equals("JAVA"))
                        language=language.toLowerCase();
            }
            System.out.println(language); //line n1
      }
}
```

A. java
B. JAVA
C. Compile time error at line n1
D. Infintie time loop will run
E. None of the above

Answer: D

Question.
 1. Will constructor be called at the time of Object creation?
 Ans. yes

 2. Can we create an object for abstract class?
 Ans. No.

 3. Does abstract class have constructor?
 Ans. yes.

 4. If object can't be created for an abstract class, then why do we need a constructor?
 Ans.Even though we can't create an object for abstract class, still constructor is required, because in inheritance
     the child class object will be initialized by making a call to parent class constructor.
     constructor in abstract class is required to initialize the object completely.

eg#1.

```
//Person class
abstract class Person
{
      String name;
      int age;
      char gender;

      Person(String name,int age, char gender)
      {
            this.name =name;
            this.age = age;
            this.gender =gender;
      }

      public void dispDetails(){
            System.out.println("Name    is :: "+name);
            System.out.println("Age     is :: "+age);
            System.out.println("Gender  is :: "+gender);
      }
}

//Concrete class
class Student extends Person
{
      int sid;
      float avg;

      Student(String name,int age,char gender,int sid, float avg)
      {
            super(name,age,gender);
            this.sid = sid;
            this.avg = avg;
      }

      public void dispDetails(){
            super.dispDetails();
            System.out.println("SID     is :: "+sid);
            System.out.println("AVG     is :: "+avg);
      }
}
public class Test
{
      public static void main(String[] args)
      {
            Person p;
            p = new Student("sachin",51,'M',10,57.5f);
            p.dispDetails();

      }
}
Output
Name    is :: sachin
Age     is :: 51
Gender  is :: M
SID     is :: 10
AVG     is :: 57.5
```

5. When we create an object of child class, will object of parent class also be created?
  Ans. No, parent class constructor will be executed to make the child object complete(initialized).

```
eg#1.
//Person class
abstract class Person
{
      String name;
      int age;
      char gender;

      Person(String name,int age, char gender)
      {
            System.out.println("HashCode is :: "+this.hashCode());
            this.name =name;
            this.age = age;
            this.gender =gender;
      }

      public void dispDetails(){
            System.out.println("Name    is :: "+name);
            System.out.println("Age     is :: "+age);
            System.out.println("Gender  is :: "+gender);
      }
}
```

```
//Concrete class
class Student extends Person
{
      int sid;
      float avg;

      Student(String name,int age,char gender,int sid, float avg)
      {
            super(name,age,gender);
            System.out.println("HashCode is :: "+this.hashCode());
            this.sid = sid;
            this.avg = avg;
      }

      public void dispDetails(){
            super.dispDetails();
            System.out.println("SID     is :: "+sid);
            System.out.println("AVG     is :: "+avg);
      }
}
public class Test
{
      public static void main(String[] args)
      {
            Person p;
            p = new Student("sachin",51,'M',10,57.5f);
            System.out.println("HashCode is :: "+p.hashCode());
            p.dispDetails();

      }
}
```

Output

```
HashCode is :: 366712642
HashCode is :: 366712642
HashCode is :: 366712642
Name    is :: sachin
Age     is :: 51
Gender  is :: M
SID     is :: 10
AVG     is :: 57.5
```

Q>
Consider below code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
        for(int x = 10, y = 11, z = 12; y > x && z > y; y++, z -= 2) {
            System.out.println(x + y + z);
        }
    }
}
```

What will be the result of compiling and executing Test class?

```
A. 33
B. 32
C. 34
D. 33
   32
E. CompilationError
```

```
Interfaces in java
++++++++++++++++++

Definition-1
 => Any Service Requirement Specification is called "interface".
  eg#1. SunMS is responsible to define JDBC API and Database vendors is responsible
to provide implementation to it.

Definition-2
 => It refers to the contract b/w client and the service provider
  eg#2. ATM GUI screen describes what bank people are offering,at the same time GUI
SCREEN represents what customer is expecting
        So GUI screen acts like a bridge b/w client and the service provider.

Definition-3
 => Inside interface every method present are "public and abstract".
 => Since it is public and abstract,we say interface as "pure abstract class".

eg#1.
interface Calculator
{
      //By default all the methods are public and abstract.
       void add(int a,int b);
       void sub(int a,int b);
       void mul(int a,int b);
       void div(int a,int b);
}

Encapsulation => Datahiding     +              abstraction
             (private)            (interface & abstract class)
                     |
                setter&getter
```