Question 1
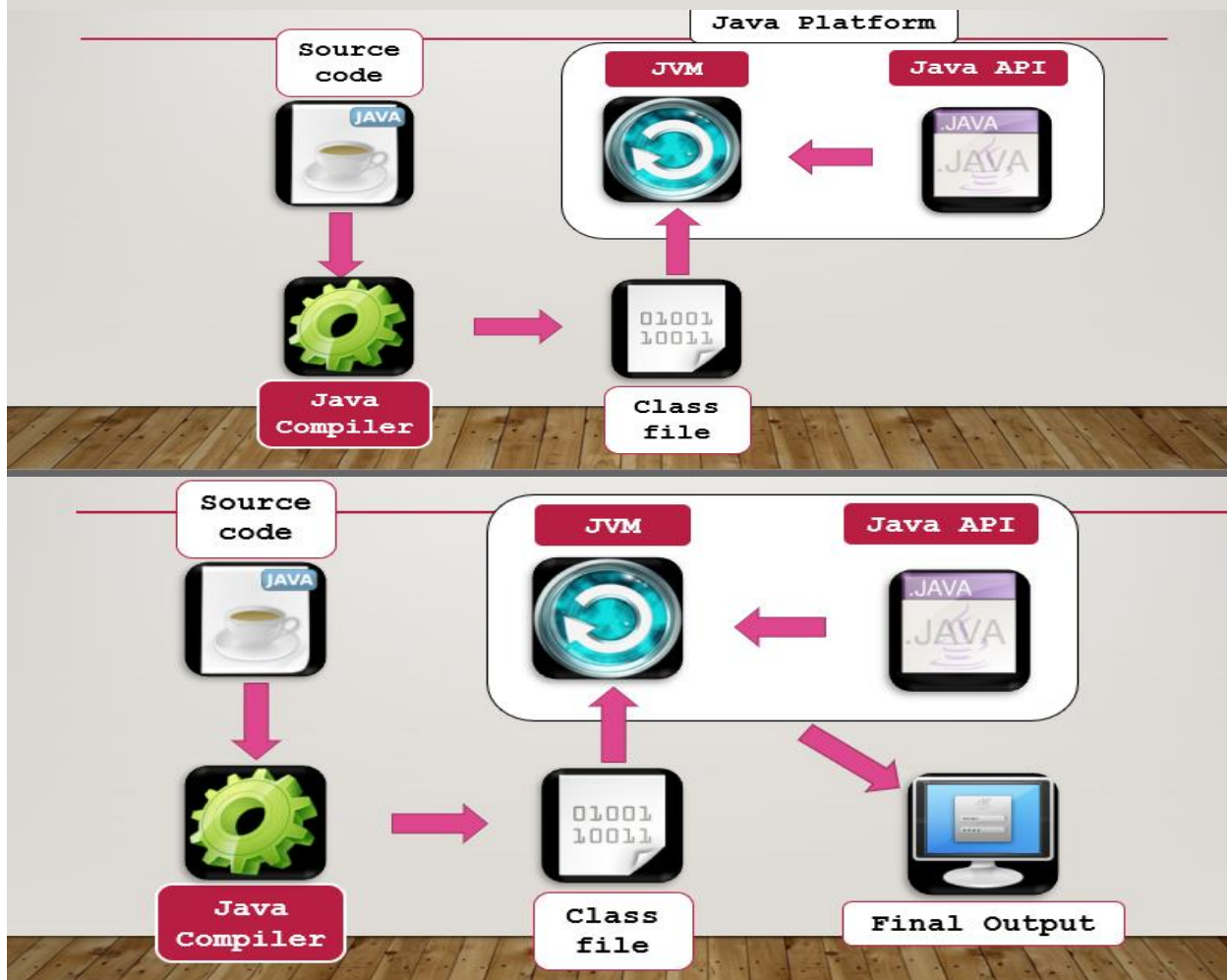

15 JAVA ARCHITECTURE CONTD…

- Source programs are written in the Java Programming Language.
- Programs are compiled into Java class files.
- Classes run in the Java Virtual Machine.
- When a Java program runs, it is assisted by other classes in the Java the Application Programming Interface (API).

What is the Output of the Following java Expression

int K=(9&4|2);

Answer: 2 ✔

Explain **Garbage Collection** in Java

Java applications obtain objects in memory as needed. It is the task of garbage collection (GC) in the Java virtual machine (JVM) to **automatically determine what memory is no longer being used** by a Java application and to recycle this memory for other uses.

The following Java code contains **Syntax Errors**. Correct the errors and rewrite the code

```
public static void main(String[] args)
{
        float r=45.78;
        sayHello();
}
void sayHello()
{
        System.out.println("Hello World");
}
```

```
public static void main(String[] args)
{
    float r 45.78f;
    sayHello();
}
public static void sayHello()
{
System.out.println("Hello World");
}
```

Write a Java program to accept a base value and a power value as integers. The program should calculate and display the power to the base value.

Sample Output:

Enter the base: 2

Enter the power: 3

Answer is 8

```java
import java.util.Scanner;

class Calculater

{

    public static void main(String[] args)

    {

        Scanner takeInput = new Scanner(System.in); //create an object for Scanner class

        System.out.print("Enter the base:"); //ask for inputs

        int base= takeInput.nextInt(); //get inputs

        System.out.print("Enter the power:");

        int power = takeInput.nextInt();

        takeInput.close(); //class getting inputs

        int answer =(int) Math.pow(base,power); //calculatiion

        System.out.println("Answer is "+answer); //print the answer

    }

}
```

The following questions are based on validating the Serial Number of a piano.

i)   The program should take the serial number as a user input. The **Serial Number** format contains **2 letters** and **4 digits**.
marks)

Example: AB1234

ii)   The program should validate the serial number and display the **year of manufacture**, based on the following criteria.

Hint: Consider the first two letters in the serial number as the letter part.

Letter Part	Year of Manufacture
IK	1991
IL	1992
IN	1994

Sample Output:

Insert a Serial Number: IK1234

Validation: Valid serial Number

Year of Manufacture: 1991

```java
package javaapplication1;

import java.util.Scanner;

class CheckSerial {

    public static void main(String[] args) {
        Scanner takeInput = new Scanner(System.in); //create an object for Scanner class
        System.out.print("Insert a Serial Number:"); //ask for inputs
        String SNumber = takeInput.nextLine(); //get inputs
        takeInput.close(); //class getting inputs
        ValidateNumber(SNumber); //call the method of ValidateNumber

    }

    public static boolean ValidateNumber(String Snumber)
    {
        String firstPart = Snumber.substring(0, 2); //get the first part of number
        String secondPart = Snumber.substring(2);//get the second part of the nubmer
        int year;

        if (firstPart.equals("IK") || firstPart.equals("IL") || firstPart.equals("IN"))
        {
          if (secondPart.length() == 4)
          {

              System.out.println("Validation : Valid Serial Number");

              switch (firstPart) {
```

```java
                    case "IK":
                        year = 1991;
                        break;
                    case "IL":
                        year = 1992;
                        break;
                    case "IN":
                        year = 1994;
                        break;
                    default:
                        year = 0;
                }
                System.out.println("Year of Manufactrue : " +year);


            } else {
                System.out.print("Validation : This is not valid Serial Number");
            }
        }
        return true;
    }
}
```

What is the difference between **Method Overloading** and **Method Overriding? Give example for each**

The most basic difference is that **overloading is being done in the same class** while for overriding base and child classes are required. Overriding is all about giving a specific implementation to the inherited method of parent class. ... private and final methods can be overloaded but they cannot be overridden.

Java Method Overloading example

```java
class OverloadingExample{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
```

Java Method Overriding example

```java
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}
```

Write a Java program for the below problem using **Compound Operator.**

A program is required to take a salary of a person and years of experiences as **command line argument** and decide the category of the person based on the age. Evaluation criteria is as follows:

Age	Category
>18	Adult
<=18	Minor

Sample Output:

Enter person's age: 8

Category is Minor

```java
package Models;


import java.util.Scanner;
public class Compoundoprerator {
    public static void main(String[] args) {
        String num1 = args[0];
        String num2 = args[1];
        int salary = Integer.parseInt(num1);
        int exYear = Integer.parseInt(num2);
        Scanner takeInput = new Scanner(System.in);
        System.out.println("Enter the your age:");
        int age = takeInput.nextInt();
        takeInput.close();
        String category = (age > 18) ? "Adult" : "Minor";
        System.out.println("salary is " + salary);
        System.out.println("ex is " + exYear);
        System.out.println("Category is " + category);
    }
}
```

Question 9.

```
public abstract class Speaker { //no instance should be able created from the speaker class

    final double noise= 12.3; //noise variable should be read only

    private double SpeakerID;//speakerID variable shoudl accessible by other classes

    double RentalPrice;


    public Speaker(double sid, double rp)

    {

        SpeakerID =sid;

        RentalPrice =rp;

    }


    abstract double getMarketValue();

    protected String TestData() //TestDate() method cannot be overriden  by the subclasses

    {

        return RentalPrice+ " "+SpeakerID;

    }


}
```

what is the output of the Following program

```
class Sample {

        int m = 20  ;
        static int n = 4 ;
        public static void main(String[] args) {
            new Sample().PlayWithVal();
            new Sample().PlayWithVal();

        }
        void PlayWithVal() {
            int m += 6;
            System.out.println("Sample n " + Sample.n);
            System.out.println("m " + m);
        n+=3  ;
        }
}
```

Output->

> **Sample n 4**
> **m 6**
>
>
> **Sample n 7**
> **m 6**

State the main **disadvantage** of a **Array** vs an **ArrayList**

- o Arrays are of fixed length. You can not change the size of the arrays once they are created. But Arraylist is a re-sizable array. And size of the arraylist is not fixed
- o You can not accommodate an extra element in an array after they are created. But in ArrayLists , Element can be inserted at or deleted from a particular position
- o Memory is allocated to an array during it's creation only, much before the actual elements are added to it.
  But in Arraylists, Arraylist class has many method to manipulate the stored objects.

Compare and contrast a **ragged array** and a **rectangular array**

```
Int[ ][ ] rectangularArray = new int[10][10];
int[ ][ ] raggedArray = new int[10][ ];

   for(int i = 0;i < 10; i++)
      raggedArray[i] = new int[i+1];
```

First one creates a 10x10 **rectangular** array, the second one creates a **ragged** array, with the 2nd dimension going from 1 to 9 elements.

There are no significant differences, but Java supports ragged arrays unlike some other languages

Write a Java code segment to do the followings

- Create an int array of size 5 and fill the array with the user input.
- Calculate and display the summation of the numbers which are greater than 50

```java
import java.util.Scanner;

class ArrayTest {

    public static void main(String[] args) {

        int[] array = new int[5]; //create new array

        int sum = 0;

        Scanner takeInput = new Scanner(System.in); //create a object for scanner class

        System.out.println("Enter 5 number :"); //ask for inputs


        for (int i = 0; i < array.length; i++) {

            System.out.println("Number " + (i + 1) + " - ");

            array[i] = takeInput.nextInt();

            if (array[i] > 50) {

                sum = sum + array[i];

            }

        }

        System.out.println("Sum of the numbers which greater than 50 is " + sum);

    }


}
```

Write a Java program which can perform the following tasks.

i)   Create an **ArrayList** Object. (Available in the java.util Package).

ii)   Insert any 5 **float numbers** between 0 and 1 to the Vector.

iii) Remove the 3rd element from the Vector.

iv) Print the remaining elements in the Vector.

```java
package javaapplication1;

import java.util.ArrayList;

public class ArrayListCheck {


  public static void main(String[] args)

  {

    ArrayList<Float> KasunList = new ArrayList<Float>(); //create an Arraylist object called KasunList

    KasunList.add(0.1f); //add 5 float numbers
    KasunList.add(0.2f);
    KasunList.add(0.3f);
    KasunList.add(0.4f);
    KasunList.add(0.5f);


    KasunList.remove(2); //remove 3rd element

    for(float i : KasunList) //print the arraylist

    {

      System.out.println(i);

    }

  }

}
```

Write a Java program to store the following price list using a **Java 2D Array.** Your program should calculate and print the total price of the price list.

Item Number	Price
1001	255
1002	135
1004	522

```java
package javaapplication1;

public class Java2DArray {


    public static void main(String[] args) {

        int[][] PriceList = {{1001, 225}, {1002, 135}, {1004, 522}};  //cretate 2d array and fill array

        int sum = 0;

        for (int i = 0; i < PriceList.length; i++) {

            sum = sum + PriceList[i][1];

        }

        System.out.println("Total price is " + sum);

    }

}
```

What is a **Checked Exception**? Give **one** example.

Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using **try-catch block** or it should declare the exception using **throws keyword**, otherwise the program will give a compilation error.

Not only that checked exception can be handled and this is user error.

Example->

 ClassNotFoundException

 IOException

 SQLException

Write the Answer to the Questions based on the Exception given below.

Note the question is case sensitive

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 7
        at DetailsArray.Change(DetailsArray.java:9)
        at DetailsArray.main(DetailsArray.java:4)
```

1. What is the Exception class: **ArrayIndexOutofBoundsException**
2. Which array index is out of bounds: **7**
3. What method throws the exception: **DetailsArray.main**
4. What file contains the method : **DetailsArray.java**
5. What line of the file throw the exception: **9**

Rewrite the Following Question After Handling the Exceptions

```
public static void main(String[] args) {
            int num = Integer.parseInt("Cartoon");
            System.out.println(num);
            String n = null;
            int le = n.length();
            System.out.println("length is "+le);

}
```

```java
public class HandlingException {
   public static void main(String[] args) {
      try {
         int num = Integer.parseInt("Cartoon");
         System.out.println("num");
         String n = null;
         int le = n.length();
         System.out.println("length is " + le);

      } catch (NumberFormatException ex) {
         System.out.println("Enter integer number");
      } catch (NullPointerException ex) {
         System.out.println("n cannot be null");
      } catch (ArrayIndexOutOfBoundsException ex) {
         System.out.println("no length in null values");
      } catch (Throwable ex) {
         System.out.println(ex.getMessage());
      }
      finally{
         System.out.println("This is the Handling Exception");
      }
   }
}
```

Answer the following Questions related to unit testing

    1. Sate Two Examples libraries for Unit Testing frameworks belongs to xUnit Family

    2. What is the Purpose of having Annotations in a Junit Testing Like @Before @After

X unit family

C# -> NUnit

Java -> Junit

Python _> PyUnit

@Beforte ->it is used to specify that method will be called before each test case

@After -> it is used to specify that method will be called after each test case.

Design a Junit Test for the Following Code Segment

```
public class LoanCal
{
    public double getTotalAmount(double amount)
    {
        return amount*1.1;
    }
}
```

```
package Test;
import org.junit.jupiter.api.*;
import Models.LoanCal;


public class Junit {

    static LoanCal c;

    double actual;

    @BeforeAll
    public static void initializeLoanCal() {
        c = new LoanCal();
    }

    @BeforeEach
    public void InitiliazegetTotalAmount() {
        actual = c.getTotalAmount(7);
    }

    @Test
    public void TestTotalAmount() {
        Assertions.assertEquals(7.7, actual);
    }

    @AfterEach
    public void cleanVarialble() {
        actual = 0;
    }

    @AfterAll
    public static void DeleteLoanCal() {
        c = null;
    }
}
```

Create an **Abstract** class called **Animal** as follows:

Private Instance Fields(S)	·	String **name**
	·	static float **price**
Constructor	Write a **constructor** to assign values (taken as parameters) to its instance fields respectively.	
Public Methods	·	String **getName()** - returns the name
	·	String **getPrice()** - returns the price
	·	abstract String **getSound()**

```
public abstract class Animal {

    private String name;

    private static float price;

    public Animal(String name, float price) {

        this.name = name;

        this.price = price;

    }

    public String getName() {

        return name;

    }

    public float getPrice() {

        return price;

    }

    public abstract String getSound();

}
```

Create sub classes **Dog** and **Cat** which **extend** the *Animal* class as given below.

Cat

Private Instance Fields(S)	int **climbdistance**
Constructor	Write a **constructor** to assign values (taken as parameters) to its instance fields respectively.
Override Public Methods	· abstract String **getSound()** - return a string "Meow Meow" · public void **climbdistance()** - prints the climb distance to the screen

```
public class Cat extends Animal {

  private int climbdistance;

  public Cat(String name, float price, int climbdistance) {

    super(name, price);

    this.climbdistance = climbdistance;

  }

  @Override

  public String getSound() {

    return "Meow Meow";

  }

  public void climbdistance() {

    System.out.print(climbdistance);

  }

}
```

Dog	
Private Instance Fields(S)	int **noofSharpTeeth**
Constructor	Write a **constructor** to assign values (taken as parameters) to its instance fields respectively.
Override Public Methods	· abstract String **getSound()** - return a string "Woof Woof" · public void **getnoofSharpTeeth()** - prints the number of sharp teeth.

```java
package Models;

public class Dog extends Animal {

   private int noofSharpTeeth;

 public Dog(String name, float price,int noofSharpTeeth) {

     super(name,price);

     this.noofSharpTeeth=noofSharpTeeth;

   }

    @Override

   public  String getSound()

   {

     return "Woof Woof";

   }

   public  void getnoofSharpTeeth()

   {

     System.out.print(noofSharpTeeth);

   }

}
```

Create **a class** called **PetShop,** which contains the **main()** method, to achieve the following tasks.

i)   Create the dog instance using the values "German Shepard", 30000 and 32.

ii)   Create the cat instance using the values "Persian cat", 10000 and 12.

iii) Print the following conversation with the use of fields and methods of the above classes.

Sample Output:

German Shepard: Hello I am a German Shepard Dog. In my language, I say hello by saying Woof Woof.

Persian cat: Hello I am a Persian cat. In my language, I say hello by saying Meow Meow.

German Shepard: I have 32 Sharp Teeth.

Persian cat: I can climb 12 feet.

German Shepard: I can be bought at Rs.30000.00.

Persian cat: I can be bought at Rs.10000.00.

```java
package Models;

public class PetShop {

    public static void main(String[] args) {

        Dog d = new Dog("German Shepard", 30000, 32);

        Cat c = new Cat("Persian Cat", 10000, 12);

        System.out.println(d.getName() + ": Hello I am a " + d.getName() + "In my language, I say hello by
saying " + d.getSound());

        System.out.println(c.getName() + ": Hello I am a " + c.getName() + "In my language, I say hello by
saying " + c.getSound());


System.out.print(d.getName() + ": I have ");

        d.getnoofSharpTeeth();

        System.out.print("Sharp Teeth.");


 System.out.print(c.getName() + ": I have ");

        c.climbdistance();

        System.out.print("feet.");


        System.out.println(d.getName() + ": I can be bought at Rs." + d.getPrice());

        System.out.println(c.getName() + ": I can be bought at Rs." + c.getPrice());

    }
}
```