Timeline

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedTimeline

Description automatically generated

**Question 1**

Graphical user interface, application

Description automatically generated

Background pattern

Description automatically generated

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.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

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

}

}

Graphical user interface, table

Description automatically generated

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) {

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;

}

}

Graphical user interface, text, application

Description automatically generated with medium confidence

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.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

package javaapplication1;

import java.util.Scanner;

class Kasun {

public static void main(String[] args) {

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

Kasun obj = new Kasun(); //create an object for Kasun class

System.out.print("Enter person's age: "); //asks for inputs

int age = takeInput.nextInt(); //getting inputs

takeInput.close(); //close inputs

obj.checkCategory(age); //call the method

}

public boolean checkCategory(int age) {

String Category = (age > 18) ? "Adult" : "Minor";

System.out.println("Category is " + Category);

return true;

}

}

**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;

}

}

Text

Description automatically generated

**Output->**

**Sample n 4**

**m 6**

**Sample n 7**

**m 6**

Table

Description automatically generated with medium confidence

* 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
* 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

* 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.

A picture containing table

Description automatically generated

**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

Text

Description automatically generated with medium confidence

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);

}

}

Graphical user interface, text, application

Description automatically generated

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);

}

}

}

Table

Description automatically generated

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);

}

}

Graphical user interface

Description automatically generated with medium confidence

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**](https://beginnersbook.com/2013/04/try-catch-in-java/) or it should declare the exception using [**throws keyword**](https://beginnersbook.com/2013/04/java-throws/), 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**

Graphical user interface, text, application, email

Description automatically generated

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**

Graphical user interface, text

Description automatically generated

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");

}

}

}

Graphical user interface, text, application

Description automatically generated

**@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.**

**X unit family**

C# -> NUnit

Java -> Junit

Python \_> PyUnit

Graphical user interface, text, application, email

Description automatically generated

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;

}

}

Table

Description automatically generated

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 static float getPrice() {

return price;

}

public abstract String getSound();

}

Table

Description automatically generated

public class Cat extends Animal {

private int climbdistance;

public Cat(int climbdistance, String name) {

super.getName();

this.climbdistance = climbdistance;

}

@Override

public String getSound() {

return "Meow Meow";

}

public void climbdistance() {

System.out.println(climbdistance );

}

}

Graphical user interface, text, application

Description automatically generated

public class Dog extends Animal {

private int noofSharpTeeth;

public Dog(int noofSharpTeeth, String name)

{

super.getName();

this.noofSharpTeeth=noofSharpTeeth;

}

@Override

public String getSound()

{

return "Woof Woof";

}

public void climbdistance()

{

System.out.println(noofSharpTeeth);

}

}

Graphical user interface, text, application, email

Description automatically generated

package Models;

public class PetShop

{

public static void main(String[] args) {

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

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

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

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

System.out.println(lucky.getName()+" I have ");

lucky.getnoofSharpTeeth();

System.out.println("Shape Teeth");

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

Chooti.getclimbdistance();

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

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

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

}

}