1 ANSWER

a)

Java is programming language but it got popularity due its object-oriented programming (OOP) language. Java is a platform-independent language.

The features of java are:

1) Platform independent:

The user can develop the program or code in one operating system and can run it in multiple operating system, because of jvm.

2) Secured :

It is secured because of the dot class file.The dot class file always saves the code in the form of byte or binary format.

3) Portable:

It is portable because the user can transfer the dot class file with the help of portable devices like usb pendrives.

4) Simple and Robust:

Java is simple because throughout the world it is written in English language and also due to some coding syntaxes.It is highly interactable to human(robust).

b)

The JVM architecture contains

**1) ClassLoader**

The class loader is a subsystem used for loading class files. It performs three major functions viz. Loading, Linking, and Initialization.

**2) Method Area**

JVM Method Area stores class structures like metadata, the constant runtime pool, and the code.

**3) Heap**

All the Objects in their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads.

**4) JVM language Stacks**

Java language Stacks store local variables, and it’s partial results. Each thread has its own JVM stack, created simultaneously as the thread is created. A new frame is created whenever a method is invoked, and it is deleted when method invocation process is complete.

**5) PC Registers**

PC register store the address of the Java virtual machine instruction which is currently executing. In Java, each thread has its separate PC register.

**6) Native Method Stacks**

Native method stacks hold the instruction of native code depends on the native library. It is written in another language instead of Java.

**7) Execution Engine**

It is a type of software used to test hardware, software, or complete systems. The test execution engine never carries any information about the tested product.

**8) Native Method interface**

The Native Method Interface is a programming framework. It allows Java code which is running in a JVM to call by libraries and native applications.

**9) Native Method Libraries**

Native Libraries is a collection of the Native Libraries (C, C++) which are needed by the Execution Engine.

c)

**Java Development Kit (JDK)** is a software development kit that develops applications in Java. The need of JDK is to convert source code into a format that the Java Runtime Environment (JRE) can execute. The JDK includes the Java Runtime Environment (JRE)

**Java Runtime environment (JRE**) is just simply a folder in which a powerful machine is present i.e. JVM

JRE = Java Virtual Machine (JVM) + Libraries to run the application

**Java Virtual Machine (JVM)** is an engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of Java Runtime Environment (JRE).

JDK = Java Runtime Environment (JRE) + Development tools

2 ANSWER

a) OOPS:

Java became popular because of Oject Oriented Programming Language.OOPS is an area where programmer can declare objects and objects according to OOPS are blue print of a class.

There are four pillars of OOPS:

1) Inheritance:

Inheritance is the ability of a class inheriting data and behaviors from another class. The main advantage of inheritance is code reusability. By using "extends" keyword. After inheriting the complete functionality of super class Sub class can access the super class methods with its reference object.

Example:

class Parent {

int a=10;

int b=20; }

class Child extends Parent {

};

Types Of Inheritance:

Java supports 3 types of inheritance.

1. Single Inheritance

2. Multi-Level Inheritance

3. Hierarchal Inheritance

1) Single Inheritance

Every java class by default inherits from “java.lang.Object” class.

By this extension only, every user object gets the behavior of real

Object. Hence every java class exhibits by default “Single Inheritance”.

2 Multi-Level Inheritance

Accessing the functionality of objects in more than one

level is called “Multi-Level Inheritance”.

Child class accessing the functionality of grand parent.

3 Hierarchal Inheritance

Sharing the properties of object to multiple child objects is

called “Hierarchal Inheritance”.

2) Polymorphism:

As the word says poly means many and morphism means forms.

Objects showing different forms at different situations/stages is called polymorphism.

Polymorphism is performed by only methods.

Simple example:

Water having many forms liquid, solid and vapour.

There are two types of Polymorphism methods:

1 Compile time Polymorphism (CTP):

When method implementation calling is going to get binded with its declarartion during compilation is called Compile time Polymorphism

Here binding is happening at the time of compilation hence it is called early binding

After compilation java is converted into class file and class file cannot be changed and hence it is called static binding

Compile time polymorphism is done by method overloading.

Method Overloading: “Writing two or more methods with the same name but with different method signature is called Method

Overloading”.Method calls are resolved at compile time.

Rules for performing overloading :

✓ Must have different argument lists.

✓ May have different return types, as long as the argument lists are

also different.

✓ May have different access modifiers.

✓ May throw different exceptions.

Example:

class Add{

static int add (int a,int b){return a+b;}

static int add (int a,int b,int c){return a+b+c;}

}

class TestOverloading1{

public static void main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}}

Output:

22

33

2) Run time Polymorphism:

When method implementation calling is going to get binded with its declarartion at the time of run time is called Compile time Polymorphism.

Method Overriding

1 “Writing two or more methods in super and sub classes with the same name

and same signature is called Method Overriding”.

2 It is also known as ‘late-binding’ or ‘run-time’ polymorphism.

3 The method present in super class is called overridden method and the method present in the sub class is called over ridding method

Example:

**class** Vehicle{

**void** run(){System.out.println("Vehicle is running");}

}

**class** Bike2 **extends** Vehicle{

  //defining the same method as in the parent class

**void** run(){System.out.println("Bike is running safely");

}

**public** **static** **void** main(String args[]){

  Bike2 obj = **new** Bike2();

  obj.run();

  }

}

Output:

Bike is running safely.

3) Encapsulation

Encapsulation is the technique of making the fields in a class

private and providing access to the fields via methods.

1 If a field is declared private, it cannot be accessed by anyone

outside the class.

2 Such that we provide security to the data from outside world

without misusing it, which is commonly known as

‘Information Hiding’ or ‘Data Hiding’.

Advantages:

* Flexibility
* Reusability
* Maintainability

Example :

public class Student{

private String name;

public String getName(){

return name;

}

public void setName(String name){

this.name=name

} }

4 Abstraction:

“Abstraction is a process of hiding the implementation details and showing only functionality to the user”, it shows only important things to the user and hides the internal details for example sending a WhatsApp message, we just type the text and send the message. We don't know the internal processing about the message delivery.

In general there are two ways to achieve Abstraction:

✓Abstract class (0 to 100%)

✓Interface (100%)

Example:

abstract class Bike {

abstract void run();

}

class Yamaha extends Bike {

void run() {

System.out.println("running safely");}

public static void main(String args[]) {

Bike obj = new Yamaha();

obj.run();

}

}

b)

**Constructor**

Constructor is a special type of method that is used to

initialize the object”.

1 Constructor is invoked at the time of object creation.

2 It constructs the values i.e. provides data for the object

that is why it is known as constructor.

Rules for creating constructor

There are basically two rules defined for the constructor.

➢ Constructor name must be same as its class name

➢ Constructor must have no return type.

Example :

Class Test{

Test(int x){

SOPLN (x); }

PSVM(){

Test t = new Test(4);

}

Output: 4

**Super Keyword**

1 It is a keyword which is predefined and non-static

variable.

2 It is used to access the complete functionality of the parent

class.

3 It must be used in non-static context.

Example :

class Tp79 {

Tp79(){

System.out.println("thor");}

public static void main(String[] args)

{

Tp79 t = new Tp79();

}

}

class Tj extends Lp {

Tj() {

super();

System.out.println("valkyrie");}

public static void main(String[] args)

{

Tj j = new Tj();

}

}

class Lp extends Tp79 {

Lp(){

System.out.println("loki");}

public static void main(String[] args)

{

Lp p = new Lp();

}

}

Output

for Tp79 class thor

for Lp class thor loki

for Tj class thor loki valkyrie

**This Keyword :**

* It is Keyword
* Non-static variable
* Used to access current class object functionality
* Must be used in non-static context.
* It holds object address.
* Must be used inside another constructor of
* same class.
* It must be first statement.

Example:

Class Test{

int i =10;

Test(int i) {

SOPLN(i);

SOPLN(this.i); }

PSVM(){

Test t = new Test(100);

}}

Output

100

10

**Final keyword:**

It is a keyword which is used to make members constant.If we declare any member as final then it is not possible to change the value or its implementation.

Inheritance is also not possible when final is declared to a class

Example:

Public class case {

final int x =10;

final static int y= 20;

PSVM() {

Final int z= 30;

Case c = new case();

c.x = 100;

y = 200;

z =300;

}};

Gives compile time error as the variables are declared with final keyword

c) Differences between abstract class and interface

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

# 

3 ANSWER

a)

Exception:

1 An abnormal event in a program is called Exception.

2 All Exceptions occur at runtime only but some are detected at

compile time and some are detected at runtime.

3 Exceptions that are checked at compile time by the java

compiler are called “Checked exceptions”.

eg: ClassNotFoundException, NoSuchMethodException, NoSuchFieldException etc.

4 Exceptions that are checked at run time by the JVM are called

“Unchecked exceptions”.

eg: ArrayIndexOutOfBoundsException, ArithmeticException, NumberFormatException etc.

Error:

An error in a program is called bug. Removing errors from

program is called debugging. There are basically three types of

errors in the Java program:

1 Compile time errors: Errors which occur due to syntax or

format is called compile time errors. These errors are detected

by java compiler at compilation time. Desk checking is solution

for compile-time errors.

2 Runtime errors: These are the errors that represent computer

inefficiency. Insufficient memory to store data or inability of

the microprocessor to execute some statement is examples to

runtime errors. Runtime errors are detected by JVM at runtime.

3 Logical errors: These are the errors that occur due to bad logic

in the program. These errors are rectified by comparing the

outputs of the program manually.

b)

An exception can be handled by the programmer where as

an error cannot be handled by the programmer.

Exception handling doesn't mean fixing an exception, We

need to provide an alternative solution for the free flow of

program.

➢Exception occurs only either inside the block or a method.

➢When exception has raised, that block or method creates

an exception object which contains the complete

information of that exception including.

Example:

try

{

……..;

……..;

……..;

}

catch(Exception e)

{

………;

}

catch(Throwable t)

{

………;

}

catch(NullpointerException ne)

{………;

}

c)

‘throw’ keyword

1 The throw keyword is mainly used to throw custom exceptions.

2 We can throw either checked or unchecked exception.

3 All methods use the throw statement to throw an exception.

4 The throw statement requires a single argument: a throwable

object.

5 Throwable objects are instances of any subclass of the Throwable class.

Example:

public class TestThrow1 {

public static void validate(int age) {

if(age<18) {

throw new ArithmeticException("Person is not eligible to vote");

}

else {

System.out.println("Person is eligible to vote!!");

}

}

public static void main(String args[]) {

validate(13);

System.out.println("rest of the code...");

}

}

‘throws’ clause

1 The "throws" keyword is used to declare an exception, It is

used to indicates what exception type may be thrown by a

method.

2 Except for methods & constructors we can’t use "throws" else

where.

3 "throws" keyword can be used only for Throwable types.

4 "throws" keyword is required only for checked exceptions.

Example:

class M{

void method()throws IOException {

throw new IOException("device error");

}

}

public class Testthrows2{

public static void main(String args[]){

try{

M m=new M();

m.method();

}

catch(Exception e){System.out.println("exception handled"); }

System.out.println("normal flow...");

}

}

4 ANSWER

a)

Multithreading in Java is a process of executing multiple threads simultaneously. Multithreading is mostly used in games, animation etc.

Example:

public class ClassA implements Runnable

{

public void run() t1.start();

{ t1.run();

for(int i=0;i<5;i++) t2.start();

System.out.println("Run method"); t2.run();

}

public static void main(String[] args)

{

ClassA a=new ClassA();

Thread t1=new Thread(a);

Thread t2=new Thread();

System.out.println("Java”):

}}

b)

Threads share the same address space. Thread is lightweight, a smallest unit of processing. Cost of communication between the thread is low. They don't allocate separate memory area so context switching between the threads takes less time than processes.

There are two ways to create a thread:

➢By extending Thread class

➢By implementing Runnable interface

Thread class:

1 Thread class is the sub class of 'Object' class and it implements Runnable interface

2 Thread class will be having constructors and methods to perform operations on thread.

Example :

public class ClassA extends Thread

{

public void run()

{

for(int i=0;i<5;i++)

System.out.println("Run method");

}

public static void main(String[] args)

{

ClassA a=new ClassA();

a.start();

System.out.println("Java is awesome");}}

Runnable interface:

1 Runnable interface will have only one method named run().

2 It is mostly recommended to use when creating thread.

3 public void run(): is used to perform action for a thread

Example:

public class ClassA implements Runnable

{

public void run() t1.start();

{ t1.run();

for(int i=0;i<5;i++) t2.start();

System.out.println("Run method"); t2.run();

}

public static void main(String[] args)

{

ClassA a=new ClassA();

Thread t1=new Thread(a);

Thread t2=new Thread();

System.out.println("Java”):

}}

c)

Synchronization

1 Synchronization in java controls multiple threads from

accessing the same shared resource in order to prevent an

inconsistent state.

2 Java Synchronization is done when we want to allow only one

thread to access the shared resource.

3 In other words Synchronization is a process of making only

one thread access a resource, where multiple threads are trying

to access the same resource, and moving all the remaining

threads in to waiting state.

Advantage:- Resolves Thread Interference & Memory

Consistency problems

Disadvantage:- Increases Thread waiting time.

Example:

class Table{

synchronized void printTable(int n){//synchronized method

for(int i=1;i<=5;i++){

System.out.println(n\*i);

try{

Thread.sleep(400);

}catch(Exception e){System.out.println(e);}

}}}

class MyThread1 extends Thread{

Table t;

MyThread1(Table t){

this.t=t; }

public void run(){

t.printTable(5); }}

class MyThread2 extends Thread{

Table t;

MyThread2(Table t){

this.t=t; }

public void run(){

t.printTable(100); }}

public class TestSynchronization2{

public static void main(String args[]){

Table obj = new Table();//only one object

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

} }

d)

Difference between Synchronization method and synchronized block

1. Scope of lock

One significant difference between the synchronized method and block is that Synchronized block generally reduces the scope of the lock. As the scope of a lock is inversely proportional to performance, it's always better to lock only a critical section of code.

2. Control over the lock

Synchronized blocks provide granular control over a lock, as you can use arbitrary any lock to provide mutual exclusion to critical section code. On the other hand, the synchronized method always locks either on the current object represented by this keyword or class level lock, if it's a static synchronized method.

3. NullPointerExcpetion

Synchronized block can throw java.lang.NullPointerException if expression provided to block as parameter evaluates to null, which is not the case with synchronized methods.

5 ANSWER

a)

class Main {

public static void main(String[] args) {

int n = 10, n1 = 0, n2 = 1;

System.out.println("Fibonacci Series till " + n + " terms:");

for (int i = 1; i <= n; ++i) {

System.out.print (n1 + ", ");

int f = n1 + n2;

n1 = n2;

n2 = f;

}}}

Output:

Fibonacci Series till 10 terms:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

b)

class Pyramid {

public static void main(String args[]){

int row=5,space=6,col=1;

for(int i=1;i<=row;i++)

{

for(int j=1;j<=space;j++)

{

System.out.print(" ");

}

for(int k=1;k<=col;k++)

{

System.out.print("\*"+" ");

}

space--;

col++;

System.out.println();

}}}

Output

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

class Pattern {

public static void main(String args[]){

int row=5,count=1;

for(int i=1;i<=row;i++){

for(int j=1;j<=count;j++){ //j<=1

System.out.print("\*");}

System.out.println();

count++;

}}}

Output

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

c)

public class StringFormatter {

public static String reverseString(String str){

char ch[]=str.toCharArray();

String rev="";

for(int i=ch.length-1;i>=0;i--){

rev+=ch[i];

}

return rev;

} }

public class TestStringFormatter {

public static void main(String[] args) {

System.out.println(StringFormatter.reverseString("my name is khan"));

System.out.println(StringFormatter.reverseString("I am sonoo jaiswal"));

}

}

Output:

nahk si eman ym

lawsiaj oonos ma

d) public class LeapYear {

public static void main(String[] args){

int year;

System.out.println("Enter an Year :: ");

Scanner sc = new Scanner(System.in);

year = sc.nextInt();

if (((year % 4 == 0) && (year % 100!= 0)) || (year%400 == 0))

System.out.println("Is a leap year");

else

System.out.println("Is not a leap year");

}

}

Output 1

Enter an Year ::

2020

Is a leap year

Output

Enter an Year ::

2017

Is not a leap year

e)

class Table{

public static void main(String[] args) {

int n=5;

for(int i=1; i <= 10; i++)

{

System.out.println(n+" \* "+i +" = "+n\*i);

}}}

Output

5 \* 1 = 5

5 \* 2 = 10

5 \* 3 = 15

5 \* 4 = 20

5 \* 5 = 25

5 \* 6 = 30

5 \* 7 = 35

5 \* 8 = 40

5 \* 9 = 45

5 \* 10 = 50