**Question 1. How is java different from C++? Explain the working of JRE.**

|  |  |  |  |
| --- | --- | --- | --- |
| **INDEX** | **COMPARISON PARAMETER** | **C++** | **JAVA** |
| 1 | Developed / Founded by | C++ was developed by Bjarne Stroustrup at Bell Labs in 1979. It was developed as an extension of the C language. | Java was developed by James Gosling at Sun Microsystems. Now, it is owned by Oracle. |
| 2 | Programming model | It has support for both procedural programming and object-oriented programming. | Java has support only for object-oriented programming models. |
| 3 | Platform dependence | C++ is platform dependent. It is based on the concept of Write Once Compile Anywhere. | Java is platform-independent. It is based on the concept of Write Once Run Anywhere. |
| 4 | Features supported | C++ supports features like operator overloading, Goto statements, structures, pointers, unions, etc. | Java does not support features like operator overloading, Goto statements, structures, pointers, unions, etc. |
| 5 | Compilation and Interpretation | C ++ is only compiled and cannot be interpreted. | Java can be both compiled and interpreted. |
| 6 | Library and Code reusability support | C++ has very limited libraries with low-level functionalities. C++ allows direct calls to native system libraries. | Java, on the other hand, has more diverse libraries with a lot of support for code reusability. In Java, only calls through the Java Native Interface and recently Java Native Access are allowed. |
| 7 | Memory Management | In C++, memory management is manual. | In Java, memory management is System controlled. |
| 8 | Type semantics | C++ is pretty consistent between primitive and object types. | In Java, semantics differs for primitive and object types. |
| 9 | Global Scope | In C++, both global and namespace scopes are supported. | Java has no support for global scope. |
| 10 | Access control and object protection | In C++, a flexible model with constant protection is available. | In Java, the model is cumbersome and encourages weak encapsulation. |

Working of JRE

The Java Run Time Environment is the crucial component of the java platform. It includes the following key elements:

1. **Java Virtual Machine (JVM)**: it is the heart of JRE. it is responsible for executing the java bytecode. the JVM loads and interprets the bytecode, manages memory and provides the runtime environment for java applications. It translates bytecode In to native machine code specific to the underlying platform, enabling platform independence.
2. **Java Standard Libraries**: the JRE includes the set of core Libraries that provides standard functionality for java applications. These libraries encompass essential classes and APIs for task such as I/O, networking, Data Structures and more.
3. **Java Runtime Environment (JRE)**: JRE includes various runtime components and resources required to run java applications, such as class libraries, configuration files and system resources.

When you run a java application, the JRE is responsible for ensuring that the application runs correctly and efficiently on your specific platform. It provides a layer of abstraction between the platform and the java application. Making it possible to run the same java code on various operating systems without modification. this is the foundation for ‘Write Once Run Anywhere’. Principle in java.

**Question 2. What is role of methods in java? How can we use ‘this’ keyword with local variables in java?**

The role of methods in java are as follows:-

1. **Modularity and Code Organization**: Methods allow you to break your code into smaller, manageable pieces or modules. This promotes a structured and organized codebase, making it easier to understand and maintain.
2. **Code Reusability**: Methods facilitate code reuse. Once you've written a method to perform a particular task, you can call that method multiple times from various parts of your program
3. **Abstraction**: Methods allow you to encapsulate a series of statements into a single unit of code
4. **Parameterization**: Methods can accept parameters (arguments) which allow you to pass data to the method for processing. Parameters make methods flexible and enable you to perform the same operation on different data.
5. **Return Values**: Methods can return values, which means they can produce results.
6. **Control Flow**: Methods define the flow of control in a program. When a method is called, the program's execution jumps to that method, performs the defined actions, and then returns control to the caller.
7. **Encapsulation and Information Hiding**: Methods can be used to encapsulate data and provide a controlled interface to access or modify that data.
8. **Testing and Debugging**: Methods allow for easier testing and debugging of code. You can test individual methods in isolation, making it simpler to identify and fix issues when they arise.
9. **Code Readability**: Well-named methods with clear and concise functionality improve the readability of your code.

If a method has a local variable with the name same as instance variable then, you can differentiate the instance variable from the local variable using this keyword.

import java.util.\*;

class Student

{

private int rollno;

private String name;

public Student(int rollno,String name)

{

this.rollno=rollno;

this.name=name;

}

public void display()

{

System.out.println("rollno: "+rollno);

System.out.println("name: "+name);

}

public static void main(String args[])

{

Student obj=new Student(1,"Varun");

obj.display();

} }

**Question 3. What is the use of String class in java? Explain any of the three methods.**

**Answer.** The String class in Java is used to represent and manipulate strings of characters. It’s a fundamental class for working with text in Java. Here’s an explanation of three commonly used methods from the String class:

Length() method: This method is used to find the length of a string, which is the number of characters in the string. For example:

String myString = “Hello, World!”;

Int length = myString.length();

charAt(int index) method: This method allows you to retrieve the character at a specific index within the string. Indices start from 0. For example:

String myString = “Java”;

Char firstChar = myString.charAt(0); // This will be ‘J’

Char thirdChar = myString.charAt(2); // This will be ‘v’

Substring(int beginIndex, int endIndex) method: This method extracts a substring from the original string. It takes two arguments: beginIndex (inclusive) and endIndex (exclusive) to specify the range of characters to include in the substring. For example:

String myString = “Hello, World!”;

String substring = myString.substring(7, 12); // This will be “World”

**Question 4. Why are Exceptions in java? How can we handle these?**

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

Here's an overview of how exception handling works in Java

Throwing Exceptions: Exceptions are typically thrown when an exceptional condition occurs within your code. You can throw exceptions explicitly using the throw statement or they can be thrown automatically by the Java runtime environment when certain errors occur (e.g., dividing by zero, trying to access an array index that doesn't exist).

java Copy code throw new SomeException("An error message");

Catching Exceptions: To handle exceptions, you use the try-catch block. Code that might throw exceptions is placed inside the try block, and you catch and handle these exceptions in the catch blocks. You can have multiple catch blocks to handle different types of exceptions.

Try {

// Code that may throw exceptions

} catch (SomeException e)

{

// Handle SomeException

} catch (AnotherException e)

{

// Handle AnotherException

}

**Question 5. What do you understand by Constructor Overloading? Explain with example how it is different from method overriding?**

Answer. Constructor overloading and method overriding are two different concepts in object-oriented programming.

Constructor Overloading:

Constructor overloading refers to the ability to define multiple constructors for a class, each with a different parameter list. These constructors have the same name but differ in the number or type of parameters they accept. The compiler differentiates between them based on the parameters, allowing you to create objects with various initializations.

Example:

Class Car {

String make;

String model;

// Constructor with no parameters

Car() {

Make = “Unknown”;

Model = “Unknown”;

}

// Constructor with two parameters

Car(String make, String model) {

This.make = make;

This.model = model;

}

}

In the example, the Car class has two constructors with different parameter lists, allowing you to create Car objects with or without specifying make and model.

Method Overriding:

Method overriding occurs in inheritance when a subclass provides a specific implementation for a method that is already defined in its superclass. The method in the subclass has the same name, return type, and parameters as the one in the superclass. This allows the subclass to replace or extend the behavior of the inherited method.

Example:

Class Animal {

Void makeSound() {

System.out.println(“Animal makes a sound”);

}

}

Class Dog extends Animal {

@Override

Void makeSound() {

System.out.println(“Dog barks”);

}

}

In this example, the Dog class overrides the makeSound method of the Animal class to provide its own implementation.

Key Differences:

1.Constructor overloading involves multiple constructors within the same class, while method overriding occurs between a superclass and its subclass.

2.Constructors have the same name (the class name) and differ in their parameter lists, whereas overridden methods have the same name, return type, and parameters.

3.Constructor overloading doesn’t require inheritance, but method overriding is a feature of inheritance.

4.Constructors are called at the time of object creation to initialize the object, while overridden methods are called to provide a specific behavior in the subclass.

**Question 6. How can we use final keyword in java? What are arrays of objects?**

Answer. . In Java, the final keyword is used to declare a constant, an unchangeable variable, or to make a class, method, or variable immutable. Here’s how it’s used:

Final Variables: When you declare a variable as final, it means its value cannot be changed after it’s initialized. For example:

Final int myConstant = 10;

Final Methods: When you declare a method as final in a class, it cannot be overridden by subclasses.

Final Classes: When you declare a class as final, it cannot be subclassed or extended.

Arrays of objects:

An array of objects in Java is simply an array where each element is an object (an instance of a class) rather than a primitive data type. You can create arrays of objects just like you would with any other data type. Here’s an example:

Class Person {

String name;

Public Person(String name) {

This.name = name;

}

}

Public class Main {

Public static void main(String[] args) {

// Create an array of Person objects

Person[] people = new Person[3];

People[0] = new Person(“Alice”);

People[1] = new Person(“Bob”);

People[2] = new Person(“Charlie”);

System.out.println(people[0].name);

}

}

In this example, people is an array of Person objects. You can access and manipulate each object in the array just like you would with any other object. Arrays of objects are useful for storing and working with collections of objects of the same class.

**Question 7. What are 2D arrays in java? Write a program to print sum of all positive integer array values.**

Answer. 2D array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns.

However, 2D arrays are created to implement a relational database look alike data structure. It provides ease of holding bulk of data at once which can be passed to any number of functions wherever required.

The syntax of declaring two-dimensional array is very much similar to that of a one-dimensional array, given as follows.

int arr[max\_rows][max\_columns];

Program:

Import java.util.Scanner;

Class Sum

{

public static void main(String args[])

{

Int i,n,sum=0;

Scanner scn=new Scanner(System.in);

System.out.println(“enter array size:”);

n=scn.nextInt();

int A[]=new int[n];

System.out.println(“ënter array elements:”)

for(i=0;i<n;i++)

{

A[i]=scn.nextInt();

If(a[i]>0)

Sum=sum+A[i];}

}

System.out.println(“sum=”+sum);

}

}

**Question 8. Write a program to show Working of static and non- static blocks in java.**

Answer. Working of Static and Non-Static Blocks:

In Java, you can use static and non-static blocks to execute code before the class or instance is initialized. Here’s an example program demonstrating their usage:

Class BlockDemo {

Static {

System.out.println(“Static Block – This runs when the class is loaded.”);

}

{

System.out.println(“Non-Static Block – This runs when an instance is created.”);

}

BlockDemo() {

System.out.println(“Constructor – This runs when an instance is constructed.”);

}

}

Public class Main {

Public static void main(String[] args) {

System.out.println(“Creating object 1:”);

BlockDemo obj1 = new BlockDemo();

System.out.println(“Creating object 2:”);

BlockDemo obj2 = new BlockDemo();

}

}

In this example, the static block runs when the class is loaded, and the non-static block (also known as an instance initializer block) runs when an instance is created, just before the constructor.

**Question 9. What are command line arguments in java? Write a program to show the use of wrapper class for printing the sum of all positive integers passed through command line arguments.**

Answer. Command line arguments in java allow us to provide input parameters to a java program when you run it from the command line or terminal. These arguments are specified after the name of the java class you are running. Command -line arguments are stored in the args parameter of the main method of your program.

Program:

import java.util.Scanner;

class Command

{

public static void main(String args[])

{

int n,i,sum=0;

for(i=0;i<args.length;i++)

{

int n=Integer.parseInt(args[i]);

if(n>0)

sum=sum+n;

}

System.out.println(“sum=”+sum);

}

}

**Question 10. What is the use of char data type in java? Write a program to override any of object class methods.**

Answer. . In Java, the char data type is used to represent a single 16-bit Unicode character. It can store any character from the Unicode character set, which includes characters from various languages, symbols, and special characters.

To override a method from the Object class, you can choose from methods like toString(), equals(), or hashCode(). Here’s an example of overriding the toString() method for a custom class:

Class CustomChar {

Private char value;

Public CustomChar(char value) {

This.value = value;

}

Public String toString() {

Return “CustomChar: “ + value;

}

}

Public class CharOverrideExample {

Public static void main(String[] args) {

CustomChar customChar = new CustomChar(‘A’); System.out.println(customChar.toString());

}

}

In this example, we’ve created a class CustomChar with a char field and overridden the toString() method to provide a custom string representation. When you call toString() on a CustomChar object, it will return “CustomChar: A” in this case.

You can similarly override other methods from the Object class to provide custom behavior based on your requirements.