Question no:01-(a)How is java different from C++?Explain the working of JRE.

Solution:-

| **Sl. No.** | **Parameters** | **Java** | **C++** |
| --- | --- | --- | --- |
| 1. | Founder | Java was developed by James Gosling at Sun Microsystems. | C++ was developed by Bjarne Stroustrup at Bell Labs in 1979 as an extension of the C language. |
| 2. | First Release | On May 23, 1995 | In October 1985 |
| 3. | Stable Release | Java SE 18 was released on 22 March 2022 | C++20  released on 15th December 2020 |
| 4. | Official Website | oracle.com/java | isocpp.org |
| 5. | Influenced By: | Java was Influenced by Ada 83, Pascal, C++, [C#](https://www.geeksforgeeks.org/csharp-programming-language/), etc. languages. | C++ was Influenced by Influenced by Ada, ALGOL 68, C, ML, Simula, Smalltalk, etc. languages. |
| 6. | Influenced to: | Java was influenced to develop BeanShell, C#, Clojure, Groovy, Hack, J#, Kotlin, PHP, Python, Scala, etc. languages. | C++ was influenced to develop C99, Java, JS++, Lua, Perl, PHP, Python, Rust, Seed7, etc. languages. |
| 7. | Platform Dependency | Platform-independent, Java bytecode works on any operating system. | Platform dependent, should be compiled for different platforms. |
| 8. | Portability | It can run on any OS hence it is portable. | C++ is platform-dependent. Hence it is not portable. |
| 9. | Compilation | Java is both Compiled and Interpreted Language. | C++ is a Compiled Language. |
| 10. | Memory Management | Memory Management is System Controlled. | Memory Management in C++ is Manual. |
| 11. | Virtual Keyword | It doesn’t have a Virtual Keyword. | It has Virtual keywords. |
| 12. | Multiple Inheritance | It supports only single inheritance. Multiple inheritances are achieved partially using interfaces. | It supports both single and multiple Inheritance. |
| 13. | Overloading | It supports only method overloading and doesn’t allow operator overloading. | It supports both method and operator overloading. |
| 14. | Pointers | It has limited support for pointers. | It strongly supports pointers. |
| 15. | Libraries | It doesn’t support direct native library calls but only Java Native Interfaces. | It supports direct system library calls, making it suitable for system-level programming. |
| 16. | Libraries | Libraries have a wide range of classes for various high-level services. | C++ libraries have comparatively low-level functionalities. |
| 17. | Documentation Comment | It supports documentation comments (e.g., /\*\*.. \*/) for source code. | It doesn’t support documentation comments for source code. |
| 18. | Thread Support | Java provides built-in support for multithreading. | C++ doesn’t have built-in support for threads, depends on third-party threading libraries. |
| 19. | Type | Java is only an object-oriented programming language. | C++ is both a procedural and an object-oriented programming language. |
| 20. | Input-Output mechanism | Java uses the (System class): **System.in** for input and **System.out** for output. | C++ uses **cin** for input and **cout** for an output operation. |
| 21. | goto Keyword | Java doesn’t support goto Keyword | C++ supports goto keyword. |
| 22. | Structures and Unions | Java doesn’t support Structures and Unions. | C++ supports Structures and Unions. |
| 23. | Parameter Passing | Java supports only the Pass by Value technique. | C++ supports both Pass by Value and pass by reference. |
| 24. | Inheritance Tree | All classes in Java are subclasses of the Object class, hence Java only ever follows a single inheritance tree. | A fresh inheritance tree is always created in C++. |
| 25. | Global Scope | It supports no global scope. | It supports both global scope and namespace scope. |
| 26. | Object Management | Automatic object management with garbage collection. | It supports manual object management using new and delete. |
| 27. | Call by Value and Call by reference | Java supports only call by value. | C++ both supports call by value and call by reference. |
| 28. | Hardware | Java is not so interactive with hardware. | C++ is nearer to hardware. |
| 29. | Language Used for | Internet and Android games, Mobile applications, Healthcare and research computation, Cloud applications, Internet of Things (IoT) devices, etc. | Game engines, Machine learning, Operating systems, Google Search Engine, Web browsers, Virtual Reality (VR), Game development, Medical technology, Telecommunications, Databases, etc. |
| 30. | Application built | Wikipedia, LinkedIn, Android OS, Uber, and Minecraft, | Mozilla Firefox, Amazon, Apple OS, Spotify, Adobe Photoshop, and Youtube. |

A Java runtime environment runs on top of the OS and provides additional Java-specific resources. It abstracts the OS and creates a consistent platform on which to run and deploy Java applications. The 3 components of the JRE work together within this environment to successfully run a Java application.

The Java class libraries contain collections of prewritten code that can be called when needed. The class loader connects class libraries with the JVM and loads them into the JVM while the program is running. The JVM then executes the code.

QuestionNo:01-(b)What is the role of methods in java?How can we use’this’keyword with local variables in java?

Solution:- n Java, methods play a crucial role in defining the behavior of classes and objects. Here's an overview of the role of methods in Java:

Encapsulation: Methods are used to encapsulate behavior and functionality within a class. They allow you to define what a class can do and how it does it. This helps in hiding the internal implementation details of a class and exposing a clean interface for interacting with objects of that class.

Code Reusability: Methods promote code reusability. You can define a method once and call it multiple times from different parts of your program, reducing code duplication and making your code easier to maintain.

Abstraction: Methods allow you to abstract away complex operations into a single, easy-to-use function. This simplifies the usage of a class and makes it more understandable.

Modularity: Methods enable the decomposition of a complex problem into smaller, manageable pieces. Each method can focus on a specific task, promoting modular and maintainable code.

Polymorphism: Methods are essential for implementing polymorphism, allowing you to define methods with the same name in different classes or interfaces. This enables method overriding and interface implementation.

Use of this keyword in java:-

The this keyword in Java is used to refer to the current instance of the class. However, it is not used with local variables; it is primarily used with instance variables and instance methods. Here's how it is used:

1.Accessing Instance Variables: You can use this to differentiate between instance variables and local variables when they have the same name. For example:

public class MyClass {

int myVar; // instance variable

public void setVar(int myVar) {

this.myVar = myVar; // using this to refer to the instance variable

}

}

2. Constructor Chaining: You can use this to call one constructor from another constructor within the same class. This is called constructor chaining. For example:

public class MyClass {

int value;

public MyClass() {

this(0); // calling another constructor with an argument

}

public MyClass(int value) {

this.value = value;

}

}

Question.No.02-(a)What is the use of String class in java? Explain any of its three methods.

Solution:- The String class in Java is a fundamental class that represents a sequence of characters. It is widely used for handling and manipulating textual data in Java applications. The String class is part of the Java Standard Library and provides various methods for working with strings.

Here are three commonly used methods of the String class:

1. length() Method:

The length() method is used to determine the number of characters in a String. It returns an integer representing the length of the string. This method is useful for checking the size of a string or iterating over its characters.

Esxample:-

String myString = "Hello, World!";

int length = myString.length();

System.out.println("Length of the string: " + length); // Output: 13

2. charAt(int index) Method:

The charAt(int index) method allows you to retrieve the character at a specific index within the string. The index is zero-based, meaning the first character is at index 0, the second character at index 1, and so on. This method is useful for accessing individual characters in a string.

Example:

java

Copy code

String myString = "Java";

char firstChar = myString.charAt(0); // 'J'

char thirdChar = myString.charAt(2); // 'v'

3. substring(int beginIndex) Method:

The substring(int beginIndex) method is used to extract a portion of the string starting from the specified beginIndex and extending to the end of the string. It returns a new string that is a substring of the original string. You can also use an overloaded version of this method to specify both beginIndex and endIndex to extract a substring with a specific range.

Example:

java

Copy code

String myString = "Hello, World!";

String subString = myString.substring(7); // "World!"

Question.No:-02(b)Why are exceptions in java?How can we handle these?

Solution:- Exceptions in Java are a mechanism for handling unexpected or erroneous situations that may occur during program execution. These situations, often referred to as "exceptional" or "error" conditions, can include things like:

1.Runtime Errors: These are errors that occur during program execution, such as dividing by zero, accessing an array out of bounds, or attempting to use a null reference.

2.I/O Errors: When reading or writing files, network operations, or other input/output operations, exceptions can occur if the file is not found, permissions are inadequate, or the network connection is lost.

3.User-Defined Errors: Java allows developers to define their custom exceptions to handle specific error conditions in their programs.

The purpose of exceptions is to provide a way to gracefully handle these error conditions rather than causing the program to crash or exhibit undefined behavior. Exception handling allows developers to:

1.Detect Errors: Exceptions provide a mechanism to detect errors or exceptional situations in the program.

2.Separate Error Handling: Exception handling allows you to separate the error-handling logic from the normal program flow, making the code more organized and maintainable.

3.Propagate Errors: Exceptions can be propagated up the call stack, allowing higher-level code to handle or log the error while lower-level code doesn't need to deal with it directly.

In Java, exception handling is done through the use of try, catch, and finally blocks:

* The try block contains the code that might throw an exception.
* The catch block is used to catch and handle exceptions. Multiple catch blocks can be used to handle different types of exceptions.
* The finally block (optional) is used for code that should always be executed, whether an exception occurs or not.

Here's an example of how to handle an exception in Java:

try {

// Code that might throw an exception

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

// Handle the exception

System.out.println("An error occurred: " + e.getMessage());

} finally {

// Code that will always be executed

System.out.println("Execution completed.");

}

In this example, the try block attempts to divide by zero, which throws an ArithmeticException. The catch block catches this exception, allowing you to handle it (in this case, printing an error message). Finally, the finally block is used to ensure that certain code is executed, regardless of whether an exception occurred.

Question.No-03(a)-What do you understand by Constructor Overloading?Explain with example how is it different from method overriding?

Solution:- Constructor overloading and method overriding are two different concepts in object-oriented programming, particularly in Java. Let's explore each concept and provide examples to illustrate the differences between them.

Constructor Overloading:

Constructor overloading refers to the practice of defining multiple constructors within a class, each with a different set of parameters. This allows objects of the class to be instantiated with different initializations, making it more flexible and accommodating different use cases.

Example of constructor overloading:

public class MyClass {

private int value;

// Constructor with no parameters

public MyClass() {

value = 0;

}

// Constructor with one parameter

public MyClass(int value) {

this.value = value;

}

}

In this example, the MyClass class has two constructors: one with no parameters and another with an int parameter. Depending on how you create an object of this class, you can choose which constructor to use:

MyClass obj2 = new MyClass(42); // Uses the constructor with an integer parameter

Method Overriding:

Method overriding occurs in inheritance when a subclass provides its own implementation of a method that is already defined in its superclass. This allows the subclass to change the behavior of the inherited method while maintaining the same method signature (name, return type, and parameters). Method overriding is typically used to achieve polymorphism and is closely associated with inheritance.

Example of method overriding:

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 makeSound method in the Dog class overrides the method of the same name in the Animal class. When you call makeSound on a Dog object, it will use the overridden method in the Dog class, producing the output "Dog barks."

Key Differences:

Purpose:

Constructor overloading is used to create multiple constructors with different parameter sets for object initialization.

Method overriding is used to change the behavior of a method inherited from a superclass in a subclass.

Associated with:

Constructor overloading is not directly related to inheritance and polymorphism.

Method overriding is used to achieve polymorphism and is closely associated with inheritance.

Name and Signature:

In constructor overloading, constructors have the same name (the class name) but different parameter lists.

In method overriding, methods have the same name and the same parameter list as the method they override.

Return Type:

Constructors do not have a return type.

Overridden methods must have the same return type as the method they override.

In summary, constructor overloading is about defining multiple constructors with different parameter lists within a single class, whereas method overriding is about providing a new implementation for a method that is inherited from a superclass in a subclass.

Question.No.03-(b)-How can we use final keyword in java?What are arrays of objects?

Solution:- The final keyword in Java is used to denote that a variable, method, or class cannot be modified or extended, depending on where it's applied.

Here's how you can use the final keyword in different contexts:

Final Variables:

A final variable is a constant, and its value cannot be changed once it's assigned.

Final variables must be initialized when they are declared or in the constructor of the class.

They are typically written in uppercase with underscores separating words (e.g., static final int MAX\_VALUE = 100;).

Example:

public class MyClass {

final int myConstant = 42;

}

Final Methods:

When a method is marked as final in a class, it cannot be overridden by any subclass.

This is often used to prevent further modification of a specific method's behavior in a subclass.

Example:

public class Parent {

public final void finalMethod() {

}

}

Final Classes:

When a class is marked as final, it cannot be extended (i.e., it cannot be a superclass for other classes).

This is often used when you want to prevent inheritance for security or design reasons.

Example:

public final class FinalClass {

}

Now, let's discuss "arrays of objects." An array of objects in Java is an array that holds references to objects rather than the objects themselves. Each element of the array contains a reference to an instance of a particular class. This allows you to store and manipulate multiple objects of the same class in a structured way.

Here's an example of how to create an array of objects:

class Student {

private String name;

private int age;

public Student(String name, int age) {

this.name = name;

this.age = age;

}

public void display() {

System.out.println("Name: " + name + ", Age: " + age);

}

}

public class ArrayOfObjectsExample {

public static void main(String[] args) {

Student[] students = new Student[3];

students[0] = new Student("Alice", 20);

students[1] = new Student("Bob", 22);

students[2] = new Student("Carol", 21);

for (Student student : students) {

student.display();

}

}

}

In this example, we create an array of Student objects, and each element of the array holds a reference to a Student object. You can then access and manipulate these objects using the array index.

Arrays of objects are useful for storing and processing collections of objects of the same class, making it easier to manage and work with groups of related data.

Question.No.4-(a)-What are 2d arrays in java?write a program to print sum of all positive integer array values.

Solution:- A 2D array in Java is an array of arrays. It's a data structure that can be thought of as a grid or a matrix, where elements are organized into rows and columns. Each element in a 2D array is identified by its row and column indices. You can use 2D arrays to represent data that has a two-dimensional structure, like a table.

Here's an example of a 2D array in Java:

int[][] twoDArray = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

To calculate the sum of all positive integer values in a 2D array, you can iterate through the array and add the positive values to a running total. Here's a program to do that:

public class SumOfPositiveValuesIn2DArray {

public static void main(String[] args) {

int[][] twoDArray = {

{1, -2, 3},

{4, 5, -6},

{7, -8, 9}

};

int sum = 0;

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

for (int j = 0; j < twoDArray[i].length; j++) {

int value = twoDArray[i][j];

if (value > 0) {

sum += value;

}

}

}

System.out.println("Sum of positive values in the 2D array: " + sum);

}

}

In this program, we first define a 2D array named twoDArray. Then, we iterate through the array using nested loops, checking each element's value. If the value is positive (greater than 0), we add it to the sum variable. Finally, we print the sum of positive values in the array.

Question.No.04-(b)-Write down the difference between Instance variable and class variable with suitable variable and class variable with suitable example.Write a program to show working of static and non static blocks in java.

Solution:-

| **Instance Variable** | **Class Variable** |
| --- | --- |
| It is a variable whose value is instance-specific and now shared among instances. | It is a variable that defines a specific attribute or property for a class. |
| These variables cannot be shared between classes. Instead, they only belong to one specific class. | These variables can be shared between class and its subclasses. |
| It usually reserves memory for data that the class needs. | It usually maintains a single shared value for all instances of class even if no instance object of the class exists. |
| It is generally created when an instance of the class is created. | It is generally created when the program begins to execute. |
| It normally retains values as long as the object exists. | It normally retains values until the program terminates. |
| It has many copies so every object has its own personal copy of the instance variable. | It has only one copy of the class variable so it is shared among different objects of the class. |
| It can be accessed directly by calling variable names inside the class. | It can be accessed by calling with the class name. |
| These variables are declared without using the static keyword. | These variables are declared using the keyword static. |
| Changes that are made to these variables through one object will not reflect in another object. | Changes that are made to these variables through one object will reflect in another object. |

**Program to Show Static and Non-Static Blocks:**

public class BlockExample {

static {

System.out.println("Static Block 1 (executed when class is loaded)");

}

{

System.out.println("Non-Static Block 1 (executed for each object)");

}

public BlockExample() {

System.out.println("Constructor (executed when an object is created)");

}

static {

System.out.println("Static Block 2 (executed when class is loaded)");

}

{

System.out.println("Non-Static Block 2 (executed for each object)");

}

public static void main(String[] args) {

System.out.println("Main Method (executed when the program starts)");

BlockExample obj1 = new BlockExample();

BlockExample obj2 = new BlockExample();

}

}

Question.No-05-(a)-What are command line arguments in java?Write a program to show the use of wrapper class for printing sum of all positive integers as passed through command line arguments.

Solution:-

Command line arguments in Java allow you to pass data or parameters to a Java program when it is executed from the command line or terminal. These arguments are passed as strings and are accessible within the main method of the Java program via the args parameter, which is an array of strings.

Here's a simple Java program that demonstrates the use of command line arguments to print the sum of all positive integers passed as arguments:

public class CommandLineArgumentsSum {

public static void main(String[] args) {

int sum = 0;

for (String arg : args) {

try {

int num = Integer.parseInt(arg);

if (num > 0) {

sum += num;

}

} catch (NumberFormatException e) {

// Handle non-integer arguments (ignore or report the error)

System.err.println("Skipping non-integer argument: " + arg);

}

}

System.out.println("Sum of positive integers: " + sum);

}

}

In this program, we use the args parameter to access the command line arguments. We iterate through each argument, attempting to parse it as an integer using Integer.parseInt(). If the parsing is successful and the parsed number is positive, we add it to the sum variable. Any non-integer arguments are skipped and can be handled as needed.

Question.No.05-(b)-What is the use of char data type in java?Write a program to override any of object class method.

Solution:-

The char data type in Java is used to represent a single 16-bit Unicode character. It is used to store characters, such as letters, digits, and symbols, and is integral in handling text and characters in Java programs. Some common uses of the char data type include reading and processing text data, character manipulation, and text formatting.

Here's a simple program that demonstrates the use of the char data type to display a character:

public class CharExample {

public static void main(String[] args) {

char myChar = 'A'; // Declaration and initialization of a char variable

System.out.println("Character: " + myChar);

}

}

In this program, we declare a char variable named myChar and initialize it with the character 'A'. The program then prints this character to the console.

Regarding overriding an Object class method, one of the most commonly overridden methods is the toString() method. The toString() method is a part of the Object class and is responsible for returning a string representation of an object.

Here's an example of overriding the toString() method in a custom class:

class MyObject {

private int value;

public MyObject(int value) {

this.value = value;

}

public String toString() {

return "MyObject with value: " + value;

}

}

public class ObjectToStringOverride {

public static void main(String[] args) {

MyObject obj = new MyObject(42);

System.out.println(obj); // This will call the overridden toString method

}

}

In this example, the MyObject class overrides the toString() method. When you create an instance of MyObject and print it to the console, the overridden toString() method is called, and it returns a customized string representation of the object.

Overriding toString() is often useful for providing meaningful and descriptive representations of your custom objects, making it easier to debug and understand the state of objects when you print them. You can override other methods from the Object class (e.g., equals, hashCode, or finalize) to customize the behavior of your objects as needed.