* **Name: Jyoti Chauhan**
* **Section: E**
* **Roll number -56**
* **Java Assignment**

Q1.How is java different from C++? Explain the working of JRE.

SOL: Java is a regular-purpose, category-based, object-oriented programming language with low **implementation** dependencies. It is built using a virtual machine, which allows developers to maintain the security of the code and make it highly portable. Java does not support pointers, unions, operator overloading, and structure, while C++ supports pointers, unions, operator overloading, and structure.

C++ is a better option for system-level programming since it offers direct calls to libraries, while Java does not allow it. Java is built using a virtual machine, which allows developers to maintain the security of the code and make it highly portable.C++ relies solely on a compiler, while Java employs a compiler and an interpreter. In comparison to Java, C++ supports operator overloading and multiple inheritances. C++ is more related to hardware, while Java has built-in support for the internet. C++ uses header files to include different libraries in the program.

JRE(Java Runtime Environment) is an installation package that provides an environment to only run(not develop) the java program(or application)onto your machine. JRE is only used by those who only want to run Java programs that are end-users of your system.

Q2.What is the role of methods in java? How can we use ‘this’ keyword with Local variables in java?

SOL: In Java, methods are a fundamental building block of a class. They define the behavior or actions that objects of the class can perform. Here are the main roles of methods in Java:

1. Encapsulation: Methods encapsulate the behavior of an object, allowing you to hide the implementation details and only expose a well-defined interface. This promotes the principle of data hiding and abstraction.
2. Reusability: Methods can be reused across different parts of the program or in other programs. You can call a method multiple times, avoiding code duplication.
3. Modularity: Methods help break down a large program into smaller, more manageable pieces. Each method can focus on a specific task, making the code easier to understand and maintain.
4. Parameter Passing: Methods can accept parameters (input values) and return values. This allows you to pass data into a method, perform operations, and get results back.

Regarding the use of the “this” keyword with local variables in Java:

The “this” keyword in Java is primarily used to refer to the current instance of the class. It can be used with instance variables and methods to disambiguate when there’s a naming conflict between instance variables and method parameters or local variables.

Here’s how we can use “this” with local variables:

```java

Public class MyClass {

Private int myVar; // instance variable

Public void setMyVar(int myVar) {

This.myVar = myVar; // Using “this” to distinguish the instance variable from the parameter

}

Public void printMyVar() {

System.out.println(“myVar = “ + this.myVar); // Using “this” to access the instance variable

}

}

```

In the example above, we use “this” to clarify that we’re referring to the instance variable “myVar” rather than the method parameter “myVar” within the `setMyVar` method. Similarly, we use “this” to access the instance variable in the `printMyVar` method. This helps avoid confusion when there are naming conflicts.

Q3.What is the use of the String class in Java? Explain any of its three methods.

SOL: The `String` class in Java is a fundamental class used to represent and manipulate sequences of characters. It provides a wide range of methods for working with strings. Here are explanations of three commonly used methods of the `String` class:

1. \*\*`length()` Method\*\*: This method returns the length of the string, which is the number of characters in the string. For example:

```java

String myString = “Hello, World!”;

Int length = myString.length(); // length will be 13

```

1. \*\*`charAt(int index)` Method\*\*: This method returns the character at the specified index within the string. The index is a zero-based position, so the first character is at index 0. For example:

```java

String myString = “Java”;

Char character = myString.charAt(1); // character will be ‘a’

```

1. \*\*` substring(int startIndex, int endIndex)` Method\*\*: This method extracts a portion of the string between the specified `startIndex` (inclusive) and `endIndex` (exclusive) positions. It can be used to create substrings from the original string. For example:

```java

String myString = “Programming”;

String subString = myString.substring(3, 7); // subString will be “gram”

```

These are just a few examples of the many methods available in the `String` class for working with strings in Java. Strings are widely used in Java for tasks like text processing, manipulation, and data storage.

Q4:Why are Exceptions in java? How can we handle these?

SOL: Exceptions in Java are used to handle exceptional or erroneous situations that can occur during the execution of a program. These situations are typically unexpected and can disrupt the normal flow of a program. Java uses a structured way to handle exceptions for better program reliability.

Here’s why exceptions are used in Java:

1. \*\*Error Handling\*\*: Exceptions provide a mechanism to handle errors and exceptional conditions gracefully rather than letting them crash the program. This helps in maintaining the program’s stability and reliability.
2. \*\*Separation of Concerns\*\*: Exceptions allow you to separate the error-handling code from the normal flow of your program. This makes the code more organized and easier to maintain.
3. \*\*Propagating Information\*\*: Exceptions can carry information about the nature of the error, which can be useful for debugging and logging.

To handle exceptions in Java, you can use a combination of the `try`, `catch`, and `finally` blocks:

* + \*\*`try` Block\*\*: You enclose the code that may throw an exception within a `try` block.
  + \*\*`catch` Block\*\*: After the `try` block, you can have one or more `catch` blocks that specify the type of exception to catch and the code to execute when that exception is thrown.
  + \*\*` finally` Block (Optional): You can include a `finally` block after the `catch` block(s) to specify code that should be executed regardless of whether an exception occurred. This block is often used for cleanup operations.

Here’s an example of exception handling in Java:

```java

Try {

// Code that may throw an exception

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

} catch (ArithmeticException ex) {

// Handle the exception

System.out.println(“An error occurred: “ + ex.getMessage());

} finally {

// This block will be executed whether there’s an exception or not

System.out.println(“Cleanup operations”);

}

```

In this example, if an `ArithmeticException` is thrown (division by zero), it will be caught In the `catch` block, and you can handle it as needed. The `finally` block ensures that cleanup operations are executed, regardless of whether an exception occurred or not.

Java provides a hierarchy of exception classes, allowing you to catch specific exceptions and handle them appropriately. You can also create custom exception classes by extending the `Exception` class to handle application-specific error scenarios.

Q5:What do you understand by Constructor Overloading? Explain with Example how it is different from method overiding?

SOL:Constructor overloading and method overriding are two different concepts in Java.

\*\*Constructor Overloading\*\*:

Constructor overloading refers to the practice of defining multiple constructors in a class, each with a different set of parameters. The constructors have the same name but differ in the number or types of parameters they accept. This allows objects of the class to Be initialized in different ways, providing flexibility in creating instances of the class.

Example of constructor overloading:

```java

Class MyClass {

Int value;

// Constructor with no parameters

MyClass() {

Value = 0;

}

// Constructor with one parameter

MyClass(int v) {

Value = v;

}

}

```

In this example, we have two constructors in the `MyClass` class. One takes no parameters, and the other takes an integer parameter. You can create instances of `MyClass` using either constructor, depending on your needs.

\*\*Method Overriding\*\*:

Method overriding occurs in the context of inheritance, particularly when you have a subclass that 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 method in the superclass. This allows the subclass to provide its own behavior while maintaining the method signature.

Example of method overriding:

```java

Class Vehicle {

Void start() {

System.out.println(“Vehicle is starting.”);

}

}

Class Car extends Vehicle {

@Override

Void start() {

System.out.println(“Car is starting.”);

}

}

```

In this example, the `Car` class overrides the `start` method of the `Vehicle` class. When you call `start` on a `Car` object, it prints “Car is starting,” whereas calling `start` on a `Vehicle` object prints “Vehicle is starting.”

Q6.How can we use final keyword in java? What are arrays of objects ?

SOL:The `final` keyword in Java is used to indicate that something is not allowed to be changed or extended further. It can be applied to variables, methods, and classes.

Here’s how you can use the `final` keyword:

1. \*\*Final Variables\*\*: When a variable is declared as `final`, its value cannot be changed once it’s assigned. This is useful For creating constants.

```java

Final int MY\_CONSTANT = 42;

// You cannot modify MY\_CONSTANT later.

```

1. \*\*Final Methods\*\*: When a method is marked as `final` in a class, it means it cannot be overridden by subclasses. This is a way to enforce a specific implementation for a method.

```java

Class Parent {

Final void finalMethod() {

// This method cannot be overridden in subclasses.

}

}

```

1. \*\*Final Classes\*\*: When a class is declared as `final`, it means it cannot be extended by other classes. You cannot create a subclass of a `final` class.

```java

Final class MyFinalClass {

// This class cannot be extended by other classes.

}

```

Now, let’s talk about “arrays of objects.” In Java, you can create arrays that store objects, just like you would create arrays for primitive data types. An array of objects is essentially an array where each element is a reference to an object. Here’s an example:

```java

Class Person {

String name;

Person(String name) {

This.name = name;

}

}

Public class ObjectArrayExample {

Public static void main(String[] args) {

// Creating an array of Person objects

Person[] people = new Person[3];

// Initializing array elements with Person objects

People[0] = new Person(“Alice”);

People[1] = new Person(“Bob”);

People[2] = new Person(“Charlie”);

// Accessing elements in the array

System.out.println(people[1].name); // Outputs “Bob”

}

}

```

In this example, we create an array `people` that can store references to `Person` objects. We initialize the array elements with instances of the `Person` class. You can access and manipulate objects in the array just like any other array type.

Q7:What are 2d arrays in java? Write a program to print sum of all positive integer array values.

SOL:In Java, a 2D array is essentially an array of arrays, forming a grid-like structure. You can think of it as a table with rows and columns. Each element in a 2D array is identified by its row and column index.

Here’s a simple Java program to print the sum of all positive integer values in a 2D array:

```java

Public class SumPositiveValuesIn2DArray {

Public static void main(String[] args) {

Int[][] array = {

{1, 2, -3},

{4, -5, 6},

{7, 8, 9}

};

Int sum = 0;

For (int row = 0; row < array.length; row++) {

For (int col = 0; col < array[row].length; col++) {

If (array[row][col] > 0) {

Sum += array[row][col];

}

}

}

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

}

}

```

In this program, we define a 2D array called `array` and then use nested loops to iterate through all elements in the array. We check if each element is positive (greater than 0) and add it to the `sum` if it is. Finally, we print the sum of positive values.

Q8:Write down the difference between Instance variable and class variable with suitable example. Write a program to show working of static and non static blocks in java.

SOL:Instance variables and class variables in Java serve different purposes and have distinct characteristics:

1. Instance Variable:
   * An instance variable is associated with an instance (object) of a class.
   * Each instance of the class has its own copy of the instance variable.
   * It is defined without the `static` keyword and is typically used to store state specific to each object.
   * Instance variables are created when an object is instantiated and destroyed when the object is garbage collected.

Example:

```java

Public class Car {

String model; // instance variable

Public Car(String model) {

This.model = model;

}

}

```

1. Class Variable (Static Variable):
   * A class variable is associated with the class itself rather than with instances of the class.
   * There is only one copy of the class variable for the entire class, shared among all instances.
   * It is defined using the `static` keyword and is often used for constants or shared data among objects.

Example:

```java

Public class MathConstants {

Public static final double PI = 3.14159265359; // class variable (constant)

}

```

Here’s a program demonstrating the use of static and non-static blocks in Java:

```java

Public class StaticNonStaticBlocksExample {

Static {

System.out.println(“This is a static block.”);

}

{

System.out.println(“This is a non-static block.”);

}

Public StaticNonStaticBlocksExample() {

System.out.println(“This is the constructor.”);

}

Public static void main(String[] args) {

System.out.println(“Inside the main method.”);

StaticNonStaticBlocksExample obj1 = new StaticNonStaticBlocksExample();

StaticNonStaticBlocksExample obj2 = new StaticNonStaticBlocksExample();

}

}

```

In this program, we have a class with a static block (executed when the class is loaded), a non-static (instance) block (executed when an instance is created), and a constructor. When you run the `main` method, you’ll see the output showing the order in which these blocks and the constructor are executed.

Q9.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.

SOL:Command line arguments in Java are values or parameters that you can pass to a Java program when you run it from the command line. They allow you to customize the behavior of your program without modifying the source code. Command line arguments are stored in the `args` parameter of the `main` method as an array of strings.

Here’s a Java program that uses a wrapper class to print the sum of positive integers passed as command line arguments:

```java

Public class CommandLineArgumentsExample {

Public static void main(String[] args) {

Int sum = 0;

For (String arg : args) {

Try {

Int number = Integer.parseInt(arg);

If (number > 0) {

Sum += number;

}

} catch (NumberFormatException e) {

// Ignore non-integer arguments

}

}

System.out.println(“Sum of positive integers in command line arguments: “ + sum);

}

}

```

In this program, we iterate through the `args` array, parse each element as an integer using `Integer.parseInt()`, and check if the number is positive. If it’s a positive integer, we add it to the `sum`. Finally, we print the sum of positive integers entered as command line arguments.

You can run this program from the command line and provide positive integer values as arguments, and it will calculate and display their sum. For example, if you run the program as follows:

```

Java CommandLineArgumentsExample 5 10 15 -2 20

```

The program will calculate the sum of positive integers (5, 10, 15, and 20) and display the result, which is 50 in this case.

Q10.What is the use of char data type in java? Write a program to override any of object class methods.

SOL:The `char` data type in Java is used to represent a single 16-bit Unicode character. It’s typically used to store and manipulate individual characters, such as letters, digits, or symbols. `char` is particularly useful for tasks like processing text and working with characters in strings.

Here’s a simple example program that demonstrates the use of the `char` data type to print a character:

```java

Public class CharExample {

Public static void main(String[] args) {

Char myChar = ‘A’;

System.out.println(“The character is: “ + myChar);

}

}

```

Now, regarding your question about overriding a method from the `Object` class: All Java classes implicitly inherit from the `Object` class. You can override any of the methods from the `Object` class in your custom classes. Here’s an example where we override the `toString()` method:

```java

Class CustomObject {

Private int value;

Public CustomObject(int value) {

This.value = value;

}

@Override

Public String toString() {

Return “CustomObject with value: “ + value;

}

}

Public class ObjectMethodOverrideExample {

Public static void main(String[] args) {

CustomObject obj = new CustomObject(42);

System.out.println(obj.toString()); // Calls the overridden toString() method

}

}

```

In this example, the `CustomObject` class overrides the `toString()` method from the `Object` class to provide a custom string representation. When we call `obj.toString()`, it calls the overridden `toString()` method in the `CustomObject` class, and the custom string representation is printed.