ASSIGNMENT

Subject: Object-Oriented Programming (JAVA)

Class: MSc (Prev) Computer Science

Roll No: 40

Student Name: UMAIR AHMED

★ THEORY PART

Q.1: Why is Java called a portable and robust programming language?

Java is called a portable and robust programming language because of the following reasons:

Portability: Java is compiled into bytecode, which is a platform-independent format. This means that Java code can be run on any platform that has a Java Virtual Machine (JVM). The JVM is responsible for translating the bytecode into machine code that can be executed by the underlying hardware.

Robustness: Java is a robust programming language because it has a number of features that help to prevent errors and make it easier to recover from errors. These features include:

Strong memory management: Java uses a garbage collector to automatically manage memory allocation and deallocation. This frees developers from having to worry about memory leaks and other memory-related errors.

Type checking: Java is a statically typed language, which means that the types of all variables and expressions must be known at compile time. This helps to prevent many types of errors, such as trying to assign a value of the wrong type to a variable.

Exception handling: Java provides a mechanism for handling exceptions, which are unexpected errors that occur during program execution. By handling exceptions, Java programs can recover from errors and continue running.

Q.2: What are the differences between JDK, JRE, and JVM?

JDK, JRE, and JVM are all important components of the Java platform, but they have different roles.

JDK: stands for Java Development Kit. It is a set of tools and libraries that are used to develop Java applications. The JDK includes a compiler, a debugger, a documentation generator, and other tools.

JRE: stands for Java Runtime Environment. It is a set of libraries and other files that are needed to run Java applications. The JRE includes the JVM, as well as other libraries that provide common functionality, such as input and output, networking, and graphics.

JVM: stands for Java Virtual Machine. It is an abstract machine that executes Java bytecode. The JVM is responsible for loading Java classes, verifying bytecode, and executing bytecode instructions.

The main difference between the JDK and the JRE is that the JDK includes developer tools, while the JRE does not. If you are only interested in running Java applications, you only need to install the JRE. However, if you are developing Java applications, you will need to install the JDK.

Q.3: What is the latest version of JDK, and which version is considered the long-term version?

The latest version of JDK is 21. It was released in September 2023.

The current long-term support (LTS) version of Java is JDK 21. LTS releases are supported with security and bug fixes for at least six years. The previous LTS version was JDK 17, which was released in September 2021.

Q.4: When Java program source code is compiled, what code does it generate?

When Java program source code is compiled, it generates Java bytecode. Bytecode is a platform-independent intermediate representation of Java code that can be executed by any Java Virtual Machine (JVM).

Bytecode is a low-level language that is more efficient to execute than Java source code. It is also more compact, which makes it easier to download and store.

The Java compiler converts Java source code into bytecode by performing the following steps:

1. Lexical analysis: The compiler breaks the source code into tokens, which are the basic building blocks of the language.

2. Syntactic analysis: The compiler parses the tokens into an abstract syntax tree (AST), which is a representation of the program's structure.

3. Semantic analysis: The compiler checks the AST for errors and verifies that the program is semantically correct.

4. Code generation: The compiler generates bytecode from the AST.

Q.5: What are the differences between the String, StringBuilder, and StringBuffer classes?

The String, StringBuilder, and StringBuffer classes are all used to represent sequences of characters in Java. However, there are some important differences between them.

String: The String class is immutable, which means that once a String object is created, its value cannot be changed. Any attempt to change the value of a String object will result in the creation of a new String object.

String objects are also thread-safe, which means that they can be safely accessed and modified by multiple threads at the same time.

StringBuilder: The StringBuilder class is mutable, which means that the value of a StringBuilder object can be changed. StringBuilder objects are not thread-safe, which means that they should not be accessed and modified by multiple threads at the same time.

StringBuilder objects are typically more efficient than String objects for string manipulation operations, such as concatenation and insertion.

Q.6: What is the difference between the next() and nextLine() methods of the String class?

The next() and nextLine() methods of the String class are both used to extract tokens from a string. However, there is a key difference between the two methods:

next() returns the next token in the string, up to the next whitespace character.

nextLine() returns the next line in the string, up to the next newline character.

Q.7: Whatisthe signature/profile of a method? What are the compulsory keywords/components in the main method's signature?

The signature of a method is a unique identifier that consists of the method name, the return type, and the parameter types. The signature of a method is important because it allows the compiler to determine which method to call when a method call is made.

The signature of a method is written as follows:

return_type method_name(parameter_type_list)

For example, the following method has the signature `int add(int a, int b)`:

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

The main method is a special method that is the entry point for all Java programs. The main method must have the following signature:

public static void main(String[] args)

The `public` keyword means that the method can be accessed from anywhere in the program. The `static` keyword means that the method does not need to be associated with an object in order to be called. The `void` keyword means that the method does not return a value. And the `String[] args` parameter means that the method accepts an array of String objects as arguments.

Here is an example of a main method:

```
public static void main(String[] args) {
  System.out.println("Hello, world!");
}
```

This main method simply prints the message "Hello, world!" to the console.

Q.8: Discuss the basic structure of a Java program.

A Java program is made up of the following parts:

Package declaration: The package declaration specifies the package that the program belongs to. Packages are used to organize Java classes and interfaces.

Import statements: Import statements are used to import classes and interfaces from other packages.

Class definitions: A class definition defines a new class or interface. Classes are used to create blueprints for objects, and interfaces are used to define contracts between classes.

Method definitions: A method definition defines a new method. Methods are used to encapsulate code and make it reusable.

Main method: The main method is the entry point for all Java programs. It is the method that is executed when the program is run.

Q.9: What isthe difference between a compiler and an interpreter, and which one doesJava use?

A compiler and an interpreter are both programs that translate source code into a form that can be executed by a computer. However, they do this in different ways.

A compiler translates the entire source code program into machine code, which is the language that the computer's processor understands. This machine code is then saved as an executable file, which can be run on any computer that has the appropriate operating system.

An interpreter, on the other hand, translates the source code program line by line and executes it immediately. This means that an interpreter does not need to generate any machine code.

Q.10: What are control flow structures? List and discuss them.

Control flow structures are programming constructs that allow you to control the flow of execution of your program. They allow you to make decisions, iterate over loops, and jump to different parts of your program.

There are three main types of control flow structures:

Selection statements: Selection statements allow you to make decisions. They allow you to choose between two or more blocks of code to execute, depending on a condition. The most common selection statements are the `if` statement, the `else if` statement, and the `switch` statement.

Here is an example of an `if` statement:

```
if (age >= 18) {
  System.out.println("You are an adult.");
} else {
  System.out.println("You are a minor.");
}
```

This `if` statement checks the `age` variable to see if it is greater than or equal to 18. If it is, then the `System.out.println()` statement in the `if` block is executed. Otherwise, the `System.out.println()` statement in the `else` block is executed.

Iteration statements: Iteration statements allow you to iterate over loops. They allow you to execute a block of code repeatedly until a condition is met. The most common iteration statements are the `for` loop, the `while` loop, and the `do-while` loop.

Here is an example of a `for` loop:

```
for (int i = 0; i < 10; i++) {
  System.out.println(i);
}
```

This `for` loop initializes the variable `i` to 0. Then, it checks to see if `i` is less than 10. If it is, then the `System.out.println()` statement in the loop body is executed, and the value of `i` is incremented by 1. The loop continues to iterate until `i` is no longer less than 10.

Jump statements: Jump statements allow you to jump to different parts of your program. They allow you to interrupt the normal flow of execution and start executing code at a different location. The most common jump statements are the `break` statement and the `continue` statement.

Here is an example of a `break` statement:

```
for (int i = 0; i < 10; i++) {
  if (i == 5) {
    break;
  }
  System.out.println(i);
}
```

This `for` loop initializes the variable `i` to 0. Then, it checks to see if `i` is equal to 5. If it is, then the `break` statement is executed, and the loop terminates. Otherwise, the `System.out.println()` statement in the loop body is executed, and the value of `i` is incremented by 1.

Control flow structures are an essential part of any programming language. They allow you to write programs that are more complex and efficient.

Q.11: What istype casting? Differentiate between implicit and explicit type casting.

Type casting is the process of converting a value from one data type to another. It is a common operation in programming, and it is used in a variety of situations.

There are two types of type casting: implicit and explicit.

Implicit type casting: Implicit type casting is performed automatically by the compiler. It occurs when a value of one data type is assigned to a variable of another data type, and the two data types are compatible. For example, the following code will compile and run successfully:

int i = 10;

double d = i;

The compiler will automatically convert the value of `i` to a double before assigning it to the variable `d`.

Explicit type casting: Explicit type casting is performed manually by the programmer. It occurs when the programmer uses a cast operator to convert a value from one data type to another. For example, the following code will compile and run successfully:

int i = 10;

double d = (double) i;

The cast operator `(double)` tells the compiler to convert the value of `i` to a double before assigning it to the variable `d`.

Q.12: What is an array, and what does the length property of an array return?

An array is a data structure that stores a collection of elements. The elements of an array can be of any data type, such as integers, floating-point numbers, strings, or objects.

Arrays are indexed, which means that each element in an array has a unique index. The index of the first element in an array is 0, and the index of the last element in an array is the size of the array minus 1.

The length property of an array returns the number of elements in the array. The length property is a read-only property, which means that it cannot be changed.

Here is an example of an array in Java:

int[] numbers = new int[10];

This code declares an array of integers named `numbers` with a size of 10. The size of an array is the number of elements that the array can store.

To access an element in an array, you use the array's index. For example, the following code accesses the first element in the `numbers` array:

int firstElement = numbers[0];

This code will assign the value of the first element in the `numbers` array to the variable `firstElement`.

You can also use the length property of an array to iterate over the elements of the array. For example, the following code prints all of the elements in the `numbers` array to the console:

```
for (int i = 0; i < numbers.length; i++) {
  System.out.println(numbers[i]);
}
```

This code will iterate over the elements of the `numbers` array, starting at index 0 and ending at index 9. For each element in the array, the code will print the element's value to the console.

Arrays are a powerful data structure that can be used to store and manipulate data in a variety of ways. They are an essential part of many programming languages, and they are used in a wide variety of applications.

Q.13: What are the different editions of Java? List and explain each one.

Java has three main editions, each of which is designed for a specific set of use cases.

Java Standard Edition (Java SE)

The Java Standard Edition (Java SE) is the most common edition of Java. It is the foundation for all other editions of Java, and it includes the core Java APIs that are used to develop

desktop and server applications.

Java SE includes APIs for common tasks such as:

* Input and output

* Networking

* Graphics

* Security

* Collections

* Concurrency

Java Enterprise Edition (Java EE)

The Java Enterprise Edition (Java EE) is designed for developing enterprise applications. It includes APIs for common enterprise tasks such as:

* Web development

* EJB (Enterprise JavaBeans)

* JPA (Java Persistence API)

* JMS (Java Message Service)

* JTA (Java Transaction API)

Java Micro Edition (Java ME)

The Java Micro Edition (Java ME) is designed for developing applications for mobile devices and embedded systems. It includes APIs for common tasks such as:

* Mobile device user interfaces

* Device management

* Networking

* Security

★ IMPLEMENTATION PART

1. Basic Calculator.

```
public class BasicCalculator {
public static void main(String[] args) {
        int firstNumber = 2;
        int secondNumber = 5;
        int result = firstNumber + secondNumber;
        System.out.println(firstNumber + " + " + secondNumber + " = " + result);
    }
}
```

2. String Manipulation, Arrays and Control Statements.

```
import java.util.Arrays;
public class StudentDataProcessor {
    public static void main(String[] args)
        String studentData = "John:85, Alice:92, Bob:78, Carol:95, David:88, Emma:79,
Frank:90";
        String[] studentRecords = studentData.split(",")
        String[] studentNames = new String[studentRecords.length];
        int[] studentScores = new int[studentRecords.length];
        for (int i = 0; i < studentRecords.length; i++) {
            String[] studentInfo = studentRecords[i].split(":");
```

```
    studentNames[i] = studentInfo[0];
    studentScores[i] = Integer.parseInt(studentInfo[1]);
}
int numberOfStudents = studentNames.length;
int totalScore = 0;
for (int i = 0; i < studentScores.length; i++) {
    totalScore += studentScores[i];
}
double averageScore = (double) totalScore / numberOfStudents;
int highestScore = studentScores[0];
String highestScoringStudentName = studentNames[0];
for (int i = 1; i < studentScores.length; i++) {
    if (studentScores[i] > highestScore) {
        highestScore = studentScores[i];
        highestScoringStudentName = studentNames[i];
    }
}
int lowestScore = studentScores[0];
String lowestScoringStudentName = studentNames[0];
for (int i = 1; i < studentScores.length; i++) {
    if (studentScores[i] < lowestScore) {
        lowestScore = studentScores[i];
        lowestScoringStudentName = studentNames[i];
    }
```

```
        }

        System.out.println("Total number of students: " + numberOfStudents);

        System.out.println("Average score of all students: " + averageScore);

        System.out.println("Highest score: " + highestScore + ", achieved by " +
highestScoringStudentName);

        System.out.println("Lowest score: " + lowestScore + ", achieved by " +
lowestScoringStudentName);

    }

}
```