ASSIGNMENT

Q1 ans-

Statically typed and dynamically typed are two different type systems used in programming languages to handle variable types during compile-time and run-time, respectively. Let's explore each type system:

- **Statically Typed Programming Language**:
- In a statically typed language, variable types are determined and checked at compile-time, which is the process of translating source code into machine code or an intermediate representation.
- Variable types need to be explicitly declared when defining variables, and the type of each variable is known at compile-time and cannot change during runtime.
- The compiler checks the correctness of types before the program is executed, which helps catch type-related errors early in the development process.
- Statically typed languages are often considered more robust and provide better performance because type information is known beforehand.
- Examples of statically typed languages include Java, C, C++, Swift, and Kotlin.
- **Dynamically Typed Programming Language**:
- In a dynamically typed language, variable types are determined and checked at runtime, which is the process of executing the program.
- Variables do not require explicit type declarations, and their types are determined based on the value assigned to them during runtime.
- The type of a variable can change during the program's execution. For example, a variable can hold an integer value at one point and a string value at another point in the program.
- Since type checking is done at runtime, dynamically typed languages may be more flexible and concise, but they can lead to runtime errors related to type mismatches.
- Examples of dynamically typed languages include Python, JavaScript, Ruby, PHP, and Perl.

Q2 ans-

In Java, a variable is a named storage location that holds a value of a specific data type. Variables are used to store and manipulate data during the execution of a Java program. Before you can use a variable, you must declare it, specifying its name and data type.

Java supports several primitive data types, such as `int`, `double`, `boolean`, `char`, etc., which represent simple values like integers, floating-point numbers, boolean values (true or false), and characters. Additionally, Java allows you to create and work with objects, which are instances of classes. Object references, such as `String`, `ArrayList`, `Scanner`, and custom-defined classes, are also stored in variables.

Here's the general syntax to declare a variable in Java:

```java
data_type variable_name;
```

For example, to declare an integer variable named `age`, you would write:

```java
int age;
```

To declare a floating-point variable named `temperature`, you would write:

```java
double temperature;
```

To declare a boolean variable named `isRaining`, you would write:

```java
boolean isRaining;
```

To declare a character variable named `grade`, you would write:

```java
char grade;
```

To declare a reference variable for a `String` object named `message`, you would write:

```java
String message;
```

After declaring a variable, you can assign a value to it using the assignment operator (`=`):

```java
age = 30;
temperature = 25.5;
isRaining = true;
grade = 'A';
message = "Hello, Java!";
```

Alternatively, you can combine variable declaration and assignment in a single line:

```java
```

```
int age = 30;

double temperature = 25.5;

boolean isRaining = true;

char grade = 'A';

String message = "Hello, Java!";
```

Remember that once you declare a variable, you can change its value later in the program if needed. Variables provide flexibility and enable you to perform calculations, store user inputs, and manipulate data during the execution of a Java program.

Q3 ans-

In Java, you can assign a value to a variable using the assignment operator (`=`). The assignment operator is used to store a value in the memory location associated with the variable. Here's the general syntax for assigning a value to a variable:

```java
data_type variable_name = value;
```

Here's a step-by-step guide on how to assign a value to a variable in Java:

1. **Declare the Variable**: First, you need to declare the variable with its data type. This tells Java what type of data the variable will hold. For example, to declare an integer variable named `age`, you would write:

```java
int age;
```

2. **Assign a Value**: After declaring the variable, you can assign a value to it using the assignment operator (`=`). For example, to set the value of `age` to 30, you would write:

```java
age = 30;
```

Alternatively, you can combine variable declaration and assignment in a single line:

```java
int age = 30;
```

3. **Use the Variable**: Once you have assigned a value to the variable, you can use it in your program as needed. For example, you can perform calculations, use the variable in conditional statements, or display its value using `System.out.println()`.

Here's a complete example demonstrating how to assign values to variables in Java:

```java
public class Example {
    public static void main(String[] args) {
        // Declare and assign values to variables
        int age = 30;
        double temperature = 25.5;
        boolean isRaining = true;
        char grade = 'A';
        String message = "Hello, Java!";

        // Use the variables in the program
```

```
        System.out.println("Age: " + age);
        System.out.println("Temperature: " + temperature);
        System.out.println("Is it raining? " + isRaining);
        System.out.println("Grade: " + grade);
        System.out.println("Message: " + message);
    }
}
```

When you run this program, it will display the values assigned to the variables:

```
Age: 30
Temperature: 25.5
Is it raining? true
Grade: A
Message: Hello, Java!
```

That's how you assign values to variables in Java, allowing you to store and manipulate data in your programs.

Q4 ans-

In Java, primitive data types are basic data types that represent simple values. They are predefined by the Java language and are not objects or instances of classes. There are eight primitive data types in Java, which can be categorized into four groups: integer types, floating-point types, character type, and boolean type. Here are the primitive data types in Java:

1. **Integer Types**:

- `byte`: 8-bit signed two's complement integer. Range: -128 to 127.
- `short`: 16-bit signed two's complement integer. Range: -32,768 to 32,767.
- `int`: 32-bit signed two's complement integer. Range: -2^31 to 2^31 - 1.
- `long`: 64-bit signed two's complement integer. Range: -2^63 to 2^63 - 1.

2. **Floating-Point Types**:
   - `float`: 32-bit IEEE 754 floating-point. Used for decimal numbers with single-precision. Range: Approximately ±3.4e38 with 7 significant digits.
   - `double`: 64-bit IEEE 754 floating-point. Used for decimal numbers with double-precision. Range: Approximately ±1.7e308 with 15 significant digits.

3. **Character Type**:
   - `char`: 16-bit Unicode character. Represents a single character or letter enclosed in single quotes, e.g., 'A', 'b', '1', '@', etc.

4. **Boolean Type**:
   - `boolean`: Represents a boolean value, which can be either `true` or `false`. Used for conditions and logical expressions.

For example, you can declare variables using these primitive data types as follows:

```java
byte myByte = 100;
short myShort = 10000;
int myInt = 100000;
long myLong = 1000000000L; // Note the 'L' suffix for long literals
float myFloat = 3.14f; // Note the 'f' suffix for float literals
double myDouble = 2.71828;
char myChar = 'A';
boolean myBoolean = true;
```

Java's primitive data types are efficient and require less memory compared to objects (reference types). However, they lack the ability to store additional methods and properties, which objects possess. When working with more complex data structures and functionalities, Java developers often use wrapper classes to convert primitive data types to objects (e.g., `Integer`, `Double`, `Character`, etc.) to take advantage of the object-oriented features of Java.

Q5 ans-

In Java, an identifier is a name given to a variable, method, class, package, or other programming elements. Identifiers are used to uniquely identify these elements within a Java program. They serve as human-readable names that allow developers to refer to specific entities in the code.

Rules for forming valid identifiers in Java:

- **Must Begin with a Letter or Underscore**: An identifier must start with a letter (a-z or A-Z) or an underscore (_). It cannot begin with a digit (0-9).
- **Can Include Letters, Digits, and Underscores**: After the first character, an identifier can consist of letters, digits, and underscores. It must not contain any other special characters, spaces, or punctuation marks.
- **Must Not Be a Reserved Word**: Identifiers cannot be Java reserved words or keywords. For example, you cannot use "public," "class," "int," etc., as identifiers.
- **Case Sensitivity**: Java is case-sensitive, meaning that uppercase and lowercase letters are considered different. So, "myVariable" and "myvariable" are two distinct identifiers.
- **No Length Limit**: Java imposes no strict limit on the length of an identifier, but it is recommended to keep identifiers meaningful and concise.

Q6 ans-

In Java, operators are symbols or keywords that perform specific operations on variables, literals, or expressions. Java supports a variety of operators, which can be categorized into different groups based on their functionality. Here is a list of operators in Java:

1. **Arithmetic Operators**:

   - `+` Addition

   - `-` Subtraction

   - `*` Multiplication

- `/` Division

- `%` Modulus (Remainder)

2. **Assignment Operators**:

- `=` Assigns a value to a variable

- `+=` Adds the right operand to the left operand and assigns the result to the left operand (e.g., `a += 5;` is equivalent to `a = a + 5;`)

- `-=` Subtracts the right operand from the left operand and assigns the result to the left operand (e.g., `a -= 3;` is equivalent to `a = a - 3;`)

- `*=` Multiplies the right operand with the left operand and assigns the result to the left operand (e.g., `a *= 2;` is equivalent to `a = a * 2;`)

- `/=` Divides the left operand by the right operand and assigns the result to the left operand (e.g., `a /= 4;` is equivalent to `a = a / 4;`)

- `%=` Calculates the modulus (remainder) of the division of the left operand by the right operand and assigns the result to the left operand (e.g., `a %= 2;` is equivalent to `a = a % 2;`)

3. **Comparison (Relational) Operators**:

- `==` Equal to

- `!=` Not equal to

- `>` Greater than

- `<` Less than

- `>=` Greater than or equal to

- `<=` Less than or equal to

4. **Logical Operators**:

- `&&` Logical AND

- `||` Logical OR

- `!` Logical NOT

5. **Increment and Decrement Operators**:

   - `++` Increment by 1

   - `--` Decrement by 1

6. **Conditional (Ternary) Operator**:

   - `condition ? expression1 : expression2` If the condition is true, evaluate `expression1`; otherwise, evaluate `expression2`.

7. **Bitwise Operators**:

   - `&` Bitwise AND

   - `|` Bitwise OR

   - `^` Bitwise XOR (Exclusive OR)

   - `~` Bitwise NOT (Complement)

   - `<<` Left shift

   - `>>` Right shift

   - `>>>` Unsigned right shift

These operators play a crucial role in performing various operations and calculations in Java programs. Understanding how to use operators is essential for writing efficient and functional Java code.

Q7 ans -

In Java, the increment and decrement operators (`++` and `--`) are used to increase or decrease the value of a numeric variable by one. These operators can be applied to integer types (byte, short, int, long) and floating-point types (float, double).

Increment Operator (`++`):

The increment operator `++` adds 1 to the current value of a variable. It can be used as a prefix or a postfix operator.

1. As a Prefix Operator:

```java
int x = 5;
++x;
System.out.println(x); // Output: 6
```

2. As a Postfix Operator:

```java
int y = 10;
int result = y++;
System.out.println(result); // Output: 10 (the value of y before increment)
System.out.println(y);      // Output: 11 (the value of y after increment)
```

Decrement Operator (`--`):

The decrement operator `--` subtracts 1 from the current value of a variable. Like the increment operator, it can be used as a prefix or a postfix operator.

1. As a Prefix Operator:

```java
int a = 8;
--a;
System.out.println(a); // Output: 7
```

2. As a Postfix Operator:

```java
int b = 15;
int result = b--;
System.out.println(result); // Output: 15 (the value of b before decrement)
System.out.println(b);      // Output: 14 (the value of b after decrement)
```

When used in expressions, the position of the increment or decrement operator matters. If the operator is used as a prefix (`++x` or `--x`), the variable is incremented or decremented before the expression is evaluated. If used as a postfix (`x++` or `x--`), the variable is incremented or decremented after the expression is evaluated.

Example with a loop:

```java
int count = 0;

// Postfix increment inside the loop condition
while (count < 5) {
   System.out.println(count++); // Output: 0, 1, 2, 3, 4
}

// Prefix decrement inside the loop condition
while (--count >= 0) {
   System.out.println(count); // Output: 4, 3, 2, 1, 0
}
```

Remember that the increment and decrement operators should be used with caution, especially in complex expressions or loops, to avoid unexpected behavior or confusion.