

## Task 1:

### 1. Describe the syntax and structure of a Java™ method and arrays.

```
<access_modifie/may or not> <return_type> <method_name> (<parameter_list>)  
{  
    // Method body  
    // Code that defines the behavior of the method  
    // May include variable declarations, control flow statements, etc.  
    // Optionally, a return statement to return a value  
}
```

- **<access\_modifier>:** It specifies the visibility of the method. It can be public, private, protected, or default (no explicit modifier).
- **<return\_type>:** It specifies the data type of the value returned by the method. Use void if the method doesn't return any value.
- **<method\_name>:** It is the name of the method. Choose a meaningful name that describes the purpose of the method.
- **<parameter\_list>:** It specifies the input parameters (if any) required by the method. Parameters are enclosed in parentheses and separated by commas. Each parameter consists of a data type followed by a parameter name.

```
<datatype>[] <array_name> = new <datatype>[<size>];
```

- **<datatype>:** It represents the data type of the elements stored in the array. It can be any valid data type in Java, including primitive types or reference types.
- **<array\_name>:** It is the name given to the array variable.
- **new <datatype>[<size>]:** It is used to create a new array object with the specified size.

## 2. Classify the types of methods.

### By Return Type:

- **Void methods:** These methods do not return any value and are typically used to perform actions or modify data. (e.g., a method to print a message)
- **Value-returning methods:** These methods return a value of a specific data type. (e.g., a method to calculate the area of a circle)

### By Access Modifier:

- **Public methods:** These methods are accessible from any code within the same project or package.
- **Private methods:** These methods are only accessible from within the class where they are defined.
- **Protected methods:** These methods are accessible from within the same class and subclasses.

### By Scope:

- **Static methods:** These methods belong to the class itself and can be called without creating an object of the class. (e.g., a mathematical utility method)
- **Instance methods:** These methods are specific to an object and are called on an instance of the class. (e.g., a method to get the name of a person object)

### Other Classifications:

- **Overloaded methods:** Methods with the same name but different parameter lists, allowing for functionality based on argument types.
- **Recursive methods:** Methods that call themselves within their definition.

## 3. Define the overloaded methods and the scope of variables.

### Overloaded methods

Overloaded methods are methods in Java (and many other programming languages) that share the same name but have different parameter lists. This allows for functionality specific to the types and number of arguments provided when calling the method.

One of overloading examples in Java is ***print*** function.

### Scope of Variables:

The scope of a variable defines where it can be accessed and modified within your program. Here's a breakdown of different scopes in Java:

- **Local Variables:** These variables are declared within a method and are only accessible within that method. Their lifetime exists only during the method's execution.
- **Instance Variables:** These variables are declared within a class but outside any method. They are specific to an object of the class and can be accessed by any method of that object.
- **Class Variables (Static Variables):** These variables are declared with the static keyword within a class. They belong to the class itself and are shared by all objects of the class. They are accessed using the class name, not the object reference.

## 4. Implement 5 built-in methods in Java.

### 1. Random Method

```
int randy = (int) (Math.random()*10);  
System.out.println(randy); // Random Number
```

### 2. Uppercase

```
String str = "vini vidi vici";  
str = str.toUpperCase();  
System.out.println(str); // VINI VIDI VICI
```

### 3. LowerCase

```
String str = "VINI VIDI VICI";  
str = str.toLowerCase();  
System.out.println(str); // vini vidi vici
```

### 4. Equals

```
String str_1 = "vini vidi vici";
```

```
String str_2 = "VINI VIDI VICI";
String str_3 = "vini vidi vici";
System.out.println(str_1.equals(str_2)); // false
System.out.println(str_1.equals(str_3)); // true
```

## 5. Trim

```
String str = " vini vidi vici ";
System.out.println(str.trim()); //vini vidi vici
```

## 6. Length

```
String str = "vini vidi vici";
System.out.println(str.length()); // 14
```

5. Apply the one-dimensional array to store 5 values from the user then sort these values and print it after sorting.

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int array[] = new int[5];
    for (int i = 0; i < array.length; i++) {
        int number = input.nextInt();
        array[i] = number;
    }
    Arrays.sort(array);
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
    // Input: 4 3 2 5 1
    // Output: 1 2 3 4 5
}
```

6. Apply the multidimensional array [5][5]. This program takes this value from the user then sorts this values and prints it after sorting as two dimensions.

```

1  import java.util.Arrays;
2  import java.util.Scanner;
3
4  public class qq {
5
6      public static void main(String[] args) {
7          Scanner input = new Scanner(System.in);
8          int array[][] = new int[5][5];
9          for (int i = 0; i < array.length; i++) {
10             for (int j = 0; j < array.length; j++) {
11                 // int number = input.nextInt();
12                 int number = (int)(Math.random()*10);
13                 array[i][j] = number;
14             }
15         }
16         System.out.println("Before Sorting");
17         for (int i = 0; i < array.length; i++) {
18             for (int j = 0; j < array.length; j++) {
19                 System.out.print(array[i][j] + " ");
20             }System.out.println();
21         }
22
23         int array_1d[] = new int[25];
24         int pos = 0;
25
26         // Convert 2d to 1d
27         for (int i = 0; i < array.length; i++) {
28             for (int j = 0; j < array.length; j++) {
29                 array_1d[pos] = array[i][j];
30                 pos++;
31             }
32         }System.out.println();
33
34         Arrays.sort(array_1d);
35
36         // Convert 1d to 2d
37         int row = 0;
38         for (int i = 0; i < array_1d.length; i++) {
39             array[row][i%5] = array_1d[i];
40             if(i%5==4) row++;
41         }
42
43         System.out.println("After Sorting");
44         for (int i = 0; i < array.length; i++) {
45             for (int j = 0; j < array.length; j++) {
46                 System.out.print(array[i][j] + " ");
47             }System.out.println();
48         }
49         // Input : Random
50         // Output : Random : X(i j) < X(i+1 j+1)
51     }
52 }

```

7. Implement a method to sort the array in 5 using this method.



```
1 import java.util.Scanner;
2
3 public class q7 {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         int array[] = new int[5];
7         System.out.println("Enter Array Elements");
8         for (int i = 0; i < array.length; i++) {
9             int number = input.nextInt();
10            array[i] = number;
11
12        }
13        bubbleSort(array);
14        for (int i = 0; i < array.length; i++) {
15            System.out.print(array[i] + " ");
16        }
17    }
18
19    static void bubbleSort(int arr[])
20    {
21        int i, j, temp;
22        boolean swapped;
23        for (i = 0; i < arr.length; i++) {
24            swapped = false;
25            for (j = 0; j < arr.length - i - 1; j++) {
26                if (arr[j] > arr[j + 1]) {
27
28                    // Swap arr[j] and arr[j+1]
29                    temp = arr[j];
30                    arr[j] = arr[j + 1];
31                    arr[j + 1] = temp;
32                    swapped = true;
33                }
34            }
35
36            if (swapped == false)
37                break;
38        }
39    }
40 }
```

## Task 2:

1. Describe the relationship between classes and objects.

### **Classes:**

- A class is a blueprint or template for creating objects.
- It defines the properties (attributes or fields) and behaviors (methods) that objects of the class will have.
- It serves as a user-defined data type, encapsulating data and methods that operate on that data.
- Classes encapsulate the common attributes and behaviors shared by multiple objects.

### **Objects:**

- An object is an instance of a class.
- It represents a specific entity or instance in the real world that possesses the characteristics defined by its class.
- Each object has its own set of attributes, which are defined by the class, and can perform actions or methods defined by the class.
- Objects are concrete instances of classes and exist at runtime.

### **Relationship between Classes and Objects:**

- **Instantiation:** Objects are created based on classes through a process called instantiation. When you instantiate a class, you create an object of that class in memory.
- **Multiple Objects:** Multiple objects can be created from the same class, each with its own distinct set of attributes and behaviors.
- **Inheritance:** Classes can inherit attributes and behaviors from other classes through inheritance. This allows for code reuse and facilitates the creation of hierarchies of related classes.
- **Polymorphism:** Objects of different classes can be treated interchangeably through polymorphism. This allows for flexibility in designing systems and enables more generic and reusable code.
- **Encapsulation:** Classes encapsulate data and methods into a single unit, providing a level of abstraction and hiding the internal details of implementation from the outside world. Objects interact with each other through well-defined interfaces provided by their classes.



## 2.Explain the interaction with objects through reference variables

### Reference Variables:

- In Java, objects are accessed through reference variables. These variables hold references to objects in memory rather than the objects themselves.
- When you create an object using the new keyword, memory is allocated for the object, and a reference variable is assigned to hold the memory address (reference) where the object resides.
- Reference variables are typed, meaning they are declared to refer to objects of a specific class or interface.

### Object Creation:

- To interact with an object, you first need to create an instance of its class using the new keyword followed by a constructor invocation.
- For example:

```
MyClass obj = new MyClass(); // Creating an object of MyClass
```

### Accessing Members:

- Once you have a reference variable pointing to an object, you can access the members (fields and methods) of the object using the dot (.) operator.
- For example:

```
obj.fieldName; // Accessing a field of the object  
obj.methodName(); // Invoking a method of the object
```

### Passing References:

- When you pass an object as an argument to a method or assign it to another reference variable, you are passing or assigning the reference, not the object itself.
- This means that changes made to the object through one reference variable are reflected when accessed through another reference variable pointing to the same object.

**Null References:**

- Reference variables can also hold a special value called null, which indicates that they are not currently referring to any object.
- Attempting to access members of an object through a null reference will result in a runtime error (NullPointerException).

**Dereferencing:**

- Dereferencing refers to the process of accessing the object itself through its reference variable.
- When you access members of an object through a reference variable, the Java runtime system automatically dereferences the reference to access the correct memory location where the object resides.

### 3. Write an example of the classes with instance variables and instance methods.

```
1
2 public class t2_q3 {
3     public static class Student {
4         // Instance variables
5         private String name;
6         private int age;
7         private String major;
8
9         // Constructor
10        public Student(String name, int age, String major) {
11            this.name = name;
12            this.age = age;
13            this.major = major;
14        }
15
16        // Instance method to display student information
17        public void displayInfo() {
18            System.out.println("Student Information:");
19            System.out.println("Name: " + name);
20            System.out.println("Age: " + age);
21            System.out.println("Major: " + major);
22        }
23
24        // Instance method to check if the student is eligible to vote
25        public boolean isEligibleToVote() {
26            return age >= 18;
27        }
28
29        // Instance method to update student's major
30        public void updateMajor(String newMajor) {
31            this.major = newMajor;
32            System.out.println("Major updated to: " + newMajor);
33        }
34    }
35
36    public static void main(String[] args) {
37        Student student1 = new Student("Alice", 20, "Computer Science");
38        student1.displayInfo();
39
40        System.out.println("Is " + student1.name + " eligible to vote? " + student1.isEligibleToVote());
41
42        student1.updateMajor("Software Engineering");
43        student1.displayInfo();
44    }
45 }
```

#### 4. Describe the effect of common Java™ access modifiers and explain encapsulation.

##### 1. **public:**

- Members with the public access modifier are accessible from any other class or package.
- They have the widest scope of accessibility and can be accessed from anywhere in the program where the class is visible.

##### 2. **protected:**

- Members with the protected access modifier are accessible within the same package and by subclasses (whether in the same package or different package).
- They are not accessible by unrelated classes outside the package, unless they subclass the class containing the protected member.

##### 3. **default (no modifier):**

- Members with no explicit access modifier (i.e., default access) are accessible only within the same package.
- They are not accessible outside the package, even if they are in subclasses.

##### 4. **private:**

- Members with the private access modifier are accessible only within the same class.
- They are not accessible from any other class, including subclasses.

#### 5. Distinguish between reference variables and primitive variables with an example.

##### **Primitive Variables:**

- Primitive variables store simple data types (integers, floating-point numbers, characters, booleans) directly in memory.
- They hold the actual value of the data.
- Primitive variables are declared using the primitive data types in Java.
- Examples of primitive data types include int, double, char, boolean, etc.

```
int x = 5; // Primitive variable declaration and initialization
double y = 3.14;
char ch = 'A';
boolean flag = true;
```

## Reference Variables:

- Reference variables store memory addresses (references) of objects rather than the actual objects themselves.
- They do not hold the object's data directly; instead, they point to the memory location where the object is stored.
- Reference variables are declared based on classes or interfaces.
- They are used to access and manipulate objects in Java.

```
String str = "Hello"; // Reference variable
                        declaration and initialization
Scanner scanner = new Scanner(System.in);
ArrayList<Integer> list = new ArrayList<>();
```

## 6. Distinguish between instance and static variables

### Instance Variables:

- Instance variables are associated with instances or objects of a class. Each object of the class has its own copy of instance variables.
- They are declared within a class but outside of any method, constructor, or block, and they are initialized when an object of the class is created.
- Instance variables are accessed using object references (this keyword) and have separate memory space for each object.
- They represent the state of individual objects and can have different values for different objects.

```
public class MyClass {
    int instanceVariable; // Instance variable
    public void setInstanceVariable(int value) {
        this.instanceVariable = value;
    }
}
```

## Static Variables:

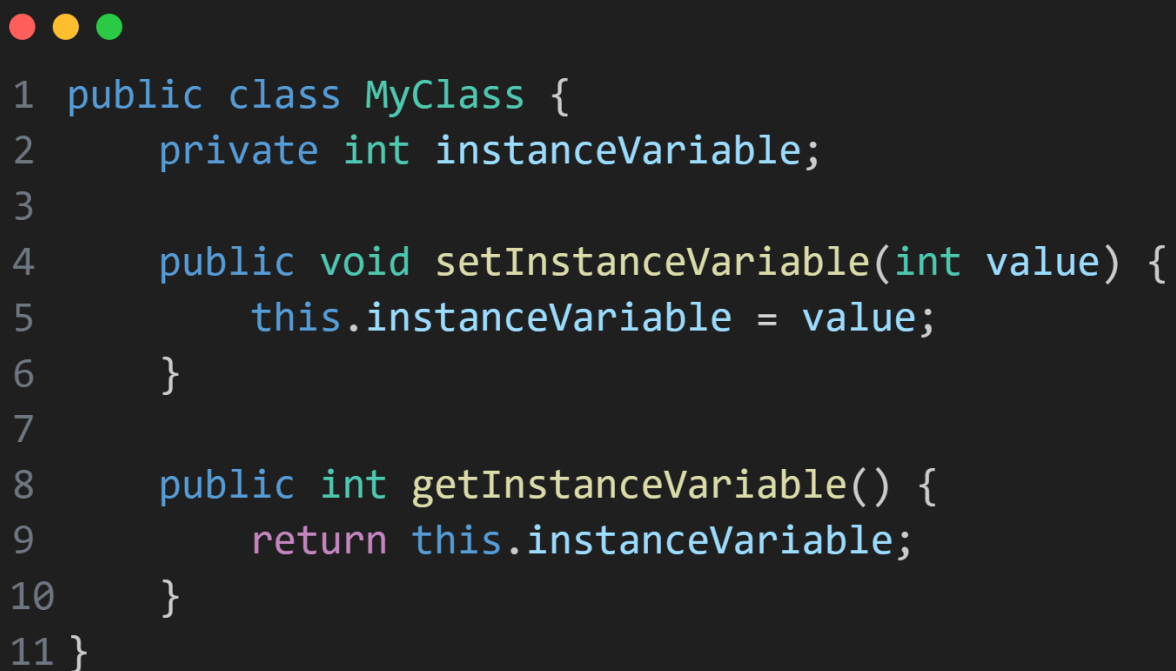
- Static variables, also known as class variables, are associated with the class rather than with any particular instance of the class.
- They are declared with the static keyword within a class, usually at the class level outside of any method, constructor, or block.
- Static variables are initialized only once when the class is loaded into memory, regardless of the number of objects created from the class.
- They are shared among all instances of the class and can be accessed directly using the class name without the need for an object reference.

```
public class MyClass {  
    static int staticVariable; // Static variable  
    public static void setStaticVariable(int value) {  
        staticVariable = value;  
    }  
}
```

## 7. Distinguish between instance and static methods.

### Instance Methods:

- Instance methods are associated with individual instances or objects of a class. They operate on the state of the object and can access instance variables and other instance methods.
- They are declared without the static keyword and are invoked on a specific object using a reference to that object.
- Instance methods can access and modify instance variables and can also call other instance methods within the same class.
- They are typically used to perform operations that are specific to individual objects and may vary in behavior depending on the state of the object.

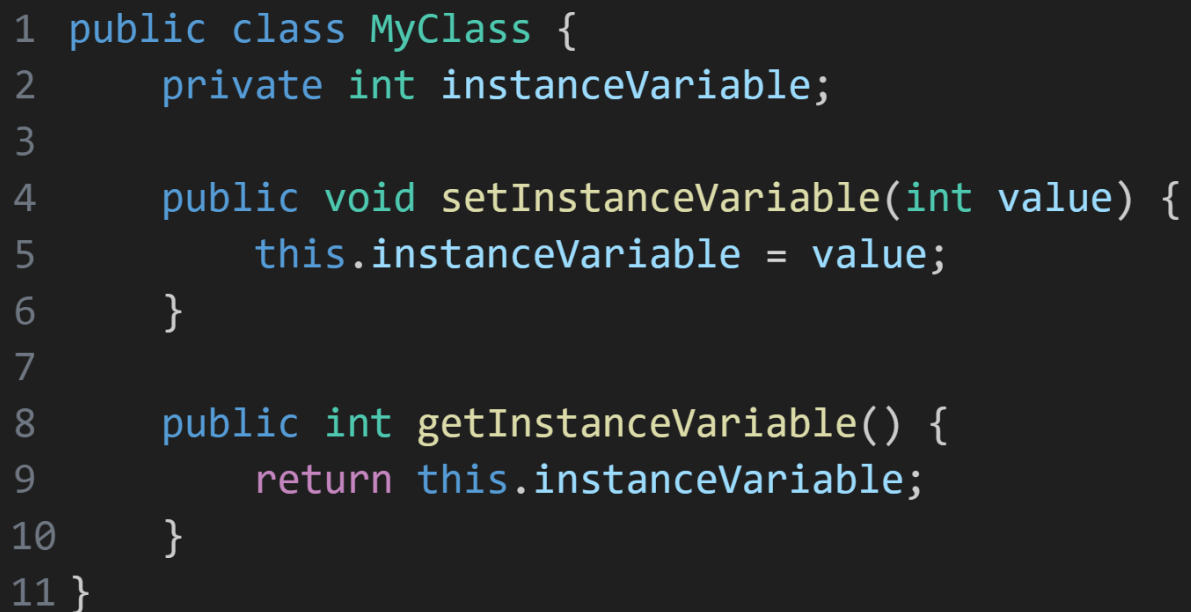


```
1 public class MyClass {  
2     private int instanceVariable;  
3  
4     public void setInstanceVariable(int value) {  
5         this.instanceVariable = value;  
6     }  
7  
8     public int getInstanceVariable() {  
9         return this.instanceVariable;  
10    }  
11 }
```

### Static Methods:

- Static methods, also known as class methods, are associated with the class rather than with any particular instance of the class. They do not operate on instance variables and cannot access this.
- They are declared with the static keyword and can be invoked directly on the class without the need for an object reference.

- Static methods can access only static variables and other static methods within the same class.
- They are commonly used for utility methods, where the behavior does not depend on the state of any particular object, or for operations that are related to the class itself rather than to instances of the class.



```
1 public class MyClass {  
2     private int instanceVariable;  
3  
4     public void setInstanceVariable(int value) {  
5         this.instanceVariable = value;  
6     }  
7  
8     public int getInstanceVariable() {  
9         return this.instanceVariable;  
10    }  
11 }
```

## 8. Compare String references and String content.

### **String References:**

- String references refer to memory addresses where strings are stored in memory.
- When you declare a string variable and assign it a value, you're creating a reference variable that points to the memory location where the string is stored.
- Multiple string variables can refer to the same string object in memory if they are assigned the same value.



- Comparing string references checks whether two string variables refer to the same memory address, not necessarily whether the content of the strings is the same.

```
1 String str1 = "Hello";  
2 String str2 = "Hello";  
3 boolean referenceEquals = (str1 == str2); // Compares references, returns true
```

## String Content:

- String content refers to the actual sequence of characters that make up the string.
- Strings in Java are immutable, meaning their content cannot be changed after they are created.
- Comparing string content checks whether the sequence of characters within the strings is the same, regardless of the memory addresses where the strings are stored.

```
1 String str1 = "Hello";  
2 String str2 = "Hello";  
3 boolean contentEquals = str1.equals(str2); // Compares content, returns true  
4
```

## 9. Use the 5 methods of String class to manipulate string in Java with examples

### 1. Uppercase

```
String str = "vini vidi vici";  
str = str.toUpperCase();  
System.out.println(str); // VINI VIDI VICI
```

### 2. LowerCase

```
String str = "VINI VIDI VICI";  
str = str.toLowerCase();  
System.out.println(str); // vini vidi vici
```

### 3. Equals

```
String str_1 = "vini vidi vici";  
String str_2 = "VINI VIDI VICI";  
String str_3 = "vini vidi vici";  
System.out.println(str_1.equals(str_2)); // false  
System.out.println(str_1.equals(str_3)); // true
```

### 4. Trim

```
String str = " vini vidi vici ";  
System.out.println(str.trim()); //vini vidi vici
```

### 5. Length

```
String str = "vini vidi vici";  
System.out.println(str.length()); // 14
```

## 10. Create objects using constructors and Create arrays of object references with an example

```
1 public class Task_2_q10 {
2     public static class Book {
3         private String title;
4         private String author;
5
6         // Constructor to initialize Book objects
7         public Book(String title, String author) {
8             this.title = title;
9             this.author = author;
10        }
11
12        // Getter for title
13        public String getTitle() {
14            return title;
15        }
16
17        // Setter for title
18        public void setTitle(String title) {
19            this.title = title;
20        }
21
22        // Getter for author
23        public String getAuthor() {
24            return author;
25        }
26
27        // Setter for author
28        public void setAuthor(String author) {
29            this.author = author;
30        }
31    }
32    public static void main(String[] args) {
33        Book book1 = new Book("Java Programming", "John Doe");
34        Book book2 = new Book("Python Programming", "Jane Smith");
35
36        Book[] booksArray = new Book[3];
37        booksArray[0] = new Book("Data Structures and Algorithms", "Alice Johnson");
38        booksArray[1] = new Book("Introduction to Machine Learning", "Bob Williams");
39        booksArray[2] = new Book("Web Development with JavaScript", "Emily Brown");
40
41        System.out.println(booksArray[0].getTitle()); // Output: Data Structures and Algorithms
42        System.out.println(booksArray[1].getAuthor()); // Output: Bob Williams
43    }
44 }
```