1Q.) Explain various control structures used in JAVA with suitable examples.

Java utilizes various control structures to manage the flow of program execution. These structures can be categorized into three main types: selection (conditional), iteration (looping), and jump (branching) statements.

1. Selection Statements (Conditional Statements)

These statements allow for the execution of different blocks of code based on specific conditions.

- **if statement**: Executes a block of code if a condition is true.

```
int score = 85;
if (score >= 60)

 {
    System.out.println("Passed the exam.");
 }
```

- **if-else statement**: Executes one block of code if a condition is true, and another if it's false.

```
int age = 17;
if (age >= 18)

{
  System.out.println("Eligible to vote.");
}

else

{
System.out.println("Not eligible to vote yet.");
}
```

- **if-else if-else ladder:** Allows for multiple conditions to be checked sequentially.

```
int grade = 75;

if (grade >= 90)
 {
   System.out.println("Grade A");
 }
else if (grade >= 80)

 {
System.out.println("Grade B");
 }

else
```

```
 {
System.out.println("Grade C or lower");
 }
```

- switch statement: Provides a way to execute different blocks of code based on the value of a variable or expression.

```
 String day = "Monday";
switch (day)

 {
   case "Monday":
    System.out.println("Start of the week.");
    break;
   case "Friday":
    System.out.println("End of the work week.");
    break;
   default:
    System.out.println("Mid-week day.");
 }
```

2. Iteration Statements (Looping Statements)

These statements allow for the repeated execution of a block of code.

- for loop: Executes a block of code a specific number of times.

```
for (int i = 0; i < 5; i++)

{
 System.out.println("Iteration: " + i);
}
```

- while loop: Executes a block of code repeatedly as long as a condition remains true.

```
 int count = 0;
while (count < 3)

{
 System.out.println("Count: " + count);
 count++;
}
```

- do-while loop: Executes a block of code at least once, and then repeatedly as long as a condition remains true.

```
 int num = 5;
do {
        System.out.println("Value: " + num);
       num--;
     } while (num > 0);
```

3. Jump Statements (Branching Statements)

These statements alter the normal flow of control within loops or methods.

- break statement: Terminates the execution of a loop or switch statement.

```
  for (int i = 0; i < 10; i++)

{
  if (i == 3)

  {
    break; // Exits the loop when i is 3
  }
 System.out.println(i);
}
```

- continue statement: Skips the current iteration of a loop and proceeds to the next iteration.

```
  for (int i = 0; i < 5; i++)

 {
    if (i == 2)

  {
   continue; // Skips printing when i is 2
 }
  System.out.println(i);
}
```

- return statement: Exits the current method and optionally returns a value.

```
  public int add(int a, int b)

{
     return a + b; // Returns the sum of a and b
}
```

2. Explain the different types of constructors with an example.

Constructors:

Constructors:

1. JAVA provides a special method called constructor which enables an object to initialize itself when it is created.
2. Constructor name and class name should be same.
3. Constructor is called automatically when the object is created.
4. Person p1=new Person() invokes the constructor Person() and Initializes the Person object p1.
5. Constructor does not return any return type (not even void).

There are two types of constructors.

1. Default constructor: A constructor which does not accept any parameters.

2. Parameterized constructor: A constructor that accepts arguments is called parameterized constructor.

Default constructor Example:

```
class Rectangle
{
    int length,bredth; // Declaration of variables
    Rectangle() // Default Constructor method
   {
      System.out.println("Constructing Rectangle..");
      length=10;
      bredth=20;
   }
   int rectArea()
  {
       return(length*bredth);
   }
}

class Rect_Defa
{
      public static void main(String args[])
     {
          Rectangle r1=new Rectangle();
          System.out.println("Area of rectangle="+r1.rectArea());
      }
}
```

Note: If the default constructor is not explicitly defined, then system default constructor automatically initializes all instance variables to zero.

Parameterized constructor:

```
class Rectangle
{
    int length,bredth; // Declaration of variables
    Rectangle(int x,int y) // Constructor method
    {
     length=x;
     bredth=y;
    }
    int rectArea()
   {
     return(length*bredth);
   }
}

class Rect_Para
{
       public static void main(String args[])
      {
           Rectangle r1=new Rectangle(5,10);
          System.out.println("Area of rectangle="+r1.rectArea());
       }
}
```

3.Write about class, object, methods, constructor and destructor.

1. **Class:**

•A class is the blueprint from which individual objects are created.

•This means the properties and actions of the objects are written in the class.

Class Declaration Syntax:

```
class ClassName
{
VariableDeclaration-1;
VariableDeclaration-2;
VariableDeclaration-3;
-----------------------
-----------------------
VariableDeclaration-n;
returnType methodname-1([parameters list])
{
body of the method…
}
returnType methodname-2([parameters list])
{
body of the method…
}
-----------------------
-----------------------
returnType methodname-3([parameters list])
{
body of the method…
}
}// end of class
```

2.Object:

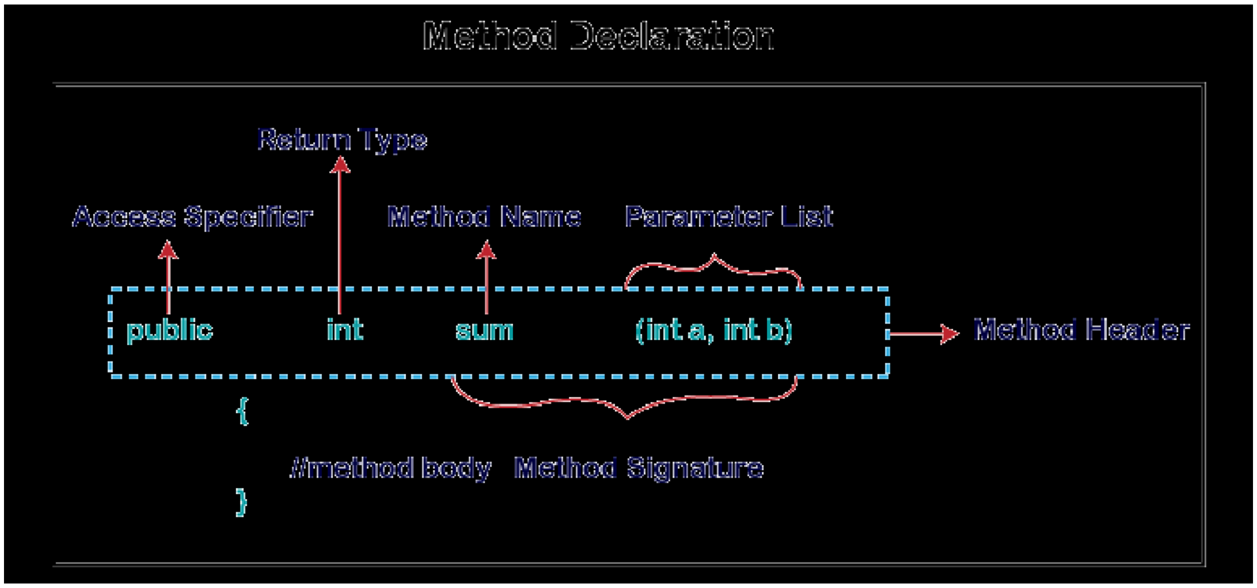
•Object is an entity of a class.

Object Creation Syntax:

```
ClassName objectName=new className ();
```

3.Methods in java:

• Method in Java is a collection of instructions that performs a specific task. It provides the reusability of code.



Types of Method

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined method:

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are length(), equals(), compareTo(), sqrt(), etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.
Each and every predefined method is defined inside a class. Such as print() method is defined in the java.io.PrintStream class.

Example:

```
public class Demo
{
public static void main(String[] args)
{
// using the max() method of Math class
System.out.print("The maximum number is: " + Math.max(9,7));
}
}
```

User-defined Method:

The method written by the user or programmer is known as a user-defined method. These methods are modified according to the requirement.

4. Constructors:

- JAVA provides a special method called constructor which enables an object to initialize itself when it is created.
- Constructor name and class name should be same.
- Constructor is called automatically when the object is created.
- Person p1=new Person() invokes the constructor Person() and Initializes the Person object p1.
- Constructor does not return any return type (not even void).

There are two types of constructors.
Default constructor: A constructor which does not accept any parameters.
Parameterized constructor: A constructor that accepts arguments is called parameterized constructor.

5.Destructor in Java

Java does **not** have explicit destructors like C++ because the garbage collector automatically manages Java’s memory. **Garbage collection** tracks objects in memory, and when they are no longer referenced, it automatically reclaims the memory.

5. Explain the various access specifier are used in java.

Access specifiers (Or) Access Control (Or) access Modifiers or Access Control for Class Members (Or) Accessing Private Members of Class:

An access specifier determines which feature of a class (class itself, data members, methods) may be used by another classes.
Java supports four access specifiers:

1. The public access specifier
2. The private access specifier
3. The protected access specifier
4. The Default access specifier

Public:

If the members of a class are declared as public then the members (variables/methods) are accessed by out side of the class.

Private:

If the members of a class are declared as private then only the methods of same class can access private members (variables/methods).

Protected:

Discussed later at the time of inheritance.

Default access:

If the access specifier is not specified, then the scope is friendly. A class, variable, or method that has friendly access is accessible to all the classes of a package (A package is collection of classes).

Example:

```
class Test
{
int a; //default access
public int b; // public access
private int c; // private access
//methods to access c
void setData(int i)
{
c=i;
}
int dispData()
{
return c;
}
}
class AccessTest
{
public static void main(String args[])
{
     Test ob=new Test();
    //a and b can be accessed directly
    ob.a=10;
    ob.b=20;
    // c can not be accessed directly because it is private
    //ob.c=100; // error
    // private data must be accessed with methods of the same class
    ob.setData(100);
    System.out.println(" value of a, b and c are:"+ob.a+" "+ob.b+" "+ob.dispData());
                                                                    ob.dispData();
}
}
```

6Q.Passing Arguments by Value and by Reference

• There is only call by value in java, not call by reference but we can pass non-primitive datatype to function to see the changes done by called function in caller function.

• If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example 1: Passing Primitive data type to function

```
 class Example
{
      int a=10;
      void change(int a) //called or callee function
      {
           a=a+100;
      }
}


    class CallByValue
    {
       public static void main(String args[]) //calling function
          {
             Example e=new Example();
             System.out.println("a value before calling change() :"+e.a); //10
             e.change(10); //call by value(passing primitive data type)
             System.out.println("a value after calling change() :"+e.a); //10
          }
    }
```

7Q)Constructor Methods overloading:

Writing more than one constructor with in a same class with different parameters is called constructor overloading.

Example:

```
class Addition
{
    int a,b;
    Addition()
  {
     a=10;
     b=20;
    }

   Addition(int a1,int b1)
  {
     a=a1;
     b=b1;
  }

  void add()
 {
   System.out.println("Addition of "+a+" and "+b+" is "+(a+b));
  }

}

class ConsoverLoading

{

   public static void main(String args[])

    {

       Addition obj=new Addition();

       obj.add(); //output:30

       Addition obj2=new Addition(2,3);

       obj2.add(); // output: 5

    }

}
```

8Q.Overloaded Methods or Method Overloading:

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

Example:

```
class Method
{
int add(int a,int b)
{
System.out.println("I am Integer method");
return a+b;
}
float add(float a,float b,float c)
{
System.out.println("I am float method");
return a+b+c;
}
int add(int a,int b,int c)
{
System.out.println("I am Integer method");
return a+b+c;
}
float add(float a,float b)
{
System.out.println("I am float method");
return a+b;
}
}
class MethodOverLoad
{
public static void main(String args[])
{
Method m=new Method();
System.out.println(m.add(10,20));
System.out.println(m.add(10.2f,20.4f,30.5f));
System.out.println(m.add(10,20,30));
System.out.println(m.add(10.2f,20.4f));
}
}
}
```

9Q. **Overriding Method:**

If subclass has the same method as declared in the parent class, it is known as method overriding.

Uses: runtime polymorphism

Rules:

• There must be a IS-A relationship(inheritance).

• The method must have same method signature as in the parent class.

Example:

```
class SuperClass
{
void calculate(double x)
{
System.out.println("Square value of X is: "+(x*x));
}
}
class SubClass extends SuperClass
{
void calculate(double x)
{
super();
System.out.println("Square root of X is: "+Math.sqrt(x));
}
}
class MethodOver
{
public static void main(String args[])
{
SubClass s=new SubClass();
s.calculate(2.5);
}
}
```