**Question Bank**

**Module 1**

**Object Oriented Programming using Java (21CIC34)**

**Semester:3rd  hansi**

1. Explain the three OOP principles.( CO1)

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

The Three OOP Principles

The three Object oriented programming (OOP) principles are 1)encapsulation

2) inheritance

3)polymorphism.

1) Encapsulation:

\*general meaning of encapsulation is the action of enclosing something in or enclosing some thing in a capsule.

\* Encapsulation is a mechanism that binds code and the data it manipulates together, and keeps both of them safe from misuse. and outside interference

Or

**\*Encapsulation in Java** is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

\* The whole idea behind encapsulation is to hide the implementation details from users

\* .In encapsulation, the data in one class is hidden from other classes so data in one class cannot be accessed by other classes, so it is also known as data-hiding

\* Both Abstraction & Encapsulation works hand in hand because Abstraction says what details to be made visible & Encapsulation provides the level of access to that visible details.

To hide data from users , you must:

* declare class variables/attributes/methods as private
* provide public **get** and **set** methods to access and update the value of a private variable

**Example** :

Power steering of a car is a complex system, which internally have lots of components tightly coupled together, they work synchronously to turn the car in the desired direction. **But to the external world there is only one interface i.e., the steering wheel is available and rest of the complexity is hidden.**

Similarly in java if want to protect the data u need declare variables/attributes/methods as private if u do like that then any another class or any other code which is not a member of same class cannot access the variable/attribute/method

Advantages of Encapsulation:

• **Data Hiding:** The user will have no idea about the inner implementation of the class

• **Increased Flexibility:** We can make the variables of the class as read-only or write-only based on our requirement.

• **Reusability**: Encapsulation also improves the re-usability

• **Testing code is easy**: Encapsulated code is easy to test for unit testing.

Inheritance:

It is **the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class**.

Important terminology:

• **Super Class or a base class or a parent class**:

The class whose features are inherited is known as super class

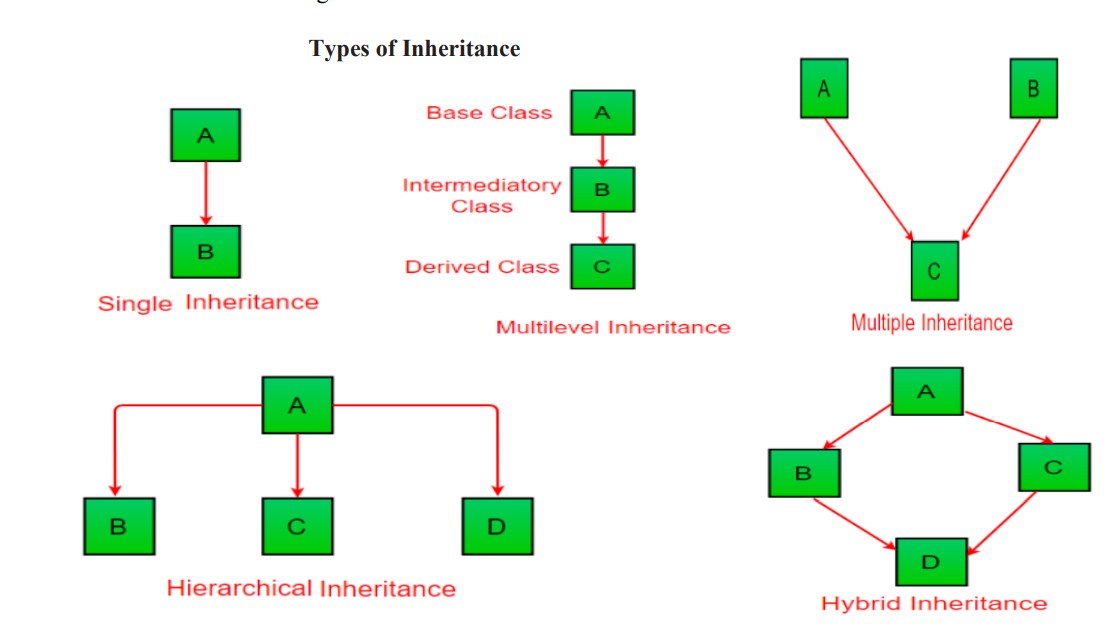
• **Sub Class or a derived class, extended class, or child class**:

\*The class that acquires or inherits the other class features is known as sub class().

\*The subclass can add its own fields and methods in addition to the superclass fields and methods.

• Reusability:

Inheritance supports the concept of ―reusability‖, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class.



In the example below, the Car class (subclass) takes or inherits the attributes and methods from the Vehicle class (superclass)

|  |
| --- |
| class Vehicle //superclass  {  public String company = "Ford"; // Vehicle attribute  public void method1() //note no static keyword // Vehicle method  {  System.out.println("Tuut, tuut!");  }  }  //subclass  class Car extends Vehicle  {  private String modelName = "Mustang"; // Car attribute  public static void main(String[] args)  {  // Creating a myCar object  Car myCar = new Car();  // Calling the method1() method (from the Vehicle class) on the myCar object  myCar.method1();  // Displaying the value of the company attribute (from the Vehicle class) and the value of the attribute modelName from the Car class  System.out.println(myCar.company + " " + myCar.modelName);  }  } |

Output:

|  |
| --- |
| Tuut, tuut! Ford Mustang |

Polymorphism

Refer the link

https://www.programiz.com/java-programming/polymorphism

Polymorphism is an important concept of object-oriented programming. It simply means more than one form.

That is, the same entity (method or operator or object) can perform different operations in different scenarios.

## Example: Java Polymorphism

class Polygon {

// method to render a shape

public void render() {

System.out.println("Rendering Polygon...");

}

}

class Square extends Polygon {

// renders Square

public void render() {

System.out.println("Rendering Square...");

}

}

class Circle extends Polygon {

// renders circle

public void render() {

System.out.println("Rendering Circle...");

}

}

class Main {

public static void main(String[] args) {

// create an object of Square

Square s1 = new Square();

s1.render();

// create an object of Circle

Circle c1 = new Circle();

c1.render();

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Rendering Square...

Rendering Circle...

In the above example, we have created a superclass: Polygon and two subclasses: Square and Circle. Notice the use of the render() method.

The main purpose of the render() method is to render the shape. However, the process of rendering a square is different than the process of rendering a circle.

Hence, the render() method behaves differently in different classes. Or, we can say render() is polymorphic.

### Why Polymorphism?

Polymorphism allows us to create consistent code. In the previous example, we can also create different methods: renderSquare() and renderCircle() to render Square and Circle, respectively.

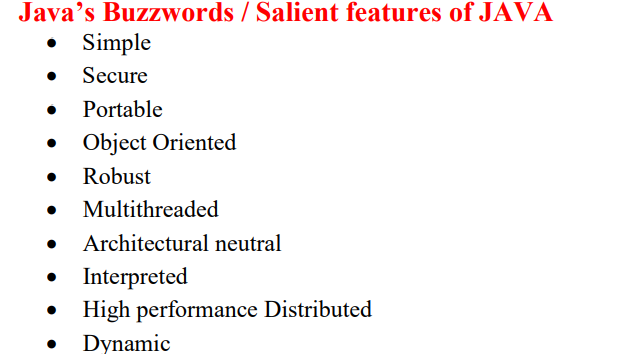
This will work perfectly. However, for every shape, we need to create different methods. It will make our code inconsistent.

To solve this, polymorphism in Java allows us to create a single method render() that will behave differently for different shapes.

**Note**: The print() method is also an example of polymorphism. It is used to print values of different types like char, int, string, etc.

We can achieve polymorphism in Java using the following ways:

1. [Method Overriding](https://www.programiz.com/java-programming/method-overriding)
2. [Method Overloading](https://www.programiz.com/java-programming/method-overloading)
3. Operator Overloading
4. Discuss Java’s buzzwords or salient features of Java.( CO1)



**Simple:**

Java was designed to be easy for the professional programmer to learn and use effectively. If one has some programming experience, he will not find Java hard to master.If you already understand the basic concepts of object-oriented programming, learning Java will be even easier.

Java is considered as one of simple language features like because it doesn’t have complex features like operator overloading, pointers, Multiple inheritance.

**Secure :**

Java is secure bcz if u are using a webbrowser if u download something from the internet there are chances of viral infection which may steal the personal information like bank account numbers ,credit card details etc which are on our local system.so java supportable browser will protects us by downloading applets with out fear of viral infection bcz java achieved this protection by allowing an applet to the Java execution environment and not allowing it access to other parts of the computer

**Portable :**

➢When we compile the java program, the compiler of java generates .class file that contains the Bytecodes of your java program.

➢The generated Bytecodes are secure and can run on any machine (portable) which has JVM.

**Object Oriented**

Basic concepts of Object Oriented programming are

1. Object

2. Class

3. Inheritance

4. Polymorphism

5. Abstraction

6. Encapsulation

Java is purely object oriented programming language because without class and object it is impossible to write any Java program. Java is not pure object oriented programming language. because java supports primitive datatypes like int ,float ,boolean, double, long etc.

**Robust :**

Robust: Two main problems that cause program failures are memory management mistakes and mishandled runtime errors.

Java handles both of them efficiently.

➢ Memory management mistakes can be overcome by garbage collection. Garbage collection automatically deallocates objects which are no longer needed.

Mishandled runtime errors are resolved by Exception Handling procedures.

**Multithreaded :**

:Java supports multithreading. It enables a program to perform several tasks simultaneously. **Distributed** :

➢RMI (Remote Method Invocation) and EJB (Enterprise Javabeans) are used for creating distributed applications in java.

➢ In simple words: The java programs can be distributed on more than one systems that are connected to each other using internet connection. Objects on one JVM (java internet virtual machine) can execute procedures on a remote JVM.

**Architectural in Neutral:**

In java there is no implementation dependent feature e.g. size of primitive types is fixed.

In C programming, float data type occupies 4 bytes of memory for 32-bit architecture and 8 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

**Interpreted and High performance** :

Java interpreter converts bytecode into processor readable binary code .

For very high performance it uses a Just-in-time compiler.

**Dynamic** :

Java is considered to be more dynamic than C or C++ since **it is designed to adapt to an evolving environment**. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.

1. Write a note on Java Development Kit (JDK).( CO1)

Refer [ppt](file:///C:\Users\prave\OneDrive\Desktop\git%20java\wrwe\java\sem%203\mod%201\Java_Module_1.pdf) pgno 39

1. Explain how “compile once and run anywhere” is implemented in java. ( CO1)

Write answer for portable

1. What is abstraction? Explain with an example.( CO1)

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user..  
Abstraction can be achieved with either **abstract classes** or [**interfaces**](https://www.w3schools.com/java/java_interface.asp) (which you will learn more about in the next chapter).

Abstraction lets you focus on what the [object](https://www.javatpoint.com/object-and-class-in-java) does instead of how it does

The abstract keyword is a non-access modifier, used for classes and methods:

* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

abstract class Animal {

public abstract void animalSound();

public void sleep() {

System.out.println("Zzz");

}

}

From the example above, it is not possible to create an object of the Animal class:

Animal myObj = new Animal(); // will generate an error

To access the abstract class, it must be inherited from another class. Let's convert the Animal class we used in the [Polymorphism](https://www.w3schools.com/java/java_polymorphism.asp) chapter to an abstract class:

1. What is polymorphism? Explain the two different types of it.( CO1)

Polymorphism is an important concept of object-oriented programming. It simply means more than one form.

That is, the same entity (method or operator or object) can perform different operations in different scenarios.

## Example: Java Polymorphism

class Polygon {

// method to render a shape

public void render() {

System.out.println("Rendering Polygon...");

}

}

class Square extends Polygon {

// renders Square

public void render() {

System.out.println("Rendering Square...");

}

}

class Circle extends Polygon {

// renders circle

public void render() {

System.out.println("Rendering Circle...");

}

}

class Main {

public static void main(String[] args) {

// create an object of Square

Square s1 = new Square();

s1.render();

// create an object of Circle

Circle c1 = new Circle();

c1.render();

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Rendering Square...

Rendering Circle...

In the above example, we have created a superclass: Polygon and two subclasses: Square and Circle. Notice the use of the render() method.

The main purpose of the render() method is to render the shape. However, the process of rendering a square is different than the process of rendering a circle.

Hence, the render() method behaves differently in different classes. Or, we can say render() is polymorphic.

Why Polymorphism?

Polymorphism allows us to create consistent code. In the previous example, we can also create different methods: renderSquare() and renderCircle() to render Square and Circle, respectively.

This will work perfectly. However, for every shape, we need to create different methods. It will make our code inconsistent.

To solve this, polymorphism in Java allows us to create a single method render() that will behave differently for different shapes.

**Note**: The print() method is also an example of polymorphism. It is used to print values of different types like char, int, string, etc.

We can achieve polymorphism in Java using the following ways:

1. [Method Overriding](https://www.programiz.com/java-programming/method-overriding)(runtime polymorphism)
2. [Method Overloading](https://www.programiz.com/java-programming/method-overloading)(compile time polymorphism)
3. Operator Overloading
4. **What is a variable? How do you declare and use it, explain with an example. ( CO1)**

# **Java Variables**

A variable is a container which holds the value while the [Java program](https://www.javatpoint.com/simple-program-of-java) is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.



### Types of Variables

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial):

* local variable
* instance variable
* static variable



#### **1) Local Variable**

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

#### **2) Instance Variable**

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as [static](https://www.javatpoint.com/static-keyword-in-java).

#### **3) Static variable**

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

### Example to understand the types of variables in java

1. **public** **class** A
2. {
3. **static** **int** m=100;//static variable
4. **void** method()
5. {
6. **int** n=90;//local variable
7. }
8. **public** **static** **void** main(String args[])
9. {
10. **int** data=50;//instance variable
11. }
12. }//end of class

### Java Variable Example: Add Two Numbers

1. **public** **class** Simple{
2. **public** **static** **void** main(String[] args){
3. **int** a=10;
4. **int** b=10;
5. **int** c=a+b;
6. System.out.println(c);
7. }
8. }

**Output:**

20

1. Explain the different data types available in java. ( CO1)



|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size** | **Description** |
| byte | 1 byte | Stores whole numbers from -128 to 127 |
| short | 2 bytes | Stores whole numbers from -32,768 to 32,767 |
| int | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| boolean | 1 bit | Stores true or false values |
| char | 2 bytes | Stores a single character/letter or ASCII values |

1. Discuss the ternary (?) operator and short circuit logical Operators.( CO1)

Ternary operator is short hand notation of if else

short-hand [if else](https://www.w3schools.com/java/java_conditions.asp) is also known as the **ternary operator** because it consists of three operands.

It can be used to replace multiple lines of code with a single line, and is most often used to replace simple if else statements:

### Syntax

If variable is already declared(note variable need to be declared of type value u returns if expression is true or false)

variable *= (*condition*) ?* expressionifconsitionTrue *:*  expressionFif conditionfalse*;*

if variable is not declared yet:

datatype variable *= (*condition*) ?* expressionTrue *:*  expressionFalse*;*

here:

\*variable is assigned with expesiionTrue or expressionFalse based on the condition for ex:

|  |
| --- |
| public class Main {  public static void main(String[] args) {  int time = 20;  String result;  result = (time < 18) ? "Good day." : "Good evening.";  System.out.println(result);  }  } |

If we write this in if else format:

|  |
| --- |
| public class Main {  public static void main(String[] args) {  int time = 20;  if (time < 18) {  System.out.println("Good day.");  } else {  System.out.println("Good evening.");  }  }  } |

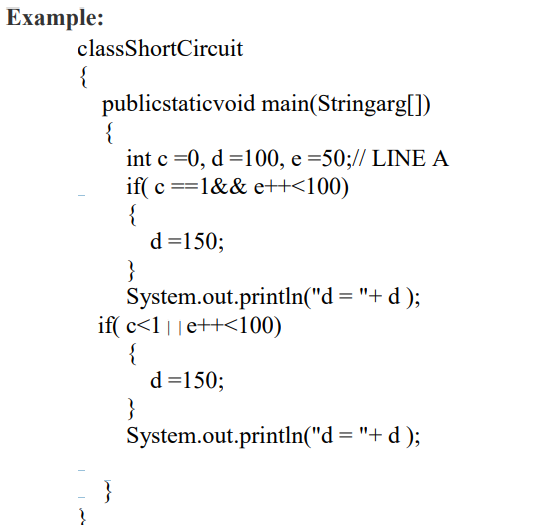
**Short Circuit Logical Operators**

\* Short circuit logical operators evaluate second expression only if this is needed.

\* When short-circuit AND (&&) is used, if the first value is false, second value is not evaluated as the result is false irrespective of the second value.

\* Similarly for short-circuit OR (||), if the first value is true, then second value is not evaluated as the result is true irrespective of the second value..

\* These short-circuit operators will be useful when we want to control the evaluation of right hand operand.



Or

1. Differentiate method overloading and overriding.( CO1)

|  |  |  |
| --- | --- | --- |
| **No.** | **Method Overloading** | **Method Overriding** |
| 1) | Method overloading is used *to increase the readability* of the program. | Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class. |
| 2) | Method overloading is performed *within class*. | Method overriding occurs *in two classes* the parent class and child class that have IS-A (inheritance) relationship. |
| 3) | In case of method overloading, *parameter must be different*. | In case of method overriding, *parameter must be same*. |
| 4) | Method overloading is the example of *compile time polymorphism*. | Method overriding is the example of *run time polymorphism*. |
| 5) | In java, method overloading can't be performed by changing return type of the method only. | *Return type must be same* in method overriding. |

1. Demonstrate the uses of break and continue statements with an example.( CO1)

The break and continue statements are the jump statements that are used to skip some statements inside the loop or terminate the loop immediately without checking the test expression. These statements can be used inside any[loops](https://www.geeksforgeeks.org/loops-in-java/) such as for, while, do-while loop.

[**Break**](https://www.geeksforgeeks.org/break-statement-in-java/)**:**The break statement in java is used to terminate from the loop immediately. When a break statement is encountered inside a loop, the loop iteration stops there, and control returns from the loop immediately to the first statement after the loop. Basically, break statements are used in situations when we are not sure about the actual number of iteration for the loop, or we want to terminate the loop based on some condition.

**Syntax :**

**break;**

B reak;

|  |
| --- |
| / Java program to demonstrate using  // break to exit a loop  **class** GFG {  **public** **static** **void** main(String[] args)      {          // Initially loop is set to run from 0-9  **for** (**int** i = 0; i < 10; i++) {              // Terminate the loop when i is 5  **if** (i == 5)  **break**;              System.out.println("i: " + i);          }          System.out.println("Out of Loop");      }  } |

**Output**

i: 0

i: 1

i: 2

i: 3

i: 4

Out of Loop

**Continue:**

The continue statement in Java is used to skip the current iteration of a loop. We can use continue statement inside any types of loops such as for, while, and do-while loop. Basically continue statements are used in the situations when we want to continue the loop but do not want the remaining statement after the continue statement.

**Syntax:**

continue;

**Using continue to continue a loop**

Using continue, we can skip the current iteration of a loop and jumps to the next iteration of the loop immediately.

**Example:**

Java

|  |
| --- |
| // Java program to demonstrates the continue  // statement to continue a loop  **class** GFG {  **public** **static** **void** main(String args[])      {  **for** (**int** i = 0; i < 10; i++) {              // If the number is 2              // skip and continue  **if** (i == 2)  **continue**;                System.out.print(i + " ");          }      }  } |

**Output**

0 1 3 4 5 6 7 8 9

1. Differentiate Implicit and explicit type casting with examples. ( CO1)

# **Type Casting in Java**

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer. In this section, we will discuss **type casting** and **its types** with proper examples.



## Type casting

Convert a value from one data type to another data type is known as **type casting**.

## Types of Type Casting

There are two types of type casting:

* Widening Type Casting
* Narrowing Type Casting

### Widening Type Casting

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when:

* Both data types must be compatible with each other.
* The target type must be larger than the source type.

1. **byte** -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**

For example, the conversion between numeric data type to char or Boolean is not done automatically. Also, the char and Boolean data types are not compatible with each other. Let's see an example.

**WideningTypeCastingExample.java**

1. **public** **class** WideningTypeCastingExample
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. **int** x = 7;
6. //automatically converts the integer type into long type
7. **long** y = x;
8. //automatically converts the long type into float type
9. **float** z = y;
10. System.out.println("Before conversion, int value "+x);
11. System.out.println("After conversion, long value "+y);
12. System.out.println("After conversion, float value "+z);
13. }
14. }

**Output**

Before conversion, the value is: 7

After conversion, the long value is: 7

After conversion, the float value is: 7.0

In the above example, we have taken a variable x and converted it into a long type. After that, the long type is converted into the float type.

### Narrowing Type Casting

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

1. **double** -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

Let's see an example of narrowing type casting.

In the following example, we have performed the narrowing type casting two times. First, we have converted the double type into long data type after that long data type is converted into int type.

**NarrowingTypeCastingExample.java**

1. **public** **class** NarrowingTypeCastingExample
2. {
3. **public** **static** **void** main(String args[])
4. {
5. **double** d = 166.66;
6. //converting double data type into long data type
7. **long** l = (**long**)d;
8. //converting long data type into int data type
9. **int** i = (**int**)l;
10. System.out.println("Before conversion: "+d);
11. //fractional part lost
12. System.out.println("After conversion into long type: "+l);
13. //fractional part lost
14. System.out.println("After conversion into int type: "+i);
15. }
16. }

**Output**

Before conversion: 166.66

After conversion into long type: 166

After conversion into int type: 166

1. Distinguish while and do-while statements with example.( CO1)

Loops:

\*Loops can execute a block of code as long as a specified condition is reached.

\*Loops are handy because they save time, reduce errors, and they make code more readable.

While loop:

\*The while loop loops through a block of code as long as a specified condition is true

Or

The while loop iterates through a block of code as long as a specified condition is true

### Syntax

while (condition) {

*// code block to be executed*

}

EX:

public class Main {

public static void main(String[] args) {

int i = 0;

while (i < 5)

{

System.out.println(i);

i++;

}

}

}Output:

0  
1  
2  
3  
4

**Note:** Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Do/While loop:

\*The do/while loop is a variant of the while loop.

\*This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

### Syntax

do {

*// code block to be executed*

}

while (condition);

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

### Example

|  |
| --- |
| public class Main {  public static void main(String[] args) {  int i = 0;  do {  System.out.println(i);  i++;  }  while (i < 5);  }  } |

Output:

0  
1  
2  
3  
4

1. Explain the switch statement with example. ( CO1)

The switch statement is Java‘s multiway branch statement

. It provides an easy way to skip some parts of code and execute some other parts of code based on the value of an expression.

It provides a better alternative to large series of if-else-if statements.

Here is the general form of a switch statement:

|  |
| --- |
| switch (expression)  {  case value1:  statement 1;  statement 2;  .  .  Statement n;  break;  case value2:  statement 1;  statement 2;  .  .  Statement n;  break;  .  .  .  .  case valueN:  statement 1;  statement 2;  .  .  Statement n;  break;  default:  statement 1;  statement 2;  .  .  Statement n;  break;  } |
|  |

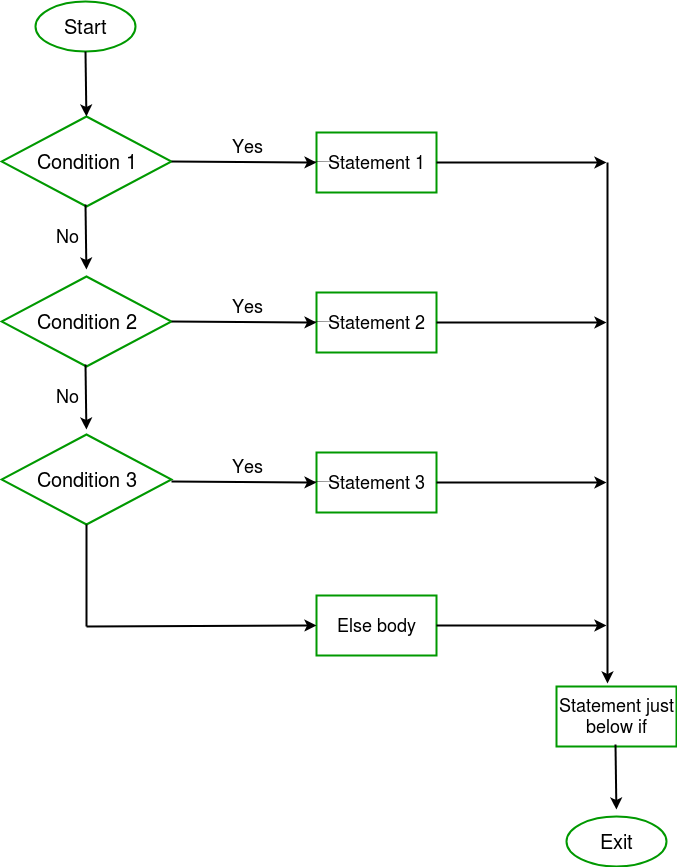
Ex: switch with expression values of integers

|  |
| --- |
| import java.util.Scanner;  public class switch1 {  public static void main(String args[]) {  System.out.println("1:hi 2:bye 3:tata");  Scanner sc = new Scanner(System.in);  int a = sc.nextInt();  switch (a) {  case 1:  System.out.println("hi");  break;  case 2:  System.out.println("bye");  break;  case 3:  System.out.println("tata");  break;  }  }  } |

switch with expression values as char

|  |
| --- |
| public class switchwithchar {  public static void main(String[] args) {  char num = 'a';  switch (num) {  case 'a':  System.out.println("number is 0");  break;  case 'b':  System.out.println("number is 1");  break;  default:  System.out.println(num);  }  }  } |

1. Discuss if-else-if ladder with an example.( CO1)



// Java program to illustrate if-else-if ladder

import java.io.\*;

class GFG {

public static void main(String[] args)

{

// initializing expression

int i = 20;

// condition 1

if (i == 10)

System.out.println("i is 10\n");

// condition 2

else if (i == 15)

System.out.println("i is 15\n");

// condition 3

else if (i == 20)

System.out.println("i is 20\n");

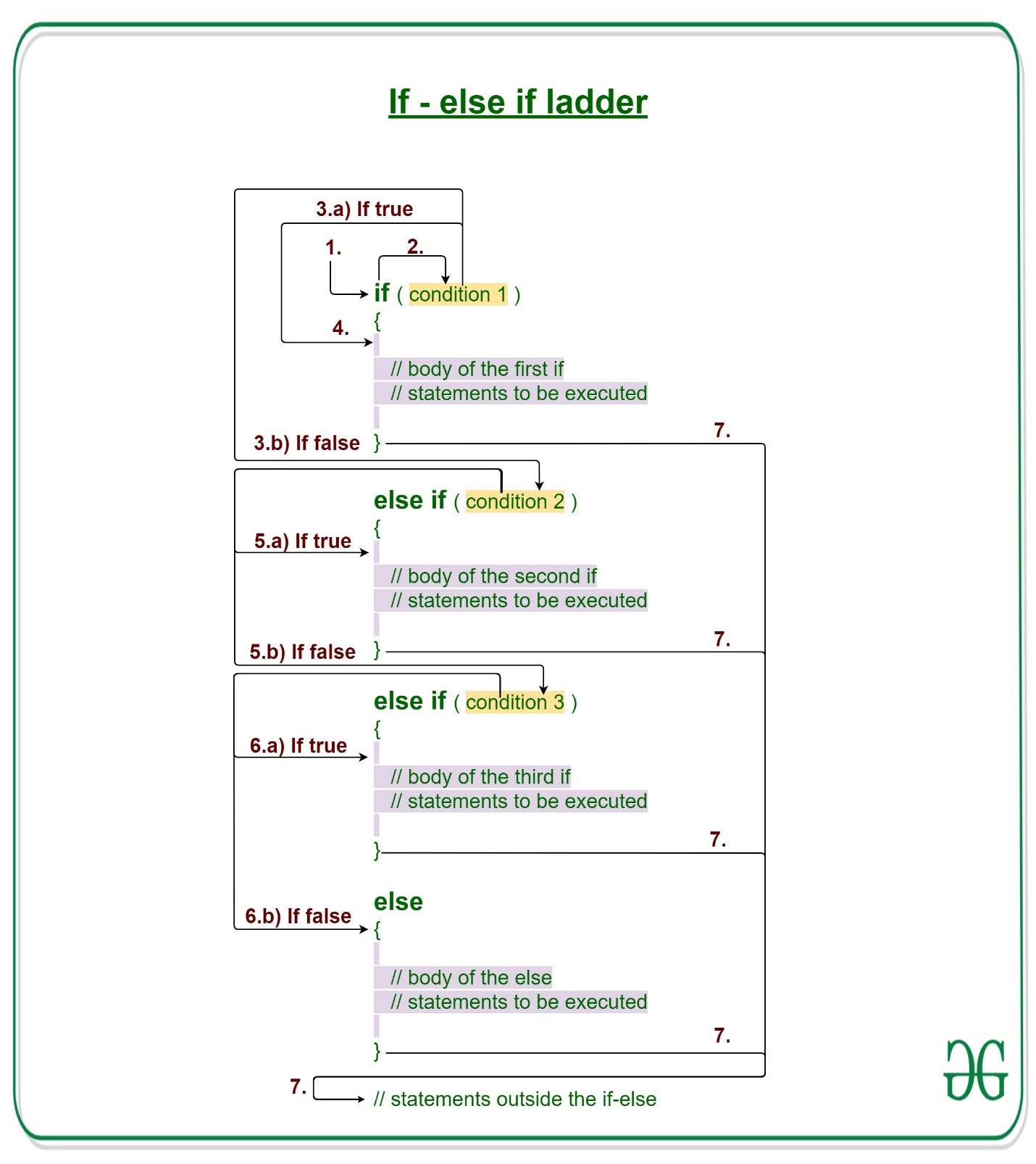
else

System.out.println("i is not present\n");

System.out.println("Outside if-else-if");

}

}



1. Write a java program
2. To find the sum of the elements of the array using for each loop.

|  |
| --- |
| class foreachsum  {      public static void main(String[] args) {          int a[]={1,2,3,4,5};          int sum=0;          for(int i:a)          {              sum=sum+i;          }          System.out.println(sum);      }  } |

1. To search an element in the array using for each loop.

|  |
| --- |
| import java.util.Scanner;  class foreachsearch  {      public static void main(String[] args)      {          Scanner sc=new Scanner(System.in);          System.out.println("entert the element to search for");          int ele=sc.nextInt();          int k=0;          int a[]={1,2,3,4,5};          for(int i:a)          {              if(i==ele)              {                  k=k+1;              }          }          if(k>=1)          {              System.out.println("element "+ele+" found");          }          else          {              System.out.println("element "+ele+" not found");          }      }  } |

1. To check a number is prime or not using for loop.

|  |
| --- |
| import java.util.\*;  class forprime  {      public static void main(String[] args) {          Scanner sc=new Scanner(System.in);          System.out.println("enter the numner to check whether prime or not");          int n=sc.nextInt();          int noofdivisors=0;          for(int i=1;i<=n;i++)          {              if(n%i==0)              {                  noofdivisors=noofdivisors+1;              }          }          if(noofdivisors==2)          {              System.out.println("entered number "+n+" is a prime number");          }          else          {              System.out.println("entered number "+n+" is not a prime number");          }      }  } |

1. To display Fibonacci Series based on the user input using while loop.

|  |
| --- |
| import java.util.Scanner;  class whilefibonacci  {      public static void main(String[] args) {          Scanner sc=new Scanner(System.in);          int i=1;          System.out.println("no of fibonacci squence u want");          int n=sc.nextInt();          int ft=0;          int st=1;          int nt=0;          while(i<=n)          {              System.out.println(ft);              nt=ft+st;              ft=st;              st=nt;              i++;          }      }  } |

1. [To find factorial using while loop](https://beginnersbook.com/2017/09/java-program-to-display-fibonacci-series-using-loops/)

|  |
| --- |
| import java.util.Scanner;  class whilefactorial  {  public static void main(String[] args)  {      Scanner sc=new Scanner(System.in);      System.out.println("enter the number for which u want the factorial");      int n=sc.nextInt();      int fact=1;      while(n>0)      {          fact=fact\*n;          n--;      }      System.out.println(fact);  }  }  s |

Imp:

1))))))))))))))))))))))))))))))))))))))

Pre increment:

 we can say that the pre-increment operator increases the value of the variable first and then use it in the expression.

**Syntax:**

b = ++a;

**For example**, if the initial value of a were 5, then the value 6 would be assigned to b.

Post increment:

 in post-increment value is first used in the expression, and then it is incremented.

**Syntax:**

b = a++;

For example, assume the initial value of a to be 5. Then after executing the above statement the final value of b will be 5 as the value of a will be incremented after performing the expression.

2)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

For each loop use

It is mainly used **to traverse the array or collection elements**. The advantage of the for-each loop is that it eliminates the possibility of bugs and makes the code more readable. It is known as the for-each loop because it traverses each element one by one.