**JSPM's**
**RAJARSHI SHAHU COLLEGE OF ENGINEERING**
**TATHAWADE, PUNE-33**
**(Autonomous Institute Affiliated to Savitribai Phule Pune University)**

# LAB MANUAL

# OF

# Programming Language I

# Contents

## Vision

To create quality computer professionals through an excellent academic environment.

## Mission

1. To empower students with the fundamentals of Computer Engineering for being successful professionals.
2. To motivate the students for higher studies, research, and entrepreneurship by imparting quality education.
3. To create social awareness among the students.

## PEOs

1. Graduate shall have successful professional careers, lead and manage teams.
2. Graduate shall exhibit disciplinary skills to resolve real life problems.
3. Graduate shall evolve as professionals or researchers and continue to learn emerging technologies.

# Program Outcomes

Engineering Graduates will be able to:

**1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

**2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

**6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**12. Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

## Program Specific Outcome

A graduate of the Computer Engineering Program will demonstrate-

**PSO1: Domain Specialization** – Apply domain knowledge to develop computer-based solutions for Engineering Applications.

**PSO2: Problem-Solving Skills -** Find solutions for complex problems by applying problem solving skills and standard practices and strategies in software project development.

**PSO3: Professional Career and Entrepreneurship -** Incorporate professional, social, ethical, effective communication, and entrepreneurial practices into their holistic development.

**Course Outcomes and its mapping with POs and PSOs**

| | | |
|---|---|---|
| CO1 | Use an integrated development environment to write, compile, run, and test simple object-oriented Java programs. | BL 2 |
| CO2 | Read and make elementary modifications to Java programs that solve real-world problems. | BL 2 |
| CO3 | Identify classes, objects, members of a class and relationships among them needed for a specific problem | BL 2 |
| CO4 | Write Java application programs using OOP principles and proper program structuring | BL 3 |

| Sub code Subject | PO | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 | PSO1 | PSO2 | PSO3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS2114 Programming Lab | CO1 | 2 | | | | | | | | | | | | 2 | | |
| | CO2 | 2 | 3 | 2 | | | | | | | | | | 2 | | |
| | CO3 | 2 | | | | | | | | | | | | 2 | | |
| | CO4 | 2 | 2 | | | | | | | | | | 2 | 2 | | 2 |

**Lab Plan**

| Lect. No | Lab Assignment | CO | Teaching Aids | Planned Date | Actual Date |
|---|---|---|---|---|---|
| 1 | To find the largest among three numbers x, y, and z. Use if-then-else construct in Java. | CO1 | Demonstration, Colab. | | |
| 2 | WriteaJavaProgramtoUseEqualsIgnoreCaseMethodInaStringClass. | CO1 | Demonstration | | |
| 3 | Program to remove duplicate | CO1 | Demonstration | | |

| | | | | | |
|---|---|---|---|---|---|
| | elements from sorted array ex. Input : Array={1,2,2,5,6,6,9} result : Array={1,2,5,6,9}. | | | | |
| 4 | Design a class for a bank database the database should support the following operations. 1. Deposit() 2. Withdrawal() 3. Display | CO2 | Demonstration | | |
| 5 | Write a Java program to show parameterized constructor | CO2 | Demonstration | | |
| 6 | Write a Java program to show method Overriding | CO 3 | Demonstration | | |
| 7 | Write a java program to demonstrate a Exception Handling. | CO 4 | Demonstration | | |
| 8 | Write a Java Program to Create a Thread that Implement the Runnable Interface. | CO 3 | Demonstration, You tube video | | |

**JSPM's**
**RAJARSHI SHAHU COLLEGE OF ENGINEERING**
**TATHAWADE, PUNE-33**
**(Autonomous Institute Affiliated to Savitribai Phule Pune University)**

# Lab Assignment no:01

1. **Program to find the largest among three numbers x, y, and z. Use if-then-else construct in Java.**

**Objectives: Understand how to Use if-then-else construct in Java.**

**Problem statement**: Implement a Java program Use if-then-else construct in Java.

Software and Hardware Requirement

1. 64-bit machine
2. Windows 8 / Open-Source Operating System.
3. MySQL Server and Client

**Theory:**

The **if-else statement in Java** is a powerful decision-making tool used to control the program's flow based on conditions. It executes one block of code if a condition is true and another block if the condition is false. In this article, we will learn **Java if-else statement** with examples.

## Importance of If Else Statement:

The importance of if else statements lies in their ability to control the execution of a program. Using if else statements allows developers to apply logic that responds to situations, making programs more versatile and powerful. Whether manipulating user statements, manipulating data, or controlling program flow, if else statements play an important role in programming.

## Basic Syntax of If Else Statement:

**Generally, the basic syntax of if else statements follows this pattern:**

```
if (condition) {
 / / Code block to execute if condition is true
} else
{
// Code block to execute if condition is false
   }
```

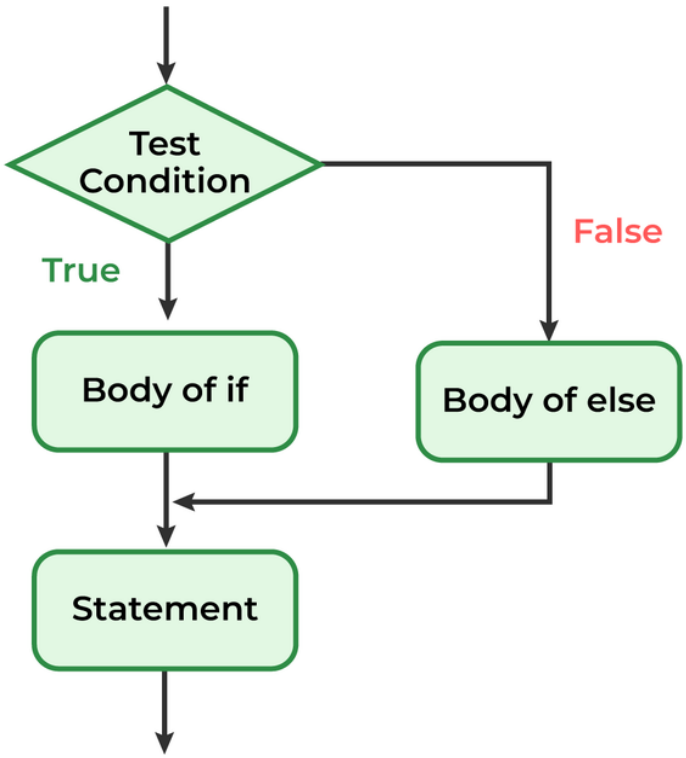
## Working of if-else Statement

1. Control falls into the if block.
2. The flow jumps to the condition.
3. The condition is tested:
    1. If the condition yields true, go to Step 4.
    2. If the condition yields false, go to Step 5.
4. The if block or the body inside the if is executed.

5. If the condition is false, the else block is executed instead.
6. Control exits the if-else block.

**Flowchart of Java if-else Statement**

Below is the Java if-else flowchart.




In the above flowchart of Java if-else, it states that the condition is evaluated, and if it is true, the if block executes; otherwise, the else block executes, followed by the continuation of the program.

JSPM's
RAJARSHI SHAHU COLLEGE OF ENGINEERING
TATHAWADE, PUNE-33
(Autonomous Institute Affiliated to Savitribai Phule Pune University)

**Program to find the largest among three numbers x, y, and z. Use if-then-else construct in Java.**

```
import java.util.Scanner;
 public class LargestNumber {
  public static void main(String[] args) {
// Create a scanner object to read input
   Scanner sc = new Scanner(System.in);

// Input three numbers
System.out.print("Enter the first number (x): ");
int x = sc.nextInt();

System.out.print("Enter the second number (y): ");
int y = sc.nextInt();

System.out.print("Enter the third number (z): ");
int z = sc.nextInt();

// Determine the largest number using if-then-else construct
if (x >= y && x >= z) {
   System.out.println(x + " is the largest number.");
} else if (y >= x && y >= z) {
   System.out.println(y + " is the largest number.");
} else {
   System.out.println(z + " is the largest number.");
}

// Close the scanner
sc.close();
  }
}
```

**Explanation:**

- The program first takes three numbers as input from the user.
- It then compares the values using if-then-else statements.
- The first condition checks if x is greater than or equal to both y and z. If true, x is the largest.
- If the first condition is false, the program checks if y is greater than or equal to both x and z.
- If both the first and second conditions are false, it means z is the largest number.

**Input of the program:**

Enter the first number (x): 12
Enter the second number (y): 8
Enter the third number (z): 15

**Output**
15 is the largest number.

This program works with any integers and will correctly determine the largest among the three.

# Lab Assignment no:02

## Write a Java Program to Use Equals Ignore Case Method in a String Class.

**Objectives:** Understand the concept of **Use Equals Ignore Case Method in a String Class.**

**Problem statement:** Implement a Java program to **Use Equals Ignore Case Method in a String Class**.

Software and Hardware Requirement

1. 64 bit machine
2. Windows 8 / Open Source Operating System.
3. MySQL Server and Client

### Java Program to Use equalsIgnoreCase() Method in the String Class

In Java, the String class provides a method called equalsIgnoreCase() that compares two strings while ignoring case differences. This method is particularly useful when you want to compare two strings without considering their case (uppercase vs. lowercase).

### *Theory:*

The equalsIgnoreCase(String anotherString) method is a case-insensitive comparison method provided by the String class in Java. It compares the content of two strings, but it does not consider the case of the characters in the strings.

**Syntax:** boolean equalsIgnoreCase(String anotherString)

- **Parameter:**
  - Another String: The string to be compared with the current string.
- **Return Value:**
  - true: If the two strings are equal, ignoring case differences.
  - false: If the two strings are not equal, even when ignoring case differences.

***How it works:***

1. It compares the two strings character by character.
2. It ignores the case of each character while performing the comparison.
3. If the strings have the same sequence of characters but differ in case, the method will return true.
4. If the strings are not identical even after ignoring case differences, it will return false.

## Program to Use Equals Ignore Case Method in a String Class

```
public class EqualsIgnoreCaseExample
{
public static void main(String[] args) {
// Create two strings
String str1 = "Hello World";
String str2 = "hello world";

// Compare the strings ignoring case
if (str1.equalsIgnoreCase(str2)) {
    System.out.println("The strings are equal, ignoring case.");
} else {
    System.out.println("The strings are not equal.");
}

// Another example
String str3 = "Java";
String str4 = "JAVA";
if (str3.equalsIgnoreCase(str4)) {
    System.out.println("The strings are equal, ignoring case.");
} else {
    System.out.println("The strings are not equal.");
}
}
```

### *Explanation of the Program:*

1. **String Creation**: We create two string variables str1 and str2 with the values "Hello World" and "hello world".

   Note that the only difference between them is the case of the first letter of the words. Similarly, we create another pair str3 and str4 with the values "Java" and "JAVA".

2. **Comparison with equalsIgnoreCase()**: We use the equalsIgnoreCase() method to compare str1 with

   str2 and str3 with str4. Despite the differences in case, the method will ignore the case and compare

   the strings character by character.

3. **Output**: In both cases, since the strings are identical except for their case, the method will

   return true and print "The strings are equal, ignoring case.".

### *Output:*

```
The strings are equal, ignoring case.
The strings are equal, ignoring case.
```

### *Important Points to Note:*

- The method equalsIgnoreCase() is different from equals(). The equals() method performs a case-sensitive

comparison, while equalsIgnoreCase() ignores case differences.

- This method is particularly useful when you want to perform case-insensitive checks, such as comparing user inpu

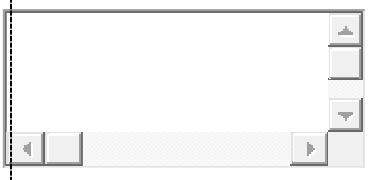
  t or case-insensitive search functionality in applications.

## Summary:

- The equalsIgnoreCase() method is a case-insensitive string comparison method.
- It compares two strings character by character, ignoring the case of the characters.
- This method is useful when the case of the strings does not matter, and you want to compare their

  contents regardless of letter case.

- It returns true if the strings are equal, ignoring case; otherwise, it returns false.



**Program:**

```
  public class EqualsIgnoreCaseExample {
public static void main(String[] args) {
    // Define two strings
 String str1 = "Hello World";
  String str2 = "hello world";

// Compare the strings using equalsIgnoreCase
  if (str1.equalsIgnoreCase(str2)) {
System.out.println("The strings are equal (case ignored).");
  } else {
 System.out.println("The strings are not equal.");
   }
    }
      }
```

**Example Output:**

For the given strings str1 = "Hello World" and str2 = "hello world", the output will be:

The strings are equal (case ignored).

**Explanation:**

1. Two strings, str1 and str2, are defined with values "Hello World" and "hello world".
2. The equalsIgnoreCase() method is used to compare these two strings in a case-insensitive way.
3. If the strings are equal ignoring their case, the program prints: "The strings are equal (case ignored)."
4. Otherwise, it prints: "The strings are not equal."

# Lab Assignment no : 03

**3.Write a Java Program to Remove Duplicate Elements forms Sorted array ex. Input: Array = {1,2,2,5,6,6,9} result: Array= {1,2,5,6,9}**

**Objectives:** The objective of this Java program is to remove duplicate elements from a sorted array and output the array with unique elements only. This ensures that the resulting array contains no repeated values while maintaining the original order of the elements.

**Problem statement:** Given a sorted array, we need to write a Java program that removes all duplicate elements. The array is sorted in non-decreasing order, so the duplicate elements will be adjacent to each other**.**

## Software and Hardware Requirement

1. 64 bit machine
2. Windows 8 / Open-Source Operating System.
3. MySQL Server and Client

## Theory:

In a sorted array, duplicates are located next to each other. By iterating over the array, we can compare each element with the previous one. If the current element is the same as the previous element, it is a duplicate and can be skipped. The remaining unique elements are t hen copied to a new array or updated within the original array.

## To achieve this:

1. We start by iterating through the sorted array.
2. We compare each element with the previous one. If it's different, we add it to the result.
3. Finally, we output the array with the duplicates removed.

**Java Program to Remove Duplicate Elements from Sorted Array:**

```
.
import java.util.Arrays;
public class RemoveDuplicates

{
```

```
public static int[] removeDuplicates(int[] arr)
      {
    // If the array is empty or has only one element, no duplicates are possible
   if (arr == null || arr.length == 0)
    {
    return arr;

      } // To keep track of the number of unique elements
     int uniqueCount = 1;
       // Iterate through the sorted array starting from the second element
    for (int i = 1; i < arr.length; i++)
       {
// If the current element is different from the previous one, it's unique
  if (arr[i] != arr[i - 1])
      {
    arr[uniqueCount] = arr[i];

    uniqueCount++;
        }
        }
       }
       } // Create a new array of size 'uniqueCount' to store the unique elements
     return Arrays.copyOf(arr, uniqueCount);
          }
    public static void main(String[] args)
       {
     // Input: Sorted Array
      int[] arr = {1, 2, 2, 5, 6, 6, 9};
    // Remove duplicates
   int[] result = removeDuplicates(arr);
      // Output the resulting array
    System.out.println("Array with duplicates removed: " + Arrays.toString(result));
                }
               }
```

### Explanation of the Program:

1. **Input Array**: The input array is `{1, 2, 2, 5, 6, 6, 9}`.
2. **removeDuplicates Method**:
   - We first check if the array is `null` or empty. If so, we return the array immediately.
   - We use a variable `uniqueCount` to keep track of how many unique elements we've found.
   - We iterate over the array starting from the second element. If the current element is different from the previous one, we update the position of `uniqueCount` and increase it by 1.
3. **Result**:
   - After removing duplicates, we create a new array of size `uniqueCount` to store the unique elements. We use `Arrays.copyOf` to create a new array with the correct size.

4. **Output**: The program will output the array after duplicates are removed.

**Sample Input and Outpu**

Input:

```
Array = {1, 2, 2, 5, 6, 6, 9}
```

**Original Array: [1, 2, 2, 5, 6, 6, 9] Array afte removing duplicates: [1, 2, 5, 6, 9]**

## Conclusion:

The program successfully removes duplicate elements from a sorted array using the following approach:

- It iterates through the array, comparing each element with the previous one.
- If an element is different from the previous one, it is added to a new result array.
- The result array is resized to only contain unique elements, which are returned as the final output.

**Original Array: [1, 2, 2, 5, 6, 6, 9] Array afte removing duplicates: [1, 2, 5, 6, 9]**

## Conclusion:

The program successfully removes duplicate elements from a sorted array using the following approach:

- It iterates through the array, comparing each element with the previous one.
- If an element is different from the previous one, it is added to a new result array.
- The result array is resized to only contain unique elements, which are returned as the final output.

# Lab Assignment no : 04

## 4.Write a program to Design a class for a bank database the database should support the following operations.

## 1. Deposit()

## 2. Withdrawal()

## 3. Display

**Objectives:** The objective of this program is to design and implement a class-based structure to represent a bank database. The class should provide the necessary functionality to perform operations like depositing money, withdrawing money, and displaying the current balance of an account.

**Problem statement:** The problem at hand is to create a bank account system that supports basic operations for managing account balances. The operations include:

1. **Deposit**: To add money to the account.
2. **Withdrawal**: To subtract money from the account, provided there are sufficient funds.
3. **Display**: To show the current balance of the account.The database should also allow multiple instances (representing different bank accounts)

to be created with unique account numbers and corresponding balances.

**Software and Hardware Requirement**

1. 64 bit machine
2. Windows 8 / Open Source Operating System.
3. MySQL Server and Client

## Theory:

In object-oriented programming (OOP), we encapsulate data and behaviors together in the form of classes.For this case, we can design a `BankAccount` class,

which will have the following attributes and methods:

- **Attributes**: The balance of the account, account holder's name, and account number.
- **Methods**:

**Deposit()**: This method will take an amount as input and add it to the account balance.

**Withdrawal()**: This method will take an amount as input and subtract it from the account balance, if the balance is sufficient.

**Display()**: This method will print the account details, including the balance.

A good practice in banking systems is to ensure that a withdrawal operation doesn't result in a negative balance unless there is an overdraft facility.

For simplicity, this example assumes no overdraft is allowed.

## program to Design a class for a bank database the database should support the following operations

```
public class BankAccount {

// Instance variables (attributes)
private String accountHolder;
private String accountNumber;
private double balance;

// Constructor to initialize the bank account
public BankAccount(String accountHolder, String accountNumber, double initialBalance) {
    this.accountHolder = accountHolder;
    this.accountNumber = accountNumber;


this.balance = initialBalance;
}

// Method to deposit money into the account
```

```
  }

   ublic void deposit(double amount) {
   if (amount > 0) {



balance += amount;
       System.out.println("Deposited " + amount + ". New balance: " + balance);
   } else {
       System.out.println("Deposit amount must be positive.");
   }
}

// Method to withdraw money from the account
public void withdraw(double amount) {
   if (amount > 0) {
       if (amount <= balance) {
           balance -= amount;
           System.out.println("Withdrew " + amount + ". New balance: " + balance);
       } else {
           System.out.println("Insufficient funds for withdrawal.");
       }
   } else {
       System.out.println("Withdrawal amount must be positive.");
   }
}

// Method to display account details
public void display() {


  System.out.println("Account Holder: " + accountHolder);
   System.out.println("Account Number: " + accountNumber);
   System.out.println("Current Balance: " + balance);
}

// Main method to test the BankAccount class
public static void main(String[] args) {
   // Create a new bank account instance
   BankAccount account1 = new BankAccount("John Doe", "123456789", 5000.0);

   // Display account details
   account1.display();

   // Perform deposit operation
   account1.deposit(1500.0);

   // Perform withdrawal operation
   account1.withdraw(2000.0);

   // Display updated account details
   account1.display();

   // Attempting to withdraw more than the balance
```

aC  .Wi.draw(5000.0);
}

**Explanation of the Code:**

1. **Attributes**:
2. 
   - accountHolder: Holds the name of the account holder.
   - accountNumber: Holds the account number.
   - balance: Holds the balance of the account.
3. **Constructor**: The BankAccount constructor initializes an instance with the account holder's name, account number, and an initial balance.
4. **Methods**:
   - **deposit(double amount)**: Adds the given positive amount to the balance.
   - **withdraw(double amount)**: Subtracts the given positive amount from the balance if the balance is sufficient.
   - **display()**: Displays the account details including the account holder's name, account number , and the current balance.
5. **Main Method**:
   - In the main method, a BankAccount object is created, and then deposit and withdrawal operations are performed. After each operation, the account details are displayed to show the updated balance.

**Output:**

Account Holder: John Doe
Account Number: 123456789
Current Balance: 5000.0
Deposited 1500.0. New balance: 6500.0
Withdraw 2000.0. New balance: 4500.0
Account Holder: John Doe
Account Number: 123456789

Current Balance: 4500.0

Insufficient funds for withdrawal.

**Conclusion:**

This Java program effectively models a simple bank account system, supporting basic functionality for depositing, withdrawing, and displaying account details. The program ensures that users cannot withdraw more than the available balance and provides feedback on each operation.

# Lab Assignment no: 05

## 5.Write a Java program to show parameterized constructor

**Objective:** A parameterized constructor allows you to initialize an object with values passed as arguments at the time of object creation. This enhances the flexibility of object initialization and allows the creation of objects with different initial states.

**Problem statement:** We need to create a class with a parameterized constructor that accepts parameters to initialize the instance variables of the class.
The program should:

1. Accept parameters for initializing object attributes.
2. Print the details of the object after initialization to show how the constructor works.

## Software and Hardware Requirement

1. 64 bit machine
2. Windows 8 / Open Source Operating System.
3. MySQL Server and Client

## Theory:

A **constructor** is a special method used to initialize objects. It is called when an object of the class is created. A **parameterized constructor** is a constructor that takes arguments (parameters) to initialize the instance variables of the class.

In this program, the constructor `Car(String make, String model, int year)` is parameterized.

It takes three parameters:

- `make` (a `String` representing the car's manufacturer),
- `model` (a `String` representing the car's model),
- `year` (an `int` representing the car's manufacturing year).

The `this` keyword refers to the current instance of the object. It is used here to distinguish the instance variables (`make`, `model`, and `year`) from the constructor parameters that have the same names.

When an object is created using this constructor, the values passed as arguments

are assigned to the instance variables of that object.

In object-oriented programming (OOP), a constructor is a special method used to initialize objects. There are two types of constructors:

1. **Default Constructor**: A constructor with no parameters.
2. **Parameterized Constructor**: A constructor that takes parameters to initialize object attributes at the time of object creation.

A **parameterized constructor** provides a way to initialize an object with specific values,

rather than using default values. This is particularly useful when creating objects with different attributes.

## program to show parameterized constructor

```
public class Car {
  // Instance variables
   String make;
   String model;
   int year;

  // Parameterized constructor
   public Car(String make, String model, int year) {
      this.make = make;
      this.model = model;
      this.year = year;
  }

  // Method to display car details
   public void displayCarInfo() {
      System.out.println("Car Make: " + make);
      System.out.println("Car Model: " + model);
      System.out.println("Car Year: " + year);
  }

    public static void main(String[] args) {
      // Creating objects using the parameterized constructor
      Car car1 = new Car("Toyota", "Corolla", 2020);
      Car car2 = new Car("Honda", "Civic", 2022);

      // Displaying car details
      car1.displayCarInfo();
      System.out.println();
      car2.displayCarInfo();
  }
   }
```

### Explanation:

- **Parameterized Constructor**: The constructor `Car(String make, String model, int year)` takes three parameters (`make`, `model`, and `year`) to initialize the instance variables of the class.
- **Objects Creation**: In the `main` method, two objects (`car1` and `car2`) are created using the parameterized constructor with specific values for the car make, model, and year.
- **Display Method:** The `displayCarInfo()` method prints the details of the car.

### Sample Output:

Car Make: Toyota

Car Model: Corolla
Car Year: 2020

Car Make: Honda
Car Model: Civic
Car Year: 2022

## Conclusion:

In summary, the **parameterized constructor** in Java enables you to initialize an object's state at the time of creation, making your code more efficient, flexible, and clean. It enhances encapsulation, as objects are immediately initialized with valid data, and it reduces the need for repetitive code.

# Lab Assignment no : 06

## 6. Write a Java program to show method Overriding

**Objectives:** To demonstrate the concept of method overriding in Java, where a subclass provides a specific implementation of a method that is already defined in its superclass

**Problem statement:** We have two classes: `Animal` (superclass) and `Dog` (subclass).
The superclass `Animal` has a method `sound()`, which is overridden in the subclass `Dog`.
The task is to create a Java program that:

1. Demonstrates the use of method overriding.
2. Shows that the overridden method in the subclass is invoked at runtime, depending on the actual object type.

.

## Software and Hardware Requirement

1. 64 bit machine
2. Windows 8 / Open Source Operating System.
3. MySQL Server and Client

## Theory:

**Method Overriding** in Java occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. This allows the subclass to modify the behavior of the method inherited from the superclass.

### Key points about method overriding:

1. **Same Method Signature**: The method in the subclass must have the same name, return type, and parameters as the method in the superclass.
2. **Runtime Polymorphism**: The method that gets called is determined at runtime based on the object's actual class, not the reference type.

3. **@Override Annotation**: Although optional, it is recommended to use the @Override annotation to avoid errors, as it tells the compiler that the method is intended to override a method in the superclass.

```
class Animal {
// Method in the superclass
public void sound() {
   System.out.println("Animals make sound");
}
}

// Subclass
class Dog extends Animal {
 // Method overriding in the subclass
 @Override
  public void sound() {
   System.out.println("Dog barks");
 }
}

public class MethodOverridingExample {
public static void main(String[] args) {
   // Creating objects of Animal and Dog
   Animal myAnimal = new Animal();  // Superclass reference
   Animal myDog = new Dog();         // Subclass reference, but object is Dog

   // Calling the overridden method
   myAnimal.sound();  // Output: Animals make sound
   myDog.sound();     // Output: Dog barks

}
```

## Explanation of the Program:

1. **Superclass (`Animal`)**:
   - The Animal class has a method sound(), which prints "Animals make sound". This is the method in the superclass.
2. **Subclass (`Dog`)**:
   - The Dog class extends the Animal class and overrides the sound() method. In the subclass, the sound() method prints "Dog barks".
   - The @Override annotation is used to ensure that the method in Dog correctly overrides the method in Animal.

## 3. **Method Overriding in Action**:

- In the main() method, two objects are created:
  - myAnimal is a reference of type Animal pointing to an object of type Animal.

myDog is a reference of type Animal pointing to an object of type Dog (this is an example of polymorphism).

- When myAnimal.sound() is called, the method in the Animal class is invoked.
- When myDog.sound() is called, even though the reference type is Animal, the overridden sound() method in the Dog class is called due to the actual object being of type Dog.

**Conclusion:**

Method overriding is a powerful feature in Java that allows subclasses to provide their own implementation of a method defined in the superclass. It is fundamental to achieving runtime polymorphism and allows for the creation of more flexible and extensible object-oriented designs.

# Lab Assignment no:0 7

## 7. Write a java program to demonstrate a Exception Handling

**Objectives:** The objective of this Java program is to demonstrate how exception handling works in Java using `try`, `catch`, and `finally` blocks. Exception handling is crucial to manage runtime errors, maintain the normal flow of the program, and ensure the program continues running smoothly even if an error occurs.

**Problem statement:** In a Java application, you need to handle runtime errors such as division by zero, accessing an invalid array index, or opening a file that does not exist.
The goal is to write a Java program that:

1. Demonstrates how to handle exceptions using `try` and `catch`.
2. Ensures that the program continues to run even after an exception occurs.
3. Uses `finally` to execute code regardless of whether an exception was thrown or not.

### Software and Hardware Requirement

1. 64 bit machine
2. Windows 8 / Open Source Operating System.
3. MySQL Server and Client

### Theory of Exception Handling:

**Exception Handling** in Java provides a way to handle runtime errors, which helps maintain the normal flow of the program. Java uses several keywords (try, catch, throw, throws, and finally) to handle exceptions.

1. **Try Block**: The code that might throw an exception is placed inside the `try` block.
2. **Catch Block**: The `catch` block is used to handle the exception. The type of exception to be caught must be specified in the `catch` block.

3. **Finally Block**: This block is optional. It is used to execute code that should run regardless of whether an exception was thrown or not, such as closing resources (files, database connections, etc.).
4. **Throw and Throws**: The `throw` keyword is used to explicitly throw an exception, and `throws` is used in method signatures to indicate that a method might throw an exception.

## Types of Exceptions in Java:

- **Checked Exceptions**: These exceptions are checked at compile time (e.g., `IOException`).
- **Unchecked Exceptions**: These exceptions are not checked at compile time, and occur during runtime (e.g., `ArithmeticException`, `NullPointerException`).

## java program to demonstrate a Exception Handling

```
public class FullExceptionHandlingDemo
      {
public static void main(String[] args)
     {
       // Example 1: Handling ArithmeticException (Division by zero)
    try
    {
    int result = 10 / 0;
   // This will throw ArithmeticException
      System.out.println("Result of division: " + result);
        }
        catch (ArithmeticException e)
           {
     System.out.println("Error: Cannot divide by zero!");
        }

// Example 2: Handling ArrayIndexOutOfBoundsException (Invalid array index)
      try
  {
   int[] numbers = {1, 2, 3};
     System.out.println(numbers[5]);
```

```
// This will throw ArrayIndexOutOfBoundsException
            }


 catch (ArrayIndexOutOfBoundsException e)
          {
        System.out.println("Error: Array index is out of bounds!");
                }

// Example 3: Handling NullPointerException (Accessing method on null object)
      try
         {
     String str = null; System.out.println(str.length());
        // This will throw NullPointerException
           }
      catch (NullPointerException e)
        {
  System.out.println("Error: NullPointerException occurred!");
       }

      // Example 4: Finally block (Executing code that should run regardless of exception)
         try
        {
       System.out.println("Executing code in try block.");
         }
        finally
           {
       System.out.println("Finally block is always executed.");
         }
// Example 5: Handling Multiple Exceptions in a Single catch Block
        try
         {
         int[] numbers = new int[3]; numbers[3] = 5;
          // This will throw ArrayIndexOutOfBoundsException int divisionResult = 10 / 0;
         // This will throw ArithmeticException
           }
         catch (ArithmeticException | ArrayIndexOutOfBoundsException e)
      {
                System.out.println("Error: " + e);
            }

 // Example 6: Handling Custom Exception (Throwing an Exception manually)
 Try
  {
     validateAge(15);
   // This will throw a custom exception
```

```
}
    catch (IllegalArgumentException e)
      {
      System.out.println("Error: " + e.getMessage());
            }
// Custom exception throwing method
    public static void validateAge(int age)
     {
      if (age < 18)
    {
      throw new IllegalArgumentException("Age must be 18 or older.");
        }
      Else
       {
      System.out.println("Age is valid.");

       }

   }
    }
```

## Explanation of the Program:

1. **Example 1: Handling ArithmeticException:**
   - We attempt to divide 10 by 0, which will throw an ArithmeticException.
   - The exception is caught and an appropriate error message is displayed.
2. **Example 2: Handling ArrayIndexOutOfBoundsException:**
   - We try to access an invalid index in an array, which throws ArrayIndexOutOfBoundsException.

     This is caught and handled.
3. **Example 3: Handling NullPointerException:**
   - We attempt to call the length() method on a null string, which results in a NullPointerException.

     This is caught and handled.
4. **Example 4: The finally Block:**
   - The finally block ensures that code inside it runs regardless of whether an exception was thrown

     or not.

     It is useful for cleanup actions, like closing files or database connections.
5. **Example 5: Handling Multiple Exceptions in a Single catch Block:**

his example demonstrates how you can catch multiple exceptions using a single `catch` block (introduced in Java 7). In this case, both `ArithmeticException` and `ArrayIndexOutOfBoundsException` are caught and handled by the same block.

6. **Example 6: Throwing Custom Exception:**
   - A method validateAge() is created, which throws a custom exception (IllegalArgumentException) if the age is less than 18. This is caught and an appropriate message is displayed.

7. **Example 7: Nested `try-catch`:**
   - In this example, there's a nested `try-catch` block. The inner `try` block throws an exception (`ArithmeticException`), which is handled by the inner `catch` block. The
   - outer `catch` block handles any other exceptions that may occur in the outer `try`.

## Sample Output:

```
Error: Cannot divide by zero!
Error: Array index is out of bounds!
Error: NullPointerException occurred!
Executing code in try block.
Finally block is always executed.
Error: java.lang.ArithmeticException: / by zero
Error: Age must be 18 or older.
Entering outer try block.
Error in inner try block: Cannot divide by zero.
```

## Conclusion:

This program demonstrates how exception handling in Java works using:

- try blocks to contain code that might throw an exception,
- catch blocks to handle exceptions and prevent program termination,
- finally blocks to execute code regardless of exceptions,
- The ability to catch multiple exceptions in a single catch block (introduced in Java 7),
- Throwing and catching custom exceptions, and
- Nested try-catch blocks to handle exceptions at different levels.

# Lab Assignment no: 08

## 8. Write a Java Program to Create a Thread that Implement the Runnable Interface

**Objectives:** The objective of this Java program is to create a thread by implementing the Runnable interface, which will allow a task to be executed concurrently.

By using Runnable, we can separate the task from the thread management.

This approach is often preferred over extending the Thread class when a class needs to extend another class while also using threads.

**Problem statement:** We need to create a simple Java program that:

1. Implements the Runnable interface.
2. Creates a thread that executes a task (such as printing numbers from 1 to 10).
3. Starts the thread to run the task concurrently.

### Software and Hardware Requirement

1. 64 bit machine
2. Windows 8 / Open Source Operating System.
3. MySQL Server and Client

### Theory:

In Java, a thread is a lightweight process that allows multiple operations to run concurrently.

Threads can be created in two primary ways:

1. By extending the `Thread` class.
2. By implementing the `Runnable` interface.

When using the `Runnable` interface, we define the code to be executed by the thread in the `run()` method.

To actually execute the thread, we pass the `Runnable` object to a `Thread`

object and start it using the `start()` method.

The `Runnable` interface has a single method, `run()`, which contains the code to be executed

## Java Program to Create a Thread that Implement the Runnable Interface

```
public class MyRunnable implements Runnable
    {
  // Override the run method to define the task to be executed by the thread @Override
    public void run()
  { // Task to be executed by the thread for (int i = 1; i <= 10; i++)
    { System.out.println(Thread.currentThread().getId() + " Value: " + i);
     try
  {
   Thread.sleep(500); // 500 milliseconds
        }
    catch (InterruptedException e)
    {
     System.out.println(e);
     }
    }
     }
     public static void main(String[] args)
        {
   // Create an instance of MyRunnable
   MyRunnable myRunnable = new MyRunnable();
// Create a thread and pass the Runnable instance to it Thread
   thread1 = new Thread(myRunnable);
 // Start the thread
  thread1.start();
 // Optionally, create another thread to demonstrate multiple threads
   Thread thread2 = new Thread(myRunnable);
  thread2.start();
// Main thread can continue doing other tasks if necessary
 System.out.println("Main thread is executing concurrently with other threads.");
 }

}
```

## Explanation of the Program:

1. **`MyRunnable` class**: This class implements the `Runnable` interface and overrides its `run()` method.

   In the `run()` method, a simple task is defined — printing numbers from 1 to 10 with a delay of 500 milliseconds between each print statement. This simulates some work being done.

2. **`Thread` creation and execution**:
   - The main method creates an instance of MyRunnable.
   - A new Thread object is created and the Runnable object (myRunnable) is passed to the constructor of Thread.
   - The start() method is called to initiate the execution of the run() method in a separate thread.
3. **Concurrency**:
   - When thread1`.start()` and `thread2.start()` are invoked, both threads will execute concurrently and print numbers from 1 to 10 independently.

   This demonstrates the ability to run multiple threads concurrently.
4. **Thread.sleep()**:

   The Thread.sleep(500) method is used to simulate some time-consuming task allowing other threads to execute while one is "sleeping."

## Output:

```
Main thread is executing concurrently with other threads.
1 Value: 1
                                        1 Value: 2
                                        1 Value: 3
                                        1 Value: 4
                                        1 Value: 5
                                        1 Value: 6
                                        1 Value: 7
                                        1 Value: 8
                                        1 Value: 9
                                       1 Value: 10
                                        2 Value: 1
                                        2 Value: 2
                                        2 Value: 3
```

2 Value: 4
2 Value: 5
2 Value: 6
2 Value: 7
2 Value: 8
2 Value: 9
2 Value: 10

## In the output:

- The threads (`thread1` and `thread2`) run concurrently, printing the numbers 1 to 10 in parallel.
- The `main` thread executes independently and prints a message that it is executing concurrently.

## Conclusion:

By implementing the Runnable interface, we achieve thread-based concurrency in Java without directly inheriting from the Thread class. This approach is more flexible because a class can implement multiple interfaces, while it can only extend one class.