Lab Manual

Course Title: Object Oriented Programming with JAVA Sessional

Course Code: CCE – 1208

Course Credit: 1

Ahsanul Karim Tanim (E221013)

Title: Object Oriented Programming with JAVA Sessional
Course Code: CCE – 1208

Lab No.	Lab Title	Topics Covered
Lab 1	**Demonstration of Java Basics**	Installation of JDK & IDE, Writing & Running a Simple Java Program
Lab 2	**Demonstration of Control Statements & Input Handling**	if, if-else, switch, Taking Input using Scanner
Lab 3	**Demonstration of Loops in Java**	for, while, do-while, break/continue, Nested Loops
Lab 4	**Demonstration of Class and Object (main() inside a single class)**	Creating Objects, Calling Methods, Understanding OOP Basics
Lab 5	**Demonstration of Class and Object (main() outside the class)**	Creating Objects, Calling Methods, main() outside class
Lab 6	**Demonstration of Inheritance in Java**	Multilevel & Hierarchical Inheritance, Constructors in Inheritance
Lab 7	**Demonstration of Polymorphism using Method Overloading**	Compile-time Polymorphism, Examples using Arithmetic Operations
Lab 8	**Demonstration of Polymorphism using Method Overriding**	Run-time Polymorphism, Overriding Methods in Parent & Child Classes
Lab 9	**Demonstration of Encapsulation**	Using Private Variables, Getter & Setter Methods for Data Hiding
Lab 10	**Demonstration of Abstraction & Interfaces**	Abstract Classes, Interfaces

Lab 1: Demonstration of Java Basics

What is JAVA?
Java একটি শক্তিশালী এবং বহুল ব্যবহৃত প্রোগ্রামিং ভাষা যা তার পোর্টেবিলিটি, সিকিউরিটি এবং বহুমুখিতার জন্য পরিচিত। এটি Sun Microsystems (বর্তমানে Oracle Corporation-এর মালিকানাধীন) দ্বারা তৈরি হয়। Java এর জনপ্রিয়তার মূল কারণ হলো এর বিভিন্ন বৈশিষ্ট্য এবং এটির দ্বারা সমর্থিত বিভিন্ন অ্যাপ্লিকেশনের বিস্তৃত পরিসর।

Why Study JAVA?

- **প্ল্যাটফর্ম ইনডিপেনডেন্স**: Java প্ল্যাটফর্ম নির্ভরশীল নয়, অর্থাৎ, Java-তে লেখা প্রোগ্রামগুলি বিভিন্ন অপারেটিং সিস্টেমে চলতে পারে কোনও পরিবর্তন ছাড়াই।
- **সিকিউরিটি**: Java এর সিকিউরিটি ফিচারগুলি ক্ষতিকর কোড এবং অননুমোদিত অ্যাক্সেস থেকে সুরক্ষা প্রদান করে।
- **বহুমুখিতা**: Java ব্যবহার করা হয় বিভিন্ন প্রয়োজনে যেমন ওয়েব ডেভেলপমেন্ট, মোবাইল অ্যাপ ডেভেলপমেন্ট (Android), ডেস্কটপ অ্যাপ্লিকেশন, এবং এমবেডেড সিস্টেমে।
- **বৃহৎ ডেভেলপার কমিউনিটি**: Java এর একটি বিশাল ডেভেলপার কমিউনিটি রয়েছে যা ভাষাটির উন্নতি সাধন করে এবং নতুন রিসোর্স সরবরাহ করে।
- **রিচ স্ট্যান্ডার্ড লাইব্রেরি**: Java এর বিস্তৃত লাইব্রেরি ব্যবহার করে প্রোগ্রামিং কাজগুলি সহজ হয়ে যায়।

Java এর ব্যবহার (Applications of Java)

Java একটি বহুমুখী প্রোগ্রামিং ভাষা, যা বিভিন্ন ক্ষেত্রে ব্যবহৃত হয়। এর কিছু প্রধান ব্যবহার নিচে দেওয়া হলো—

১. ডেস্কটপ অ্যাপ্লিকেশন

- Java ব্যবহার করে **দৃশ্যমান (GUI) ডেস্কটপ অ্যাপ্লিকেশন** তৈরি করা যায়।
- **Swing, JavaFX, এবং AWT** ব্যবহার করে ডেস্কটপ সফটওয়্যার বানানো হয়।
- **উদাহরণ:** Eclipse IDE, IntelliJ IDEA, NetBeans

২. ওয়েব ডেভেলপমেন্ট

- Java দিয়ে **ডায়নামিক ওয়েবসাইট এবং ওয়েব অ্যাপ তৈরি করা যায়**।
- **Spring, Hibernate, JSP, Servlets** ইত্যাদি ফ্রেমওয়ার্ক ব্যবহৃত হয়।
- **উদাহরণ:** Amazon, LinkedIn, Flipkart

৩. মোবাইল অ্যাপ ডেভেলপমেন্ট

- **Android অ্যাপ তৈরি করার জন্য Java সবচেয়ে জনপ্রিয় ভাষা।**

- **Android Studio ও Java API** ব্যবহার করা হয়।
- **উদাহরণ:** WhatsApp, Instagram, Google Maps (আগের ভার্সন)

৪. এন্টারপ্রাইজ অ্যাপ্লিকেশন

- বড় বড় কোম্পানির **ব্যাকএন্ড সফটওয়্যার এবং সার্ভার সিস্টেম** তৈরি করা হয়।
- **Spring Boot, J2EE, Hibernate** ইত্যাদি প্রযুক্তি ব্যবহৃত হয়।
- **উদাহরণ:** ব্যাংকিং সফটওয়্যার, কর্পোরেট ম্যানেজমেন্ট সিস্টেম

৫. গেম ডেভেলপমেন্ট

- **2D ও 3D গেম** তৈরি করা যায়।
- **LibGDX, jMonkeyEngine** ফ্রেমওয়ার্ক ব্যবহৃত হয়।
- **উদাহরণ:** Minecraft (Java Edition)

Camel Case-এর ব্যবহার (CamelCase Convention in Programming)

Camel Case হলো একটি naming convention, যেখানে প্রথম শব্দটি ছোট হাতের অক্ষর দিয়ে শুরু হয় এবং পরবর্তী প্রতিটি শব্দ বড় হাতের অক্ষর দিয়ে শুরু হয়। এটি variables, methods, এবং functions-এর নামকরণের জন্য ব্যবহৃত হয়।

Camel Case-এর ধরন

1. **lowerCamelCase** → প্রথম শব্দ ছোট হাতের অক্ষরে শুরু হয়, পরবর্তী শব্দগুলোর প্রথম অক্ষর বড় হাতের হয়।
   ব্যবহার: Variables, Methods, Function Names
   উদাহরণ: int studentAge;

   String fullName;

   void calculateTotalMarks() { }
2. **UpperCamelCase (Pascal Case)** → প্রতিটি শব্দের প্রথম অক্ষর বড় হাতের হয়।
   ব্যবহার: Class এবং Interface Names
   উদাহরণ:class StudentDetails { }

   interface DataProcessor { }

Structure of a Typical Java Program

- **ডকুমেন্টেশন সেকশন** (ঐচ্ছিক): প্রোগ্রামের নাম, লেখক এবং অন্যান্য তথ্য সংবলিত মন্তব্য।
- **প্যাকেজ স্টেটমেন্ট** (ঐচ্ছিক): Java ক্লাসগুলি প্যাকেজে বিভক্ত হয়। উদাহরণ: package student;
- **ইম্পোর্ট স্টেটমেন্ট**: অন্য প্যাকেজের ক্লাস ব্যবহারের জন্য ইম্পোর্ট করা হয়। উদাহরণ: import java.lang.Math.*;
- **ইন্টারফেস স্টেটমেন্ট** (ঐচ্ছিক): যদি মুলত: একাধিক ইনহেরিট্যান্স ব্যবহার করতে হয়।
- **ক্লাস ডেফিনিশন**: Java প্রোগ্রামের মুল উপাদান।
- **মেইন মেথড**: প্রতিটি Java প্রোগ্রাম মেইন মেথড দিয়ে শুরু হয়।

* System.out.println()

println() (print line) প্রতিটি আউটপুট শেষে **নতুন লাইন** তৈরি করে।
সহজভাবে স্ট্রিং, সংখ্যা, এবং ভেরিয়েবলের মান প্রিন্ট করতে ব্যবহৃত হয়।

* System.out.printf()

printf() (print formatted) **ফরম্যাটিং-এর জন্য ব্যবহৃত হয়**।
নতুন লাইন **স্বয়ংক্রিয়ভাবে তৈরি করে না**, এজন্য **\n(Escape Key) ব্যবহার করতে হয়**।
স্পেসিফায়ার (%d, %s, %f) ব্যবহার করে **ফরম্যাট ঠিক রাখা যায়**।

Example of a Simple Java Program

```java
public class Main {
    public static void main(String[] args) {
        int age = 23;
        String name = "Tanim";
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Example of JAVA program with typical structure

```java
/**
 * This is a Sample JAVA program with typical structure
 * @author  Ahsanul Karim Tanim
 * @version 1.0
 * @since   12-02-2025
 */


// Package Statement (Optional)
package mypackage;
```

```java
// Import Statements (Optional)
import java.util.Scanner;

// Interface Statements (Optional)
interface MyInterface {
    void displayMessage();
}

// Class Definitions
class MyClass implements MyInterface {
    // Data members
    private String message;

    // Constructor
    MyClass(String msg) {
        this.message = msg;
    }

    // Method to display message
    public void displayMessage() {
        System.out.println("Message: " + message);
    }
}

// Main Method Class (Essential)
public class MainProgram {
    public static void main(String[] args) {
        // Creating an object of MyClass
        MyClass obj = new MyClass("Hello, Java!");

        // Calling method
        obj.displayMessage();
    }
}
```


Lab 1 :

Task 1: Installation of JAVA and IDE

Task 2: Write A Java program to display your information (name, Student Id, age, Email, Phone, Address)

Lab 2: Demonstration of Control Statements & Input Handling

কন্ট্রোল স্টেটমেন্ট কী?

কন্ট্রোল স্টেটমেন্টের মাধ্যমে **প্রোগ্রামের কার্যপ্রবাহ নিয়ন্ত্রণ** করা হয়। এটি প্রধানত তিন প্রকার:

1. **Selection Statements** → if, if-else, switch
2. **Looping Statements** → for, while, do-while
3. **Jump Statements** → break, continue, return

এই ল্যাবে আমরা **Selection Statements** ও **ইনপুট নেওয়ার পদ্ধতি** শিখবো।

ইনপুট নেওয়া (Scanner ব্যবহার করে)

Java-তে ইউজার ইনপুট নিতে হলে Scanner ক্লাস ব্যবহার করতে হয়। এটি **java.util.Scanner** লাইব্রেরি থেকে আসে।

ইউজার থেকে সংখ্যা ইনপুট নেওয়া

```java
import java.util.Scanner;  // Scanner ক্লাস ইম্পোর্ট করা

public class UserInput {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in); // Scanner অবজেক্ট তৈরি
        System.out.print("Enter your age: ");
        int age = sc.nextInt();  // ইউজার থেকে পূর্ণসংখ্যা ইনপুট নেওয়া
        System.out.println("Your age is: " + age);
        sc.close();  // Scanner বন্ধ করা (অপশনাল)
    }
}
```

- Scanner sc = new Scanner(System.in); → ইনপুট নেওয়ার জন্য sc Scanner অবজেক্ট তৈরি করা

`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user

`nextShort()`	Reads a `short` value from the user

Operators In Java

1. Arithmetic Operators

সংখ্যাগত গণনা করার জন্য ব্যবহৃত হয়।

অপারেটর	অর্থ	উদাহরণ	আউটপুট
+	যোগ	10 + 5	15
−	বিয়োগ	10 − 5	5
*	গুণ	10 * 5	50
/	ভাগ	10 / 5	2
%	মডুলাস (ভাগশেষ)	10 % 3	1

2. Assignment Operators

ভেরিয়েবলে মান সংরক্ষণের জন্য ব্যবহৃত হয়।

অপারেটর	অর্থ	উদাহরণ	
=	মান সেট করা	a = 10	a = 10
+=	যোগ ও অ্যাসাইন	a += 5	a = a + 5
−=	বিয়োগ ও অ্যাসাইন	a −= 5	a = a − 5
*=	গুণ ও অ্যাসাইন	a *= 5	a = a * 5
/=	ভাগ ও অ্যাসাইন	a /= 5	a = a / 5
%=	মডুলাস ও অ্যাসাইন	a %= 5	a = a % 5

3. Relational/Comparison Operators

দুটি মানের মধ্যে তুলনা করতে ব্যবহৃত হয়।

অপারেটর	অর্থ	উদাহরণ	আউটপুট
==	সমান কিনা	10 == 10	true
!=	অসমান কিনা	10 != 5	true
>	বড় কিনা	10 > 5	true
<	ছোট কিনা	10 < 5	false
>=	বড় বা সমান কিনা	10 >= 10	true

<=	ছোট বা সমান কিনা	10 <= 5	false

4. Logical Operators

লজিক্যাল অপারেটর দুই বা ততোধিক শর্ত চেক করতে ব্যবহার করা হয়।

অপারেটর	অর্থ	উদাহরণ	আউটপুট
&&	**AND**	(10 > 5) && (5 < 8)	true
||	**OR**	(10 > 5) || (5 > 8)	true
!	**NOT**	!(10 > 5)	false

Example

```
public class OperatorExample {
    public static void main(String[] args) {
        // 1. Arithmetic Operators
        int a = 10, b = 5;
        System.out.println("Addition: " + (a + b)); // 15
        System.out.println("Subtraction: " + (a - b)); // 5
        System.out.println("Multiplication: " + (a * b)); // 50
        System.out.println("Division: " + (a / b)); // 2
        System.out.println("Modulus: " + (a % 3)); // 1

        // 2. Assignment Operators
        int x = 10;
        x += 5; // x = x + 5
        System.out.println("After x += 5: " + x); // 15

        // 3. Comparison Operators
        System.out.println("a == b: " + (a == b)); // false
        System.out.println("a > b: " + (a > b)); // true
        System.out.println("a =< b: " + (a =< b)); // false

        // 4. Logical Operators
        boolean cond1 = (a > b); // true
        boolean cond2 = (b == 5); // true
        System.out.println("cond1 && cond2: " + (cond1 && cond2)); // true
        System.out.println("cond1 || (b > 10): " + (cond1 || (b > 10))); // true
        System.out.println("!(a == 10): " + !(a == 10)); // false
    }
}
```

if এবং if-else স্টেটমেন্ট

if স্টেটমেন্ট দিয়ে আমরা **শর্ত অনুযায়ী সিদ্ধান্ত নিতে পারি**।

সংখ্যা পজিটিভ, নেগেটিভ নাকি শূন্য তা নির্ণয় করা

```java
import java.util.Scanner;

public class NumberCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        if (num > 0) {
            System.out.println("The number is Positive.");
        } else if (num < 0) {
            System.out.println("The number is Negative.");
        } else {
            System.out.println("The number is Zero.");
        }
        sc.close();
    }
}
```

switch স্টেটমেন্ট

একাধিক শর্ত থাকলে switch স্টেটমেন্ট ব্যবহার করা হয়।

সপ্তাহের দিনের নাম নির্ণয় করা

```java
import java.util.Scanner;

public class DayCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number (1-7): ");
        int day = sc.nextInt();

        switch (day) {
            case 1: System.out.println("Saturday"); break;
            case 2: System.out.println("Sunday"); break;
            case 3: System.out.println("Monday"); break;
            case 4: System.out.println("Tuesday"); break;
            case 5: System.out.println("Wednesday"); break;
            case 6: System.out.println("Thursday"); break;
            case 7: System.out.println("Friday"); break;
```

```
            default: System.out.println("Invalid input! Enter between 1-7.");
        }
        sc.close();
    }
}
```

Lab Task

Task 1: Write a Java program that takes a student's exam marks as input and determines the grade.

Task 2: Write a Java program that asks the user for two numbers and an operator (`+`, `-`, `*`, `/`), then performs the selected operation and displays the result.

Lab 3 : Demonstration of Loops in Java (for, while, do-while) & Array Implementation

লুপ (Loop) কী?

লুপ হলো একটি নির্দিষ্ট কাজ বারবার করার জন্য ব্যবহৃত প্রোগ্রামিং স্ট্রাকচার। যখন আমাদের একই কোড বারবার চালানোর প্রয়োজন হয়, তখন লুপ ব্যবহার করি।

লুপের প্রকারভেদ

Java-তে প্রধানত **তিন ধরনের লুপ** রয়েছে:

1. **for loop** → নির্দিষ্ট সংখ্যক বার চলার জন্য
2. **while loop** → শর্ত সত্য থাকলে চলতে থাকবে
3. **do-while loop** → অন্তত একবার চলবেই, তারপর শর্ত চেক করবে

for লুপ

এই লুপ সাধারণত তখন ব্যবহার করা হয় যখন **কতবার চলতে হবে তা জানা থাকে**।

Example: 1 to 10 print using for loop

```java
public class ForLoopExample {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}
```

while লুপ

যখন **কতবার লুপ চলবে তা নিশ্চিত না**, তখন while লুপ ব্যবহার করা হয়।

Example: Print numbers from 1 to 5 using while loop

```java
public class WhileLoopExample {
    public static void main(String[] args) {
        int i = 1;
        while (i <= 5) {
            System.out.println(i);
            i++;
        }
    }
}
```

do-while লুপ

do-while লুপ **অন্তত একবার চলবেই**, তারপর শর্ত চেক করবে।

Example: Print numbers from 1 to 5 using do-while loop

```java
public class DoWhileLoopExample {
    public static void main(String[] args) {
        int i = 1;
        do {
            System.out.println(i);
            i++;
        } while (i ⩽ 5);
    }
}
```

Array কী?

একটি **array হলো একাধিক একই ধরণের ডাটা সংরক্ষণ করার জন্য ব্যবহৃত data structure**।
এটি **একটি ধারাবাহিক মেমোরি লোকেশন** ব্যবহার করে যেখানে একই টাইপের ভ্যালু রাখা যায়।

Example: Store and Print an Array

```java
public class ArrayExample {
    public static void main(String[] args) {
        // Static Integer Array using new[]
        int[] numbers = new int[5];

        numbers[0] = 1;
        numbers[1] = 2;
        numbers[2] = 3;
        numbers[3] = 4;
        numbers[4] = 7;

        // Static String Array
        String[] names = {"Tanim", "Rakib", "Shakib"};

        System.out.println("Names in the array:");
        for (int i = 0; i < names.length; i++) {
            System.out.println(names[i]);
        }

        System.out.println("\nStatic Array elements:");
        for (int i = 0; i < numbers.length; i++) {
            System.out.println(numbers[i]);
        }
    }
}
```

Lab Tasks :

Task 1: Print Even Numbers Using while , do-while Loops and Array

Write a Java program that takes **starting and ending number** from user then prints all **even numbers** in that range using array.

Task 2: Store and Print Prime Numbers in an Array

Write a Java program that takes a **starting and ending number** from the user and stores **only the prime numbers in an array** and prints them.

Lab 4: Understanding of Class and Object (main() within a Single Class, Constructor Method)

ক্লাস (Class) ও অবজেক্ট (Object) কী?

একটি **ক্লাস** হলো একটি **ব্লুপ্রিন্ট**, যা থেকে অবজেক্ট তৈরি হয়।
একটি **অবজেক্ট** হলো ক্লাসের বাস্তব রূপ, যা ডাটা ধারণ করে এবং মেথড ব্যবহার করতে পারে।

Example:

```
class Student {
    String name;
    int id;
}
```

এখানে Student ক্লাসে **name** এবং **id** নামে দুটি ভ্যারিয়েবল আছে।

```
Student s1 = new Student(); // অবজেক্ট তৈরি
s1.name = "Tanim";
s1.id = 101;
```

এভাবে অবজেক্ট তৈরি করে আমরা ক্লাসের ভ্যারিয়েবল সেট করতে পারি।

কনস্ট্রাক্টর (Constructor) কী?

কনস্ট্রাক্টর হলো **একটি বিশেষ মেথড**, যা **অবজেক্ট তৈরি হওয়ার সময় অটোমেটিক কল হয়** এবং ভ্যারিয়েবল ইনিশিয়ালাইজ করতে ব্যবহৃত হয়।

Example:

```
class Student {
    String name;
    int id;

    // Constructor Method
    Student(String n, int i) {
        name = n;
        id = i;
    }

    void display() {
        System.out.println("Name: " + name);
        System.out.println("ID: " + id);
    }

    public static void main(String[] args) {
        Student s1 = new Student("Tanim", 101);
        s1.display();
```

```
    }
}
```

🔹 **গুরুত্বপূর্ণ পয়েন্ট:**

✔️ কনস্ট্রাক্টরের নাম ক্লাসের নামে হতে হবে।
✔️ এটি কোনো রিটার্ন টাইপ (void, int) রাখে না।
✔️ অবজেক্ট তৈরি করার সময় এটি অটোমেটিক কল হয়।

গেটার (Getter) এবং সেটার (Setter) কী?

গেটার এবং সেটার মেথড ব্যবহার করে আমরা প্রাইভেট ভ্যারিয়েবলের মান সেট বা রিট্রিভ করতে পারি।

Example:

```java
class StudentTemplate {
    private String name;
    private int id;

    StudentTemplate(String n, int i) {
        this.name = n;
        this.id = i;
    }

    // Setter Method
    void setInfo(String n, int i) {
        this.name = n;
        this.id = i;
    }

    // Getter Method
    void getInfo() {
        System.out.println("Name: " + name);
        System.out.println("ID: " + id);
    }

    public static void main(String[] args) {
        StudentTemplate s1 = new StudentTemplate("Tanim", 100);
        s1.getInfo();
        s1.setInfo("Rakib", 110);
        s1.getInfo();
    }
}
```

Lab Tasks:

Task 1: Car ক্লাস তৈরি এবং কনস্ট্রাক্টর ব্যবহার করে ডাটা সেট করা

- **Car** নামে একটি ক্লাস তৈরি করো, যেখানে তিনটি ভ্যারিয়েবল থাকবে: **brand, model, price**
- একটি **কনস্ট্রাক্টর** থাকবে, যা ডাটা সেট করবে
- একটি **displayCar()** মেথড থাকবে যা গাড়ির তথ্য দেখাবে

Task 2: IDCard ক্লাস তৈরি এবং কনস্ট্রাক্টর ব্যবহার করে তথ্য সেট করা

- **IDCard** নামে একটি ক্লাস তৈরি করো, যেখানে থাকবে: **name, id, department, institution**
- **একটি কনস্ট্রাক্টর থাকবে যা তথ্য ইনিশিয়ালাইজ করবে**
- **showID()** নামে একটি মেথড থাকবে যা আইডি কার্ডের তথ্য দেখাবে

Lab 5: Understanding Class and Object (main() Outside the Class)

ক্লাস (Class) ও অবজেক্ট (Object) – main() Outside the Class

আগের ল্যাবে **main() মেথড ক্লাসের ভেতরেই ছিল**। কিন্তু বড় প্রজেক্টে **main() আলাদা রাখা হয়**, যেন কোড সহজে মেইনটেইন করা যায়।

Example:

```java
class Student {
    String name;
    int id;

    // Constructor
    Student(String n, int i) {
        name = n;
        id = i;
    }

    void display() {
        System.out.println("Name: " + name);
        System.out.println("ID: " + id);
    }
}

// Main class
public class LabFive {
    public static void main(String[] args) {
        Student s1 = new Student("Tanim", 101);
        s1.display();
    }
}
```


এখানে Main ক্লাসের ভেতর main() আছে, কিন্তু Student ক্লাস আলাদা রাখা হয়েছে।

স্ট্যাটিক (Static) কী?

যখন কোনো ভ্যারিয়েবল বা মেথডকে **স্ট্যাটিক (static) করা হয়**, তখন এটি **পুরো ক্লাসের জন্য একটাই থাকে এবং অবজেক্ট তৈরি না করেও এক্সেস করা যায়।**

Example:

```
class University {
    static String universityName = "IIUC"; // Static variable

    static void showUniversity() {
        System.out.println("University: " + universityName);
    }
}

public class Main {
    public static void main(String[] args) {
        University.showUniversity(); // No need to create an object
    }
}
```

এখানে showUniversity() অবজেক্ট ছাড়া ডিরেক্ট এক্সেস করা হয়েছে কারণ এটি static।

this কী?

this কিওয়ার্ড ব্যবহার করে আমরা বর্তমান ক্লাসের ভ্যারিয়েবল এবং মেথড এক্সেস করতে পারি।

Example:

```
class Student {
    String name;
    int id;

    Student(String name, int id) {
        this.name = name; // this keyword used to differentiate
instance variable
        this.id = id;
    }

    void display() {
        System.out.println("Name: " + this.name);
        System.out.println("ID: " + this.id);
    }
}
```

এখানে this.name এবং this.id ব্যবহার করা হয়েছে কারণ কনস্ট্রাক্টরের প্যারামিটার এবং ক্লাস ভ্যারিয়েবলের নাম একই।

Lab Tasks:

Task 1: Create a BankAccount Class with Static Members

- **Create a class BankAccount** with the following attributes:
    - accountNumber
    - accountHolderName
    - balance
    - **Static variable:** bankName = "Sonali Bank"
- **Create a constructor** to initialize account details
- **Create a static method** showBankName() to display the bank name
- **Write a Main class** where you create two accounts and display their information

Task 2: Create a LibraryBook Class Using this Keyword

- **Create a class LibraryBook** with the following attributes:
    - title
    - author
    - bookID
- **Use this keyword** in the constructor to initialize values
- **Create a method displayBookInfo()** to show the book details
- **Write a Main class** where you create books and display their information

Lab 6: Demonstration of Inheritance (Multilevel & Hierarchical)

ইনহেরিটেন্স (Inheritance) কী?

Inheritance (উত্তরাধিকার) একটি Object Oriented Programming-এর গুরুত্বপূর্ণ বৈশিষ্ট্য, যার মাধ্যমে একটি ক্লাস (child class) অন্য একটি ক্লাসের (parent class) বৈশিষ্ট্য (properties) এবং মেথড (methods) ব্যবহার করতে পারে। এটি কোড রি-ইউজ কমায় এবং ওবজেক্ট-ওরিয়েন্টেড প্রোগ্রামিংকে আরও শক্তিশালী করে।

ইনহেরিটেন্সের প্রকারভেদ

- **Single Inheritance:** একটি ক্লাস অন্য একটি ক্লাস থেকে ইনহেরিট করে।
- **Multilevel Inheritance:** একটি ক্লাস অন্য একটি ক্লাস থেকে ইনহেরিট করে, এবং সেটি আবার আরেকটি ক্লাস ইনহেরিট করে।
- **Hierarchical Inheritance:** একটি Parent ক্লাস থেকে একাধিক চাইল্ড ক্লাস ইনহেরিট করে।

Multilevel Inheritance

Example: Vehicle → Car → ElectricCar

```java
// Parent class
class Vehicle {
    String brand = "Tesla";

    void honk() {
        System.out.println("Beep! Beep!");
    }
}

// Child class inheriting from Vehicle
class Car extends Vehicle {
    String model = "V3";

    void displayCar() {
        System.out.println("Brand: " + brand);
        System.out.println("Model: " +  model );
    }
}

// Grandchild class inheriting from Car
```

```java
class ElectricCar extends Car {
    int speed = 160;

    void runElectricCar() {
        System.out.println("Speed: " + speed + " KM/h");
    }
}

public class TestCode {
    public static void main(String[] args) {
        ElectricCar tesla = new ElectricCar();

        // Calling methods from parent and child classes
        tesla.honk();
        tesla.displayCar();
        tesla.runElectricCar();
    }
}
```

ব্যাখ্যা:

- Vehicle ক্লাস থেকে Car ইনহেরিট করেছে।
- Car ক্লাস থেকে ElectricCar ইনহেরিট করেছে।
- ElectricCar ক্লাস honk(), displayCar() ও নিজের runElectricCar() মেথড ব্যবহার করতে পারছে।

Hierarchical Inheritance

Example: Parent (Animal) → Multiple Children (Dog, Cat)

```java
// Parent class
class Animal {
    String name;
    void eat() {
        System.out.println(name + " is eating.");
    }
}

// Child class 1
```

```java
class Dog extends Animal {
    public Dog(String n) {
        this.name = n;
    }

    void bark() {
        System.out.println(name + " : Woof! Woof!");
    }
}

// Child class 2
class Cat extends Animal {
    public Cat(String n) {
        this.name = n;
    }

    void meow() {
        System.out.println(name + " : Meow! Meow!");
    }
}

public class TestCode {
    public static void main(String[] args) {
        Dog dog = new Dog("Husky");
        Cat cat = new Cat("Tom");

        dog.eat(); // Calling parent class method
        dog.bark();

        cat.eat(); // Calling parent class method
        cat.meow();
    }
}
```


ব্যাখ্যা:

- Animal হলো Parent ক্লাস।
- Dog এবং Cat দুটি চাইল্ড ক্লাস যা Animal থেকে ইনহেরিট করেছে।

🔹 Dog ক্লাস bark() ব্যবহার করতে পারে, এবং Cat ক্লাস meow() ব্যবহার করতে পারে, কিন্তু দুজনেই eat() ব্যবহার করতে পারে।

Lab Tasks

Task 1: Grandfather → Father → Son (Multilevel Inheritance)

- Create a class Grandfather with a method company().
- Create a class Father that extends Grandfather and adds a method car().
- Create a class Son that extends Father and add a method.
- Demonstrate how Son can access all properties.

Task 2: Person → Doctor/Teacher/Engineer (Hierarchical Inheritance)

- Create a superclass Person with method displayInfo().
- Create subclasses Doctor, Teacher, and Engineer extending Person.
- Each subclass will have its own method to describe their work.

Lab 7: Demonstration of Polymorphism Using Method Overloading

1️⃣ Polymorphism কী?

Polymorphism (পলিমরফিজম) মানে হচ্ছে **একই নামের মেথড, ভিন্ন ভিন্ন কাজ** করা।
Java-তে Polymorphism দুইভাবে হয়:

- **Compile-time Polymorphism** → Method Overloading এর মাধ্যমে
- **Runtime Polymorphism** → Method Overriding এর মাধ্যমে

আজকে আমরা Method Overloading করবো।

2️⃣ Method Overloading কী?

Method Overloading তখন হয়, যখন একই নামের একাধিক মেথড থাকে কিন্তু:

- তাদের **প্যারামিটার সংখ্যা** বা
- **প্যারামিটার টাইপ** আলাদা হয়।

Example:

```
public class MathOperations {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
}
```

এখানে add() মেথড তিনবার আছে, কিন্তু প্যারামিটার আলাদা, তাই Method Overloading হয়েছে।

Lab Tasks:

Task 1: Calculator ক্লাস তৈরি করো যেখানে Addition এর জন্য Method Overloading থাকবে

- দুটি পূর্ণসংখ্যা যোগ করবে।
- দুটি ভগ্নাংশ যোগ করবে।
- তিনটি পূর্ণসংখ্যা যোগ করবে।

Task 2: Shape ক্লাস তৈরি করো যেখানে Area বের করার জন্য Method Overloading থাকবে

- একটি মেথড Rectangle-এর জন্য (length, width নিবে)
- আরেকটি মেথড Circle-এর জন্য (radius নিবে)

Lab 8: Demonstration of Polymorphism Using Method Overriding

1️⃣ Method Overriding কী?

Method Overriding হচ্ছে এমন একটি কনসেপ্ট যেখানে:

- Subclass (child class) তার Superclass (parent class)-এর method **redefine বা override** করে।

এইটার মাধ্যমে **Runtime Polymorphism** ঘটে।

অর্থাৎ, একই মেথড নাম, কিন্তু child class নিজের মতো করে কাজ করবে।

2️⃣ Overriding-এর নিয়ম:

- Inheritance থাকতে হবে (i.e., parent → child class)
- Method-এর নাম, রিটার্ন টাইপ, প্যারামিটার একই থাকতে হবে
- Child class সেই মেথড নিজের মতো করে লিখবে

Example:

```
class Animal {
    void sound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}
```


Override অ্যানোটেশন না দিলেও **`method overriding`** কাজ করবে।

তবে @Override লেখার মূল সুবিধা হলো:

কেন @Override ব্যবহার করা ভালো?

1. **কম্পাইলার নিশ্চিত করে** যে তুমি parent class-এর method টাকে override করছো (ঠিকমতো spelling, parameters মেলানো হয়েছে কিনা)।
2. method এর নাম বা signature mismatch থাকে, তাহলে কম্পাইলার error দিবে।

3. কোড পড়ার সময় **বুঝা সহজ হয়** যে এটা overriding করা হচ্ছে।

না দিলে কী হয়?

- কোড চলবে, যদি ঠিকমতো override করো।
- কিন্তু যদি ভুল হয়, কম্পাইলার detect করতে পারবে না। ফলে প্রোগ্রাম ভুলভাবে কাজ করতে পারে।

Lab Tasks:

Task 1: Animal ক্লাস তৈরি করো এবং sound() method Override করো

- Animal → sound(): prints "Some generic sound"
- Dog → sound(): prints "Dog barks"
- Cat → sound(): prints "Cat meows"

Task 2: Shape ক্লাস তৈরি করো এবং draw() method Override করো

- Shape → draw(): prints "Drawing shape"
- Circle → draw(): prints "Drawing Circle"
- Rectangle → draw(): prints "Drawing Rectangle"

Lab 9: Demonstration of Encapsulation in Java

Encapsulation কী?

Encapsulation হচ্ছে এমন একটি OOP technique যেখানে ডেটা এবং সেই ডেটা ব্যবহারের মেথডগুলো একটি সিঙ্গেল ইউনিটে (class) রাখা হয়। এর মাধ্যমে ডেটা সরাসরি access না করে মেথডের মাধ্যমে access করা হয়। class এর ভেতরের variable গুলোকে private করে দেই এবং বাইরে থেকে access করার জন্য getter এবং setter method ব্যবহার করি।

কেন Encapsulation গুরুত্বপূর্ণ?

- Data hiding: Direct access না থাকায় sensitive data নিরাপদ থাকে
- Controlled Access: কে কিভাবে access করবে সেটা আমরা ঠিক করতে পারি
- Maintenance: Code manage এবং modify করা সহজ হয়
- Flexibility: Setter দিয়ে validation যোগ করা যায়

কিভাবে Java-তে Encapsulation করা হয়?

1. সব instance variable private করতে হবে
2. Data read করার জন্য getter method
3. Data modify/set করার জন্য setter method

Example:

```
class Student {
    private String name;
    private String id;
    private double cgpa;

    // Setter Methods
    public void setName(String n) {
        name = n;
    }

    public void setId(String i) {
        id = i;
    }

    public void setCgpa(double c) {
        cgpa = c;
    }
```

```java
    // Getter Methods
    public String getName() {
        return name;
    }

    public String getId() {
        return id;
    }

    public double getCgpa() {
        return cgpa;
    }
}
public class TestCode {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.setName("Tanim");
        s1.setId("E221013");
        s1.setCgpa(0.00);

        System.out.println("Student Info:");
        System.out.println("Name: " + s1.getName());
        System.out.println("ID: " + s1.getId());
        System.out.println("CGPA: " + s1.getCgpa());
    }
}
```


Lab Tasks

- **Task 1: Student Class with Encapsulation**

Create a Student class with private fields: name, id, and cgpa. Use setter methods to set values and getter methods to display them.

- **Task 2: BankAccount Class with Encapsulation**

Create a BankAccount class with private fields: accountHolder, accountNumber, and balance. Use methods to set values and show account info securely.

Lab 10: Demonstration of Abstraction & Interfaces

Abstraction কী?

Abstraction মানে হলো – জটিল ব্যাকএন্ড প্রসেসিং বা হিসাব-নিকাশ ব্যবহারকারীর কাছ থেকে লুকিয়ে শুধুমাত্র তার প্রয়োজনীয় অংশ (সার্ভিস বা ফিচার) দেখানো।

সহজ ভাষায়: "Abstraction হলো এমন একটি পদ্ধতি যেখানে প্রোগ্রামিংয়ের পেছনের জটিল কাজগুলো (যেমন হিসাব, validation, ডাটাবেজ অপারেশন) ব্যবহারকারীর কাছ থেকে লুকিয়ে রাখা হয়, এবং শুধু দরকারি অংশটা (যেমন বোতাম, তথ্য, ফর্ম) দেখানো হয়।"

ATM Example:

- ইউজার যখন ATM-এ কার্ড ঢুকিয়ে টাকা তোলেন, তখন পেছনে অনেক কাজ হয় যেমন authentication, ব্যালেন্স চেক, ট্রান্সাকশন ইত্যাদি।
- কিন্তু ইউজার এগুলোর কিছুই দেখেন না, শুধু নিজের দরকারি জিনিস – টাকা তোলা, ব্যালেন্স দেখা – এগুলোই দেখতে পান।

এইভাবে abstraction প্রোগ্রামে simplicity এবং security এনে দেয়।

Abstract Class কী?

- Abstract class এমন একটি class যা সম্পূর্ণ বা অসম্পূর্ণ (abstract) method থাকতে পারে।
- এটি থেকে object তৈরি করা যায় না, কিন্তু subclass তৈরি করে সেটাকে ব্যবহার করা যায়।

Example:

```
abstract class Animal {
    abstract void makeSound();

    void eat() {
        System.out.println("This animal eats food");
    }
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Dog barks");
    }
}
```

```
public class TestCode {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.makeSound();
        d.eat();
    }
}
```

Interface কী?

- Interface হলো একধরনের pure abstraction যেখানে শুধুমাত্র abstract method থাকে (Java 8 এর আগে)।
- Interface-এ কোনো constructor বা instance variable রাখা যায় না।
- এটি implement করে class গুলো নিজের মত করে method define করে।

Example:

```
// Define the Animal interface
interface Animal {
    void sound();  // Abstract method to be implemented by any class
that implements Animal
}

// Implementing the Animal interface with the Cat class
class Cat implements Animal {
    public void sound() {
        System.out.println("Cat meows");  // Implementation of the
sound method
    }
}

// Main class to test the interface and implementation
public class TestCode {
    public static void main(String[] args) {
        // Create an instance of Cat
        Animal myCat = new Cat();

        // Call the sound method on the Cat object
```

```
        myCat.sound();  // Output: Cat meows
    }
}
```

Example: ATM System দিয়ে Abstraction

Abstract Class দিয়ে:

```
// Abstract class ATM
abstract class ATM {
    // Abstract method for user authentication
    abstract void authenticateUser();

    // Abstract method for performing a transaction
    abstract void performTransaction();
}

// CityBankATM class extending the ATM class and providing concrete
implementations
class CityBankATM extends ATM {
    // Implement the authenticateUser method
    void authenticateUser() {
        System.out.println("Authenticating user with CityBank
ATM...");
    }

    // Implement the performTransaction method
    void performTransaction() {
        System.out.println("Performing transaction in CityBank
ATM...");
    }
}

// Main class to test the functionality
public class TestCode {
    public static void main(String[] args) {
        // Create an instance of CityBankATM
        ATM cityBankATM = new CityBankATM();
```

```
        // Call the methods
        cityBankATM.authenticateUser();   // Output: Authenticating
user with CityBank ATM...
        cityBankATM.performTransaction(); // Output: Performing
transaction in CityBank ATM...
    }
}
```

Interface দিয়ে:

```
// Define the ATMService interface
interface ATMService {
    void withdraw();       // Method to withdraw money
    void deposit();        // Method to deposit money
    void checkBalance();   // Method to check the balance
}

// Implementing the ATMService interface with the DBBL class
class DBBL implements ATMService {
    // Implementation of withdraw method
    public void withdraw() {
        System.out.println("Withdraw money from DBBL");
    }

    // Implementation of deposit method
    public void deposit() {
        System.out.println("Deposit money to DBBL");
    }

    // Implementation of checkBalance method
    public void checkBalance() {
        System.out.println("Check balance in DBBL");
    }
}
```

```java
// Main class to test the ATMService interface and its implementation
public class TestCode {
    public static void main(String[] args) {
        // Create an instance of DBBL
        ATMService dbblATM = new DBBL();

        // Call the methods
        dbblATM.withdraw();         // Output: Withdraw money from
DBBL
        dbblATM.deposit();          // Output: Deposit money to DBBL
        dbblATM.checkBalance();     // Output: Check balance in DBBL
    }
}
```


Lab Tasks:

Task 1: Create an abstract class Shape with abstract method area() and implement it in subclasses Circle and Rectangle.

Task 2: Create an interface ATMService and implement it in a class DBBL with methods: withdraw(), deposit(), and checkBalance().