LAB FILE

JAVA PROGRAMMING

(COURSE CODE: IT201)

SUBMITTED BY:

Name: Joydeep Nandi

Enrollment No. A023119823001

Class& Section: B. Tech AI X

SUBMITTED TO:

Mr.Abhishek Dixit

Assistant Professor

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH, NOIDA SESSION:

ODD SEM. (2024-25)

INDEX

EXPERIMENT 1

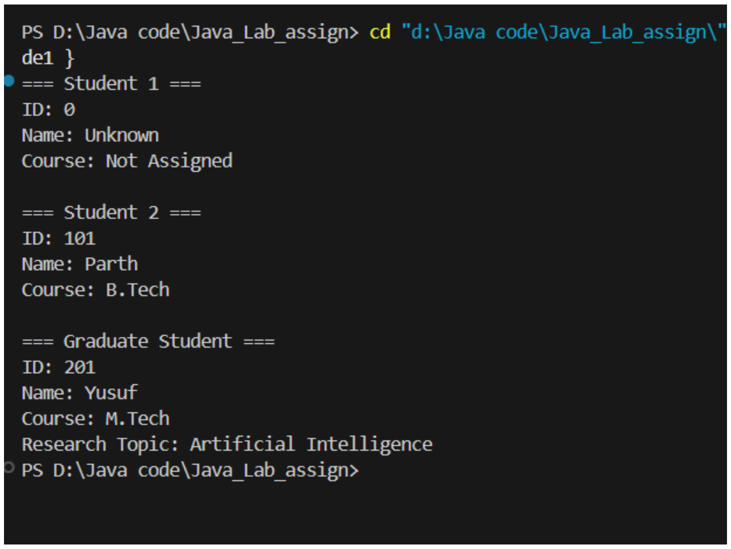
1. **A university maintains a record of students.**
    a. **Create a class Student with id, name, and course.**
    b. **Demonstrate constructor overloading and method overriding with a subclass GraduateStudent.**

CODE:

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

   Student() {
      id = 0;
      name = "Unknown";
      course = "Not Assigned";
   }

   Student(int id, String name, String course) {
      this.id = id;
      this.name = name;
      this.course = course;
   }

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

class GraduateStudent extends Student {
   String researchTopic;

   GraduateStudent(int id, String name, String course, String researchTopic) {
      super(id, name, course);
      this.researchTopic = researchTopic;
   }

   @Override
   void displayInfo() {
      super.displayInfo();
```

```
        System.out.println("Research Topic: " + researchTopic);
    }
}

public class Code1 {
    public static void main(String[] args) {
        Student s1 = new Student();
        System.out.println("=== Student 1 ===");
        s1.displayInfo();

        Student s2 = new Student(101, "Riya", "B.Tech");
        System.out.println("\n=== Student 2 ===");
        s2.displayInfo();

        GraduateStudent g1 = new GraduateStudent(201, "Aman", "M.Tech", "Artificial
Intelligence");
        System.out.println("\n=== Graduate Student ===");
        g1.displayInfo();
    }
}
```

OUTPUT :



```
PS D:\Java code\Java_Lab_assign> cd "d:\Java code\Java_Lab_assign\"
de1 }
=== Student 1 ===
ID: 0
Name: Unknown
Course: Not Assigned

=== Student 2 ===
ID: 101
Name: Parth
Course: B.Tech

=== Graduate Student ===
ID: 201
Name: Yusuf
Course: M.Tech
Research Topic: Artificial Intelligence
PS D:\Java code\Java_Lab_assign>
```

EXPERIMENT 2

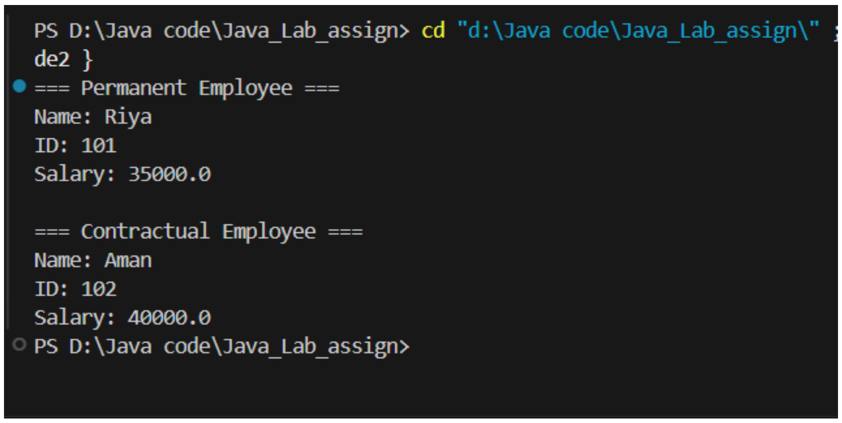
2. **You are asked to design a system where an employee can be permanent or contractual.**
    a. **Use abstract classes and runtime polymorphism to implement salary calculation differently for each employee type.**

CODE:

```
abstract class Employee {
   String name;
   int id;

   Employee(String name, int id) {
      this.name = name;
      this.id = id;
   }

   abstract double calculateSalary();

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

class PermanentEmployee extends Employee {
   double basicPay, bonus;

   PermanentEmployee(String name, int id, double basicPay, double bonus) {
      super(name, id);
      this.basicPay = basicPay;
      this.bonus = bonus;
   }

   double calculateSalary() {
      return basicPay + bonus;
   }
}

class ContractualEmployee extends Employee {
   double hourlyRate;
   int hoursWorked;

   ContractualEmployee(String name, int id, double hourlyRate, int hoursWorked) {
      super(name, id);
      this.hourlyRate = hourlyRate;
```

```
        this.hoursWorked = hoursWorked;
    }

    double calculateSalary() {
        return hourlyRate * hoursWorked;
    }
}

public class Code2 {
    public static void main(String[] args) {
        Employee e1 = new PermanentEmployee("Riya", 101, 30000, 5000);
        Employee e2 = new ContractualEmployee("Aman", 102, 500, 80);

        System.out.println("=== Permanent Employee ===");
        e1.displayInfo();
        System.out.println("Salary: " + e1.calculateSalary());

        System.out.println("\n=== Contractual Employee ===");
        e2.displayInfo();
        System.out.println("Salary: " + e2.calculateSalary());
    }
}
```

OUTPUT :



```
PS D:\Java code\Java_Lab_assign> cd "d:\Java code\Java_Lab_assign\"
de2 }
=== Permanent Employee ===
Name: Riya
ID: 101
Salary: 35000.0

=== Contractual Employee ===
Name: Aman
ID: 102
Salary: 40000.0
PS D:\Java code\Java_Lab_assign>
```

EXPERIMENT 3

3. A banking system needs to group related classes in a package banking.

   a. Create a package with classes Account and Loan.

   b. Import it in another program and perform operations like deposit and loan sanction.

CODE:

File 1 — banking/Account.java

```
package banking;

public class Account {
   private String name;
   private double balance;

   public Account(String name, double balance) {
      this.name = name;
      this.balance = balance;
   }

   public void deposit(double amount) {
      balance += amount;
      System.out.println("Deposited: " + amount);
   }

   public void showBalance() {
      System.out.println("Account Holder: " + name);
      System.out.println("Current Balance: " + balance);
   }
}
```

File 2 — banking/Loan.java

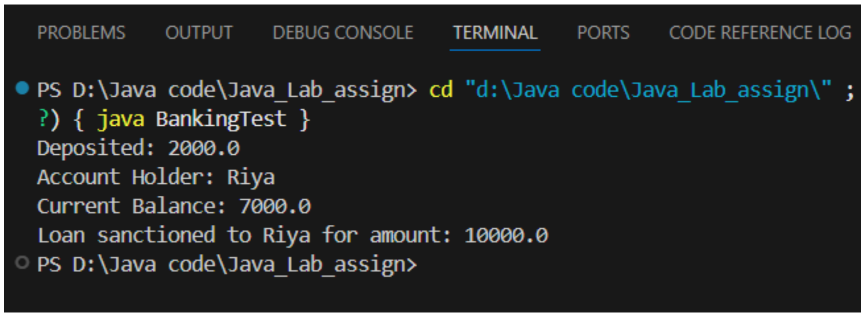
```
package banking;

public class Loan {
   private String borrower;
   private double amount;

   public Loan(String borrower, double amount) {
      this.borrower = borrower;
      this.amount = amount;
   }
```

```
    public void sanctionLoan() {
        System.out.println("Loan sanctioned to " + borrower + " for amount: " + amount);
    }
}
```

File 3 — BankingTest.java

```
import banking.Account;
import banking.Loan;

public class BankingTest {
    public static void main(String[] args) {
        Account acc = new Account("Riya", 5000);
        acc.deposit(2000);
        acc.showBalance();

        Loan loan = new Loan("Riya", 10000);
        loan.sanctionLoan();
    }
}
```

OUTPUT:



```
PS D:\Java code\Java_Lab_assign> cd "d:\Java code\Java_Lab_assign\" ;
?) { java BankingTest }
Deposited: 2000.0
Account Holder: Riya
Current Balance: 7000.0
Loan sanctioned to Riya for amount: 10000.0
PS D:\Java code\Java_Lab_assign>
```

EXPERIMENT 4

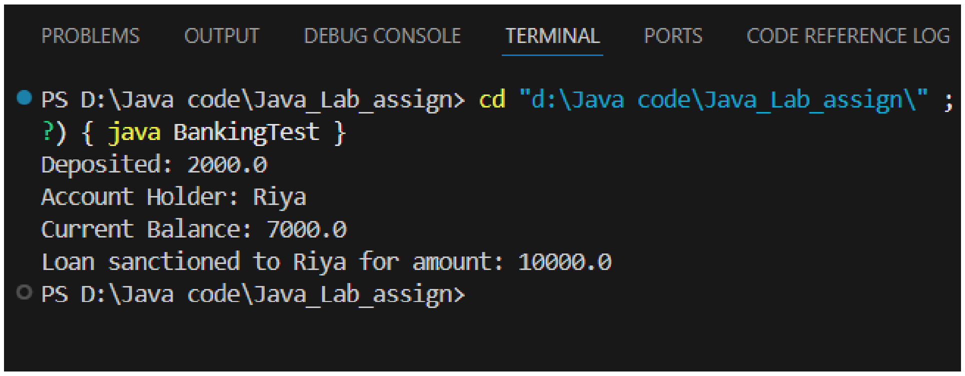
4. **An online shopping system throws an exception when a customer tries to purchase more items than available in stock.**
    a. **Write a program to define a custom exception OutOfStockException and handle it gracefully.**

CODE:

```
class OutOfStockException extends Exception {
   public OutOfStockException(String message) {
      super(message);
   }
}

public class Code4 {
   static int availableStock = 10;

   public static void purchaseItem(int quantity) throws OutOfStockException {
      if (quantity > availableStock) {
         throw new OutOfStockException("Sorry! Only " + availableStock + " items are available
in stock.");
      } else {
         availableStock -= quantity;
         System.out.println("Purchase successful! Remaining stock: " + availableStock);
      }
   }

   public static void main(String[] args) {
      try {
         System.out.println("Customer is trying to buy 12 items...");
         purchaseItem(12);
      }
      catch (OutOfStockException e) {
         System.out.println("Exception caught: " + e.getMessage());
      }

      System.out.println("\nCustomer is trying to buy 5 items...");
      try {
         purchaseItem(5);
      }
      catch (OutOfStockException e) {
         System.out.println("Exception caught: " + e.getMessage());
      }
   }
```

}

OUTPUT:



```
PS D:\Java code\Java_Lab_assign> cd "d:\Java code\Java_Lab_assign\" ;
?) { java BankingTest }
Deposited: 2000.0
Account Holder: Riya
Current Balance: 7000.0
Loan sanctioned to Riya for amount: 10000.0
PS D:\Java code\Java_Lab_assign>
```

EXPERIMENT 5

5. A file contains a list of product names.

a. **Write a program to read product names using FileReader and print them in uppercase.**

CODE:

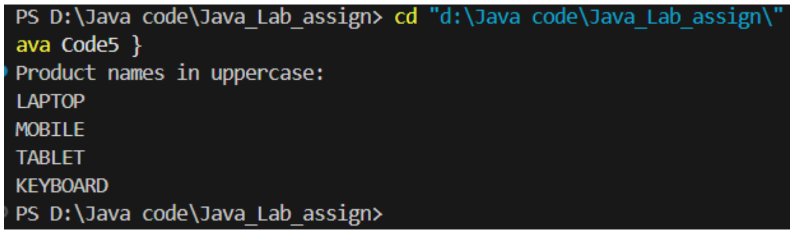
Read Product Names and Print Uppercase

```
import java.io.*;
public class Code5 {
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("products.txt");
            BufferedReader br = new BufferedReader(fr);
            String line;
            System.out.println("Product names in uppercase:");
            while ((line = br.readLine()) != null) {
                System.out.println(line.toUpperCase());
            }
            br.close();
            fr.close();
        }
        catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

products.txt

```
laptop
mobile
tablet
keyboard
```

OUTPUT:



```
PS D:\Java code\Java_Lab_assign> cd "d:\Java code\Java_Lab_assign\"
ava Code5 }
Product names in uppercase:
LAPTOP
MOBILE
TABLET
KEYBOARD
PS D:\Java code\Java_Lab_assign>
```

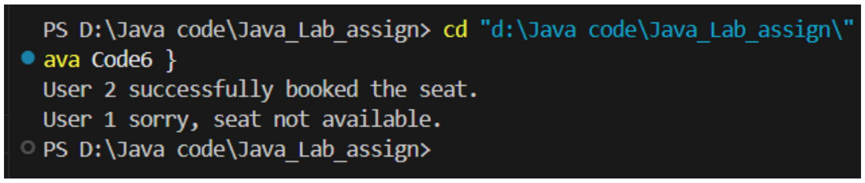
EXPERIMENT 6

6. **In a ticket booking system, two users try to book the last seat simultaneously.**
    a. **Write a program using threads and synchronization to ensure only one booking is successful.**

CODE:

```
class TicketBooking {
   int availableSeats = 1;

   synchronized void bookTicket(String user) {
      if (availableSeats > 0) {
         System.out.println(user + " successfully booked the seat.");
         availableSeats--;
      } else {
         System.out.println(user + " sorry, seat not available.");
      }
   }
}

class UserThread extends Thread {
   TicketBooking booking;
   String user;

   UserThread(TicketBooking booking, String user) {
      this.booking = booking;
      this.user = user;
   }

   public void run() {
      booking.bookTicket(user);
   }
}

public class Code6 {
   public static void main(String[] args) {
      TicketBooking booking = new TicketBooking();
      UserThread u1 = new UserThread(booking, "User 1");
      UserThread u2 = new UserThread(booking, "User 2");

      u1.start();
      u2.start();
   }
}
```

OUTPUT:



```
  PS D:\Java code\Java_Lab_assign> cd "d:\Java code\Java_Lab_assign\"
● ava Code6 }
  User 2 successfully booked the seat.
  User 1 sorry, seat not available.
○ PS D:\Java code\Java_Lab_assign>
```

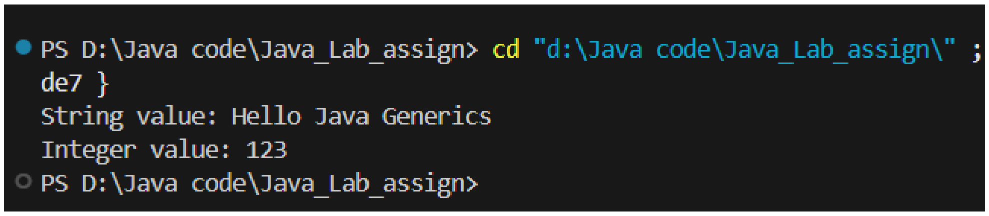
EXPERIMENT 7

7. **Create a generic class Box<T> that can hold objects of any type.**
    a. **Demonstrate storing and retrieving String and Integer values using the same class.**

CODE:

```
class Box<T> {
    private T value;

    void setValue(T value) {
        this.value = value;
    }

    T getValue() {
        return value;
    }
}

public class Code7 {
    public static void main(String[] args) {
        Box<String> stringBox = new Box<>();
        stringBox.setValue("Hello Java Generics");
        System.out.println("String value: " + stringBox.getValue());

        Box<Integer> intBox = new Box<>();
        intBox.setValue(123);
        System.out.println("Integer value: " + intBox.getValue());
    }
}
```

OUTPUT:



```
PS D:\Java code\Java_Lab_assign> cd "d:\Java code\Java_Lab_assign\" ;
de7 }
String value: Hello Java Generics
Integer value: 123
PS D:\Java code\Java_Lab_assign>
```

EXPERIMENT 8

8. A company maintains employee data.

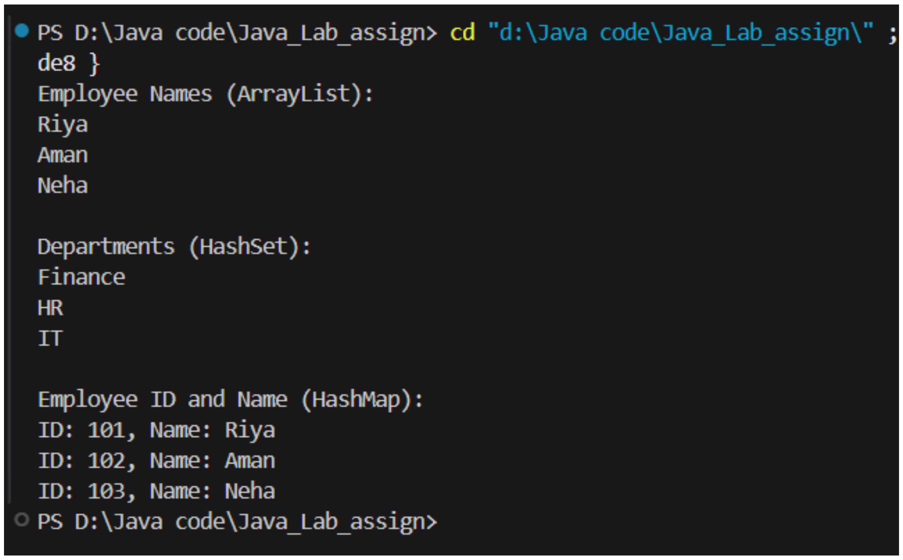
   a. Use an ArrayList to store employee names, a HashSet to store unique departments, and a HashMap to map employee IDs with names.

   b. Display all data.

CODE:

```
import java.util.*;

public class Code8 {
   public static void main(String[] args) {
      ArrayList<String> employeeNames = new ArrayList<>();
      employeeNames.add("Riya");
      employeeNames.add("Aman");
      employeeNames.add("Neha");

      HashSet<String> departments = new HashSet<>();
      departments.add("HR");
      departments.add("Finance");
      departments.add("IT");
      departments.add("IT");

      HashMap<Integer, String> employeeMap = new HashMap<>();
      employeeMap.put(101, "Riya");
      employeeMap.put(102, "Aman");
      employeeMap.put(103, "Neha");

      System.out.println("Employee Names (ArrayList):");
      for (String name : employeeNames)
         System.out.println(name);

      System.out.println("\nDepartments (HashSet):");
      for (String dept : departments)
         System.out.println(dept);

      System.out.println("\nEmployee ID and Name (HashMap):");
      for (Map.Entry<Integer, String> entry : employeeMap.entrySet())
         System.out.println("ID: " + entry.getKey() + ", Name: " + entry.getValue());
   }
}
```

OUTPUT:



```
PS D:\Java code\Java_Lab_assign> cd "d:\Java code\Java_Lab_assign\" ;
de8 }
Employee Names (ArrayList):
Riya
Aman
Neha

Departments (HashSet):
Finance
HR
IT

Employee ID and Name (HashMap):
ID: 101, Name: Riya
ID: 102, Name: Aman
ID: 103, Name: Neha
PS D:\Java code\Java_Lab_assign>
```

EXPERIMENT 9

9. An organization wants to track methods executed in a class.

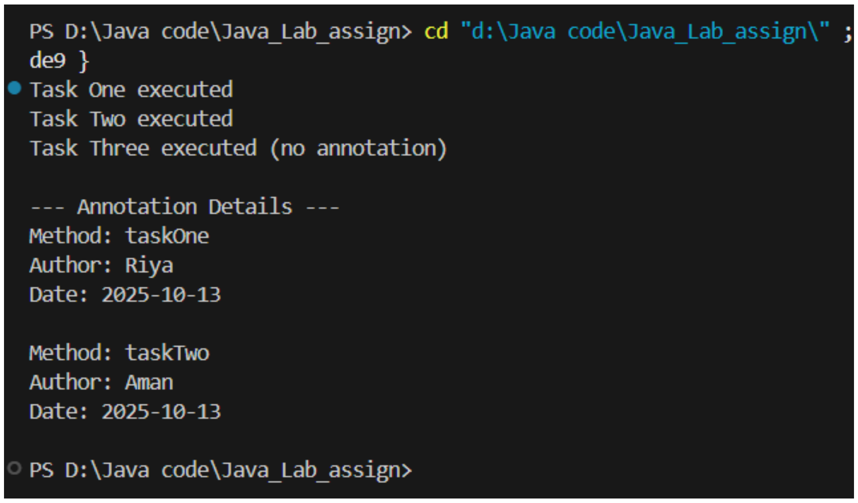
   a. Create a custom annotation @TrackExecution.

   b. Apply it to methods in a Task class and print annotation details at runtime using reflection.

CODE:

```
import java.lang.annotation.*;
import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface TrackExecution {
   String author() default "Unknown";
   String date();
}

class Task {
   @TrackExecution(author = "Riya", date = "2025-10-13")
   void taskOne() {
      System.out.println("Task One executed");
   }

   @TrackExecution(author = "Aman", date = "2025-10-13")
   void taskTwo() {
      System.out.println("Task Two executed");
   }

   void taskThree() {
      System.out.println("Task Three executed (no annotation)");
   }
}

public class Code9 {
   public static void main(String[] args) {
      Task t = new Task();
      t.taskOne();
      t.taskTwo();
      t.taskThree();

      System.out.println("\n--- Annotation Details ---");
      Method[] methods = Task.class.getDeclaredMethods();
      for (Method m : methods) {
         if (m.isAnnotationPresent(TrackExecution.class)) {
                TrackExecution te = m.getAnnotation(TrackExecution.class);
                System.out.println("Method: " + m.getName());
                System.out.println("Author: " + te.author());
                System.out.println("Date: " + te.date());
                System.out.println();
            }
        }
    }

}
```

OUTPUT:



```
PS D:\Java code\Java_Lab_assign> cd "d:\Java code\Java_Lab_assign\" ;
de9 }
Task One executed
Task Two executed
Task Three executed (no annotation)

--- Annotation Details ---
Method: taskOne
Author: Riya
Date: 2025-10-13

Method: taskTwo
Author: Aman
Date: 2025-10-13

PS D:\Java code\Java_Lab_assign>
```

EXPERIMENT 10

10. A Java project requires external libraries like MySQL Connector.

a. Write a **pom.xml snippet** to add the MySQL dependency.

CODE:

pom.xml Snippet — MySQL Connector/J

```
<dependencies>
   <!-- MySQL Connector/J -->
   <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.33</version>
   </dependency>
</dependencies>
```