LAB FILE

JAVA PROGRAMMING
(Course Code: IT201)

AMITY UNIVERSITY
UTTAR PRADESH

SUBMITTED BY:

GAURI CHOPRA

Enrollment No. : A023119823038

Program: B. Tech AI Y

SUBMITTED TO:

Dr. ABHISHEK DIXIT

AMITY UNIVERSITY UTTAR PRADESH, NOIDA

ODD SEM. (2024-25)

Practical: 1

Q1. 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(){
        name= "Undefined";
        course="btech";
        id=0;
    }
    Student(int id,String name,String course){
        this.id=id;
        this.name=name;
        this.course=course;
    }
    void getDetails(){
        System.out.println("\n id:”+id+",name:"+name+",course:"+course); } }
class GraduateStudent extends Student{
String specialisation;
```

```
GraduateStudent(int id,String name,String course, String specialisation){
    super(id,name,course);
    this.specialisation=specialisation;
}
@Override
void getDetails(){
    super.getDetails();
    System.out.println(“specialisation:"+specialisation); }
}
public class Q1 {
public static void main(String[] args){
    Student s1= new GraduateStudent(38,"Gauri Chopra","B-Tech","AI");
    Student s2= new GraduateStudent(6,”Nick","M-Tech","Java");
    Student s3 = new Student();
    s1.getDetails();
    s2.getDetails();
    s3.getDetails();
} }
```

Output:

```
 /Users/gaurichopra/Desktop/java/Lab_Assignment/out/production/Lab_Assignment Q1

 id:38,name:Gauri Chopra,course:B-Tech
specialisation:AI

 id:6,name:Nick,course:M-Tech
specialisation:Java

 id:0,name:Undefined,course:btech
```

Practical: 2

Q2. 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.**

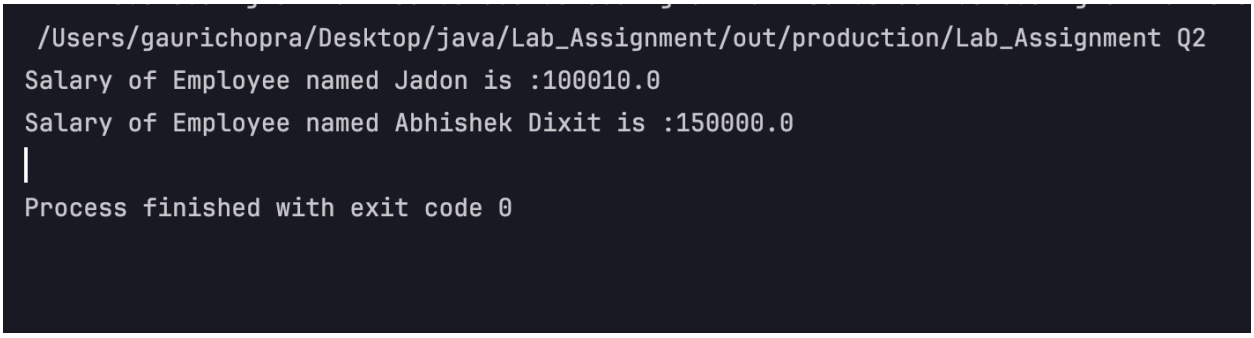
Code:

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

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

     void calculateSalary(){
        System.out.println("Employee type(Permanent/Contractual) not defined");
     }
}
class PermanentEmployee extends Employee{
    double baseSalary;
    double bonus;

    PermanentEmployee(String name, int EmployeeId, double baseSalary, double bonus){
```

```
        super(name, EmployeeId);
        this.baseSalary = baseSalary;
        this.bonus = bonus;
    }

    @Override
    void calculateSalary() {
        baseSalary = baseSalary + bonus;
        System.out.println("Salary of Employee named "+name +" is :"+ baseSalary);
    }
}
class ContractualEmployee extends Employee{
    double hoursWorked;
    double perHourWage;
    double salary;
    ContractualEmployee(String name, int EmployeeId, double hoursWorked, double
perHourWage){
        super(name, EmployeeId);
        this.hoursWorked = hoursWorked;
        this.perHourWage = perHourWage;
    }

    @Override
    void calculateSalary() {
        salary = hoursWorked * perHourWage;
        System.out.println("Salary of Employee named "+name +" is :"+ salary);
```

```
    }
}

public class Q2 {
    public static void main(String[] args){
        Employee e1 = new PermanentEmployee("Jadon", 312,100000,10);
        e1.calculateSalary();
        Employee e2 = new ContractualEmployee("Abhishek Dixit", 612,15,10000);
        e2.calculateSalary();
    }
}
```

Output:



Practical: 3

Q3. 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:

```
package BankingSystem;

public class Account {
    String name;
    double balance;
    int creditscore;


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


    public void deposit(double amount){
        balance=balance+amount;
        System.out.println("name of user : "+name);
        System.out.println("Balance after depositing: "+balance);
    }
}
package BankingSystem;
```

```
public class Loan {
int amt;
public void getLoan(Account acc, int amt){
   if(acc.creditscore>=100){
      System.out.println("Loan sanctioned of rupees "+ amt + " to "+ acc.name);}
   else{
      System.out.println("Not eligible"); }}}
```

```
import BankingSystem.*;
public class Q3 {
   public static void main(String[] args) {
      Account user1 = new Account("Gauri",150000,100);
      user1.deposit(10000);
      Loan loan = new Loan();
      loan.getLoan(user1,5000000);
      Account user2 = new Account("Java",150000,50);
      Loan loan2 = new Loan();
      loan2.getLoan(user2,5000000);  } }
```

Output:

```
name of user : Gauri
Balance after depositing: 160000.0

 Loan sanctioned of rupees 5000000 to Java
Not eligible

Process finished with exit code 0
```

Practical: 4

Q4. 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 {
   OutOfStockException(String message) {
      super(message);
   }
}

class Stock {
   int stock = 10;

   void getItem(int amt) {
      try {
         if (stock == 0) {
            throw new OutOfStockException("Product is completely out of stock!");
         } else if (amt <= stock) {
            stock -= amt;
            System.out.println(amt + " items successfully purchased.");
         } else {
```

```
            throw new OutOfStockException("Only " + stock + " items left. Cannot purchase " + amt + " items.");
        }
    } catch (OutOfStockException e) {
        System.out.println(e.getMessage());
    }
  }
}

public class Q4 {
  public static void main(String[] args) {
    Stock c1 = new Stock();
    c1.getItem(5);
    c1.getItem(6);
    c1.getItem(5);
    c1.getItem(1);
  }
}
```

Output:

```
 /Users/gaurichopra/Desktop/java/Lab_Assignment/out/production/Lab_Assignment Q4
5 items successfully purchased.
Only 5 items left. Cannot purchase 6 items.
5 items successfully purchased.
Product is completely out of stock!

Process finished with exit code 0
```

Practical: 5

Q5. A file contains a list of product names.

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

Code:

```
import java.io.*;

public class Q5 {
    public static void main(String[] args) {
        File file = new File("src/productNames.txt");
        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String line;


            System.out.println("Product Names in Uppercase:");
            while ((line = br.readLine()) != null) {
                System.out.println(line.toUpperCase());
            }
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("Error in reading file: " + e.getMessage());
        }
    }
}
```

Output:

```
 /Users/gaurichopra/Desktop/java/Lab_Assignment/out/production/Lab_Assignment Q5
Product Names in Uppercase:
BOOKS
LIP OIL
LIP GLOSS
SANITIZER
BOTTLE
BAG
SPECTACLES
LAPTOP
NOTEBOOK
MOBILE PHONE
```

Practical: 6

Q6. 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 availableTicketCount = 1;
    public void bookTicket(String userName, int ticketCount) {
        synchronized (this) {
            if (availableTicketCount >= ticketCount) {
                System.out.println(userName + " is booking " + ticketCount + " ticket...");
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    System.out.println("Interrupted");
                }
                availableTicketCount -= ticketCount;
                System.out.println("Booking successful for " + userName);
            } else {
                System.out.println("Booking failed for " + userName); } } } }

class User extends Thread {
    String name;
    int ticketCount;
```

```
    TicketBooking ticket;
    User(TicketBooking ticket, String name, int ticketCount) {
        this.ticket = ticket;
        this.name = name;
        this.ticketCount = ticketCount;
    }
    @Override
    public void run() {
        ticket.bookTicket(name, ticketCount);
    } }

public class Q6 {
    public static void main(String[] args) {
        TicketBooking ticket = new TicketBooking();
        User user1 = new User(ticket, "Abhishek", 1);
        User user2 = new User(ticket, "Gauri", 1);
        user1.start();
        user2.start();  }
 }
```

Output:

```
 /Users/gaurichopra/Desktop/java/Lab_Assignment/out/production/Lab_Assignment Q6
Abhishek is booking 1 ticket...
Booking successful for Abhishek
Booking failed for Gauri
```

Practical: 7

Q7. 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> {
    T value;
    public void set(T value) { this.value = value; }
    public T get() { return value; }}
public class Q7 {
    public static void main(String[] args) {
        Box<String> stringBox = new Box<>();
        stringBox.set("Gauri");
        String str = stringBox.get();
        System.out.println("String value: " + str);
        Box<Integer> integerBox = new Box<>();
        integerBox.set(123);
        int number = integerBox.get();
        System.out.println("Integer value: " + number);
    } }
```

Output:

```
 /Users/gaurichopra/Desktop/java/Lab_Assignment/out/production/Lab_Assignment Q7
String value: Gauri
Integer value: 123
```

Practical: 8

Q8. 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.*;


class Employe{
    String employeeName;
    String department;
    Integer id;


    Employe(String employeeName, String department, Integer id) {
        this.employeeName = employeeName;
        this.department = department;
        this.id = id;
    } }


class Company{
    ArrayList<String> employeeNames=new ArrayList<>();
    HashSet<String> departments= new HashSet<>();
    HashMap<Integer,String> employeeMap= new HashMap<>();
```

```
    void addDetails(Employe employee){
        employeeNames.add(employee.employeeName);
        departments.add(employee.department);
        employeeMap.put(employee.id, employee.employeeName);
    }
    void display(){
        System.out.println("Employee Names: "+ employeeNames);
        System.out.println("Departments: "+ departments);
        System.out.println("Id-Name Mapping "+ employeeMap);
    } }

public class Ques8 {
    public static void main(String[] args) {
        Company company = new Company();
        company.addDetails(new Employe("Gauri", "Ai", 38));
        company.addDetails(new Employe("Abhishek", "Java Development", 62));
        company.display();
} }
```

Output:

```
 /Users/gaurichopra/Desktop/java/Lab_Assignment/out/production/Lab_Assignment Ques8
Employee Names: [Gauri, Abhishek]
Departments: [Ai, Java Development]
Id-Name Mapping {38=Gauri, 62=Abhishek}
```

Practical: 9

Q9. 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:

```
# Custom Annotation
import java.lang.annotation.*;
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface TrackExecution {
    String author() default "Unknown";
    String description() default "No description";}
#Task Class Using Custom Annotation
public class Task {
    @TrackExecution(author = "Shivam", description = "Executes task A")
    public void taskA() {
        System.out.println("Running Task A...");}
    @TrackExecution(author = "Gauri", description = "Executes task B")
    public void taskB() {
        System.out.println("Running Task B..."); }
    public void notTracked() {
        System.out.println("This method is not tracked."); }}
#Using Reflection
```

```
import java.lang.reflect.Method;
public class AnnotationDemo {
    public static void main(String[] args) {
        Task task = new Task();
        Class<?> clazz = task.getClass();
        for (Method method : clazz.getDeclaredMethods()) {
            if (method.isAnnotationPresent(TrackExecution.class)) {
                TrackExecution te = method.getAnnotation(TrackExecution.class);
                System.out.println("---- Annotation Found ----");
                System.out.println("Method: " + method.getName());
                System.out.println("Author: " + te.author());
                System.out.println("Description: " + te.description());
                System.out.println();}}}}
```

```
  java AnnotationDemo
---- Annotation Found ----
Method: taskA
Author: Gauri Chopra
Description: Executes task A

Process finished with exit code 0
```

Practical: 10

Q10. A Java project requires external libraries like MySQL Connector.

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

b. **Also, explain how Maven automatically downloads and manages this dependency.**

Code:

a)

#pom.xml dependency code

```
<dependencies>
   <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-j</artifactId>
      <version>8.3.0</version>
   </dependency>
</dependencies>
```

b)

1. You add the dependency in pom.xml.
2. Maven reads the file and checks the required library.
3. Maven searches for the library in the Maven Central Repository.
4. If found, Maven downloads the JAR file automatically.
5. The downloaded JAR is stored in the local .m2/repository folder.
6. Maven then adds the JAR to the project classpath, so Java can use it.
7. If the project needs other libraries (transitive dependencies), Maven downloads them too.
8. When the version is updated in pom.xml, Maven downloads the new version and manages old ones separately.