Experiment 3

StudentName:Akash Tiwary **UID:22BCS12103**
Branch: CSE **Section/Group:636/B**
Semester: 6th **DOP:19/1/2025**
Subject: Java Lab **Subject Code: 22CSH-359**

Aim: Create an application to calculate interest for FDs, RDs based on certain conditions using inheritance.

Objective: To develop a Java application that calculates interest for Fixed Deposits (FDs) and Recurring Deposits (RDs) using object-oriented programming principles. The application will use inheritance to define common properties and methods for accounts while providing specific implementations for FDs and RDs based on their respective conditions.

Algorithm:

- **Create Account class** with attributes: accountHolderName, principal, rateOfInterest. Include methods for calculating interest (to be overridden) and displaying details.
- **Create FixedDeposit subclass** that calculates FD interest using: principal * rateOfInterest * tenureInYears / 100. Display FD details.
- **Create RecurringDeposit subclass** that calculates RD interest using: (monthlyDeposit * months *
  (months + 1) / 2) * (rateOfInterest / (12 * 100)). Display RD details.
- **In main method**, create instances of FixedDeposit and RecurringDeposit and display their details.
-

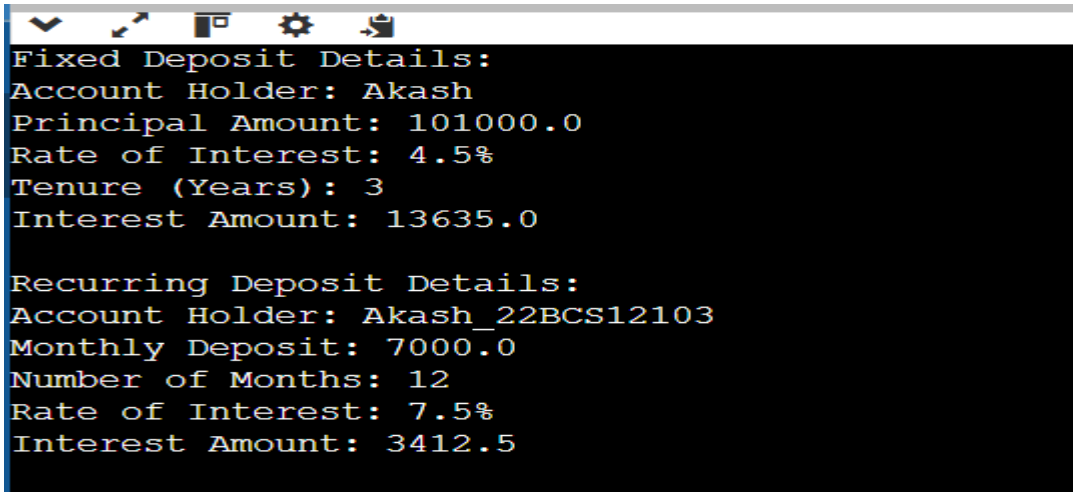
Code:

```
class Account {
   String accountHolderName;
   double principal;
   double rateOfInterest;
   public Account(String accountHolderName, double principal, double rateOfInterest) {
      this.accountHolderName = accountHolderName;
      this.principal = principal;
      this.rateOfInterest = rateOfInterest;
   }
   public double calculateInterest() {
      return 0.0;
   }
   public void displayDetails() {
      System.out.println("Account Holder: " + accountHolderName);
      System.out.println("Principal Amount: " + principal);
      System.out.println("Rate of Interest: " + rateOfInterest + "%");
   }
```

```
}
class FixedDeposit extends Account {
    int tenureInYears;
    public FixedDeposit(String accountHolderName, double principal, double rateOfInterest, int
tenureInYears) {
        super(accountHolderName, principal, rateOfInterest);
        this.tenureInYears = tenureInYears;
    }
    public double calculateInterest() {
        return principal * rateOfInterest * tenureInYears / 100;
    }
    public void displayDetails() {
        super.displayDetails();
        System.out.println("Tenure (Years): " + tenureInYears);
        System.out.println("Interest Amount: " + calculateInterest());
    }
}
class RecurringDeposit extends Account {
    int months;
    double monthlyDeposit;
    public RecurringDeposit(String accountHolderName, double monthlyDeposit, double rateOfInterest,
int months) {
        super(accountHolderName, 0, rateOfInterest);
        this.monthlyDeposit = monthlyDeposit;
        this.months = months;
    }
    public double calculateInterest() {

        double n = months;
        return (monthlyDeposit * n * (n + 1) / 2) * (rateOfInterest / (12 * 100));
    }
    public void displayDetails() {
        System.out.println("Account Holder: " + accountHolderName);
        System.out.println("Monthly Deposit: " + monthlyDeposit);
        System.out.println("Number of Months: " + months);
        System.out.println("Rate of Interest: " + rateOfInterest + "%");
        System.out.println("Interest Amount: " + calculateInterest());
    }
}
public class InterestCalculator {
    public static void main(String[] args) {
        FixedDeposit fd = new FixedDeposit("Akash", 101000, 4.5, 3);
        System.out.println("Fixed Deposit Details:");
        fd.displayDetails();
        System.out.println();
        RecurringDeposit rd = new RecurringDeposit("Akash_22BCS12103", 7000, 7.5, 12);
        System.out.println("Recurring Deposit Details:");
        rd.displayDetails();
    } }
```

Output:



Learning Outcomes:

- **Inheritance**: Use of base and derived classes for shared attributes and methods.
- **Method Overriding**: Custom implementation of methods in subclasses.
- **Constructor**: Initializing object attributes using constructors.
- **Encapsulation**: Storing and manipulating data within objects.
- **Polymorphism**: Different behavior of `calculateInterest()` based on object type.
- **Interest Calculation**: Implementing FD and RD interest formulas.
- **Class Interaction**: Creating objects and calling methods to display details.

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

Experiment 2

Student Name: Akash Tiwary **UID: 22BCS12103**
Branch: BE-CSE **Section/Group: 636/A**
Semester:6th **Date of Performance: 20/01/2025**
Subject Name: Project Based Learning in Java with Lab **Subject Code: 22CSH-359**

1. **Aim:** The aim of this project is to design and implement a simple inventory control system for a small video rental store. Define least two classes: a class Video to model a video and a class Video Store to model the actual store.
   Assume that an object of class Video has the following attributes:
   1. A title;
   2. a flag to say whether it is checked out or not;
   3. An average user rating.

Add instance variables for each of these attributes to the Video class.

In addition, you will need to add methods corresponding to the following:
1. being checked out;
2. being returned;
3. receiving a rating.

The VideoStore class will contain at least an instance variable that references an array of videos (say of length 10). The VideoStore will contain the following methods:
1. addVideo(String): add a new video (by title) to the inventory;
2. checkOut(String): check out a video (by title);
3. returnVideo(String): return a video to the store;
4. receiveRating(String, int) : take a user's rating for a video; and 5.
listInventory(): list the whole inventory of videos in the store.

2. **Objective:** Create a VideoStoreLauncher class with a main() method which will test the functionality of your other two classes. It should allow the following.
   1. Add 3 videos: "The AKASH", " AKASH II", " AKASH Wars Episode IV: A New Hope".
   2. Give several ratings to each video.
   3. Rent each video out once and return it.

   List the inventory after " AKASH II" has been rented out.

3. **Implementation/Code:**
   1. **Video Class:-**

```
class Video {
    private String title;
    private boolean checkedOut;
    private double averageRating;
    private int ratingCount;

    public Video(String title) {
        this.title = title;
        this.checkedOut = false;
        this.averageRating = 0.0;
        this.ratingCount = 0;
    }

    public void checkOut() {
        if (!checkedOut) {
            checkedOut = true;
            System.out.println("Video \"" + title + "\" has been checked out.");
        } else {
            System.out.println("Video \"" + title + "\" is already checked out.");
        }
    }
```

```
    public void returnVideo() {
        if (checkedOut) {
            checkedOut = false;
            System.out.println("Video \"" + title + "\" has been returned.");
        } else {
            System.out.println("Video \"" + title + "\" was not checked out.");
        }

    public void receiveRating(int rating) {
        if (rating < 1 || rating > 5) {
            System.out.println("Invalid rating. Please rate between 1 and 5.");
            return;
        }
        averageRating = (averageRating * ratingCount + rating) /
(++ratingCount);
        System.out.println("Received rating of " + rating + " for video \"" + title +
"\".");
    }
     public String getTitle() {
        return title;
    }
    public boolean isCheckedOut() {
        return checkedOut;
    }
    public double getAverageRating() {
        return averageRating;
    }
}
```

2. VideoStore Class:-

```
class VideoStore {
    private Video[] videos;
    private int count;
    public VideoStore(int capacity) {
        videos = new Video[capacity];
        count = 0;
    }
    public void addVideo(String title) {
        if (count < videos.length) {
            videos[count++] = new Video(title);
            System.out.println("Added video: " + title);
        } else {
            System.out.println("Inventory is full. Cannot add more videos.");
        }
    }
    public void checkOut(String title) {
        Video video = findVideo(title);
        if (video != null) {
            video.checkOut();
        } else {
            System.out.println("Video \"" + title + "\" not found.");
        }
    }
    public void returnVideo(String title) {
        Video video = findVideo(title);
        if (video != null) {
            video.returnVideo();
        } else {
            System.out.println("Video \"" + title + "\" not found.");
        }
    }
```

```
    public void receiveRating(String title, int rating) {
        Video video = findVideo(title);
        if (video != null) {
            video.receiveRating(rating);
        } else {
            System.out.println("Video \"" + title + "\" not found.");
        }
    }
    public void listInventory() {
        System.out.println("\nInventory:");
        for (int i = 0; i < count; i++) {
            Video video = videos[i];
            System.out.println("Title: " + video.getTitle() + ", Checked Out: " +
video.isCheckedOut() +
                    ", Average Rating: " + video.getAverageRating());
        }
    }
    private Video findVideo(String title) {
        for (int i = 0; i < count; i++) {
            if (videos[i].getTitle().equalsIgnoreCase(title)) {
                return videos[i];
            }
        }
        return null;
    }
}
```

3. **VideoStoreLauncher Class:-**

```
public class VideoStoreLauncher {
    public static void main(String[] args) {
        VideoStore store = new VideoStore(10);
        store.addVideo("The AKASH");
        store.addVideo("AKASH II");
        store.addVideo("AKASH Wars Episode IV: A New Hope");

        store.receiveRating("The AKASH ", 5);
        store.receiveRating("AKASH II", 4);
        store.receiveRating("AKASH Wars Episode IV: A New
        Hope", 5);

        store.checkOut("AKASH II");
        store.returnVideo("AKASH II");

        store.listInventory();
    }
}
```

4. **Output:**

```
Added video: The Akash
Added video: AKASH II
Added video: AKASH Wars Episode IV: A New Hope
Received rating of 5 for video "The Akash".
Received rating of 4 for video "AKASH II".
Received rating of 5 for video "AKASH Wars Episode IV: A New Hope".
Video "AKASH II" has been checked out.
Video "AKASH II" has been returned.

Inventory:
Title: The Akash, Checked Out: false, Average Rating: 5.0
Title: AKASH II, Checked Out: false, Average Rating: 4.0
Title: AKASH Wars Episode IV: A New Hope, Checked Out: false, Average Rating: 5.0
```

5. **Learning Outcomes:**
    1. Designed a functional system to manage video rentals, demonstrating the use of classes and objects in Java.
    2. Implemented methods for operations like adding videos, renting out, returning, and recording user ratings.
    3. Applied arrays to store and efficiently manage the video inventory within the store.
    4. Learned to integrate multiple classes and enable seamless interaction among them in a structured program.
    5. Strengthened understanding of object-oriented programming concepts like encapsulation and method abstraction.