VANIER COLLEGE

420-SF2-RE DATA STRUCTURES AND OBJECT ORIENTED PROGRAMMING
section 00002

Deliverable 4

Aminul Islam Goldar

Presented to Yi Wang

Table of Contents

1. Project Description
2. Program Features and Screenshots
3. Challenges Faced
4. Learning Outcomes

1. Scenario:

MoneyReps is an exercise game that tracks repetition of exercise and rewards users based on exercise performance and difficulty, and it serves personal fitness consumers, gyms, and competitive environments where rewards can be earned for points.

Design Paradigm:

- Users log in as an Athlete or a Trainer
- Athletes can:
    - Record exercises and reps
    - Earn points per exercise based on difficulty
    - View workout history and earnings
- Trainers can:
    - Add or update exercises
    - Assign workouts to athletes
    - View athlete performance

Expected Output:

- Athletes track reps and gain points
- Points are earned and displayed with exercise history
- Trainers control exercises and monitor athlete information

Hierarchies:

- User → Trainer / Athlete
- Exercise → Push / Pull / Core

Interface & Polymorphism:

- Redeemable interface with calculateEarnings(int reps)
- Method overriden in each Exercise type to contain different logic
- displayDetails() method polymorphically used

Additional Features:

- Exercise implements Comparable for sorting by difficulty
- Custom EarningsComparator and RepCountComparator
- Defensive checks against negative reps and difficulty

GitHub: https://github.com/Nads24/MoneyReps

2. Program Features and Screenshots

The project runs in a *fitness app* simulation where a **Trainer** assigns workouts to an **Athlete**, and the Athlete logs them to earn points. There is no user input — everything is hardcoded.

```
Trainer trainer = new Trainer( username: "CoachAminul",  id: 100);
Athlete athlete = new Athlete( username: "Yi",  id: 200);
```

Design Paradigm

Code Locations:

- Trainer.java – addOrUpdateExercise(), assignExerciseToAthlete()
- Athlete.java – logExercise(), assignExercises(), displayInfo()

This demonstrates role-based functionality, ligging and earning points, assigning and managing exercises and viewable stats

Expected Output

```
--- Trainer Info ---
Trainer: CoachAminul (ID: 100)
Exercises Managed:
Push Exercise: Pushup, Difficulty: -2
Pull Exercise: Pullup, Difficulty: 3
Core Exercise: Plank, Difficulty: 1
Assigned Exercises:
To Yi:
Push Exercise: Pushup, Difficulty: -2
Pull Exercise: Pullup, Difficulty: 3

--- Athlete Info ---
Athlete: Yi (ID: 200)
Total Earnings: 35 points
Workout History:
Assigned Exercises:
Push Exercise: Pushup, Difficulty: -2
Pull Exercise: Pullup, Difficulty: 3
Pushup: -10 reps, 20 points
Pullup: 5 reps, 15 points

--- Updated Athlete Info ---
Athlete: Yi (ID: 200)
Total Earnings: 51 points
Workout History:
Assigned Exercises:
Push Exercise: Pushup, Difficulty: -2
Pull Exercise: Pullup, Difficulty: 3
Core Exercise: Crunches, Difficulty: 2
Pushup: -10 reps, 20 points
Pullup: 5 reps, 15 points
Crunches: 8 reps, 16 points
```

This demonstrates console output of trainer and ahthlete info, earnings, workout history and assigned exercises shown per role

Hierarchies

- User → Trainer, Athlete
  User.java (abstract) → Trainer.java, Athlete.java
- Exercise → Push, Pull, Core
  Exercise.java (abstract) → Push.java, Pull.java, Core.java

Interface

- Redeemable.java

```
public interface Reedemable {  1 usage  4 implementations  Nads

    /**
     * Calculates the number of points or earnings earned
     * for performing a given number of repetitions.
     *
     * @param reps the number of repetitions completed
     * @return the amount of points earned
     */

    int calculateEarnings(int reps);  8 usages  3 implementations  Nads
}
```

Runtime Polymorphism

- **Method:** calculateEarnings (int reps)
  Classes: Push, Pull, Core

```
    @Override  8 usages  Nads
    public int calculateEarnings(int reps) { return reps * difficulty; }
```

Push.java

Also:

```
    @Override  3 usages  Nads
    public void displayDetails() {
        System.out.println("Push Exercise: " + name + ", Difficulty: " + difficulty);
    }
```

Comparable & Comparator

- **Comparable:** Exercise.java implements Comparable<Exercise>

```
    @Override  1 override  Nads
    public int compareTo(Exercise other) {
        return Integer.compare(this.difficulty, other.difficulty);
    }
}
```

Allows sorting exercises by difficulty

- **Comparator:** RepCountComparator.java implements Comparator<WorkoutLog>

```
public class RepCountComparator implements Comparator<WorkoutLog> {  no usages

    @Override  Nads
    public int compare(WorkoutLog w1, WorkoutLog w2) {
        return Integer.compare(w2.getReps(), w1.getReps());
    }
}
```

3. Challenges

- **Initial Input-Based Design**
  User-driven menus using Scanner were implemented in the initial design for login, role selection, and input of exercises. This was eventually replaced by hardcoded users and flows due to limitations in the project. Refactoring the code while not losing any functionality involved the restructuring of control flow logic in a careful manner.

- **Polymorphism and Inheritance Integration**
  Maintaining good polymorphic behavior among the Exercise subclasses (Push, Pull, Core) was challenging initially. Each subclass needed special logic for calculateEarnings() without sacrificing inter-changeability through the Redeemable interface.

- **Maintaining Role Separation**
  We had to restrict functionality based on user role (Trainer vs. Athlete), especially considering duplicate concepts such as exercise management and display. Responsibility checks at the method level were necessary to avoid role leakage and ensure encapsulation.

- **Negative Value Handling**
  There was no original prohibition against logging negative reps or setting exercises to illegal levels of difficulty. Defensive programming checks were later added to calculateEarnings() and exercise constructors to exclude these values.

- **Visibility of Assigned Exercises**
  Trainer-assigned exercises had to be passed on to the athlete's exercise list to be monitored and recorded. This created additional coordination among data exchange between Trainer and Athlete classes, particularly in assignExercises().

4. Learning Outcomes

- Utilized inheritance and polymorphism with User and Exercise hierarchies.

- Used Interfaces Effectively: Created a Redeemable interface and ensured consistency between different Exercise types.

- Practiced Runtime Polymorphism: Overridden calculateEarnings() and displayDetails() to behave differently at runtime.

- Gained Experience with Defensive Programming: Validated inputs such as reps and difficulty to ensure the system was robust.

- Improved Code Modularity: Designed loosely coupled, simple to test and extend classes.

- Developed confidence in GitHub: Utilized GitHub for project backups and version control