CS 5004 Midterm Exam - Spring 2024, Version 3

- This is a **computer-based exam**. That means that you will use your own computers to **individually** work on the problems during the designated exam time, and you will submit your solutions by **trying to push them to your individual repos on our GitHub course organization. If Github isn't available, please zip/tar your src folder, build.gradle file and any UML diagrams you may want to submit, and upload everything on Canvas.**
- The exam will be opened between 10am PT and 11:59pm PT on Tuesday, February 27th.
- Once you open the exam, you have **three hours and 20 minutes** to submit your solution.
- When pushing code to your individual repo, you will want to use our **course Gradle project** (the same Gradle project, and the corresponding build file that we have been using throughout this course).
- Additionally, we will use submission rules similar to those we followed for the homework assignments:
  - **Repository content:** Your repositories should contain only the `build.gradle` file and `src` folder with appropriate sub-folders with your code and tests.
  - **Naming convention:** Your solution to each problem should be in its own package. The package names should follow naming convention `midterm.pM`, where you replace M with the problem number, e.g., all your code for problem 1 for this exam must be in a package named `midterm.p1`.
- Some additional expectations and restrictions:
  - **Naming convention:** Please follow the naming convention for classes, methods, variables, constants, interfaces and abstract classes that we have been following throughout this whole course (camel case). Please avoid **magic numbers**.
  - **Immutability:** working in the immutable Java world is not the requirement on this exam.
  - **Methods `hashCode()`, `equals()`, `toString()`: if required,** your classes should provide appropriate implementations for methods: `boolean equals(Object o)`, `int hashCode()`, `String toString()`. Appropriate means that it is sufficient to autogenerate these methods, as long as autogenerated methods suffice for your specific implementation. Please don't forget to autogenerate your methods in an appropriate order - starting from the ancestor classes, towards the concrete classes.
  - **Testing your code:** tests are required **only if/where specifically asked for in the assignment**.
  - **Javadoc:** when asked, please include a short description of your class/method, as well as tags `@params` and `@returns` in your Javadoc documentations (code comments). Additionally, if your method throws an exception, please also include a tag `@throws` to indicate that.
  - **UML diagrams:** Please include UML diagrams only when explicitly asked for it.

Good Luck!

Question	Points	Score
1	20	
2	20	
3	20	
4	20	
5	20	
Total:	100	

Problem 1 *(20 pts)*

Consider the following code, given below, representing a class `Athlete`:

```java
package problem1;

import java.util.Objects;

/**
 * Class Name represents information about a person’s name - their first, middle and last name.
 */
public class Name {

  private String firstName;
  private String middleName;
  private String lastName;

  /**
   * Constructor for class Name
   * @param firstName first name, represented as a String
   * @param middleName middle name, represented as a String
   * @param lastName last name, represented as a String
   */
  public Name(String firstName, String middleName, String lastName) {
    this.firstName = firstName;
    this.middleName = middleName;
    this.lastName = lastName;
  }

  /**
   * Getter for field first name.
   * @return first name, represented as a String.
   */
  public String getFirstName() {
    return firstName;
  }

  /**
   * Getter for field middle name.
   * @return middle name, represented as a String.
   */
  public String getMiddleName() {
    return middleName;
  }

  /**
   * Getter for field last name.
   * @return last name, represented as a String.
   */
  public String getLastName() {
    return lastName;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) {
      return true;
    }
    if (o == null || getClass() != o.getClass()) {
      return false;
    }
    Name name = (Name) o;
    return Objects.equals(firstName, name.firstName) && Objects.equals(middleName,
        name.middleName) && Objects.equals(lastName, name.lastName);
  }

  @Override
  public int hashCode() {
    return Objects.hash(firstName, middleName, lastName);
  }

  @Override
  public String toString() {
    return "Name{" +
        "firstName='" + firstName + '\'' +
        ", middleName='" + middleName + '\'' +
        ", lastName='" + lastName + '\'' +
        '}';
  }
}

package problem1;

import java.util.Objects;

/**
Class Athlete stores information about an athlete.
 **/
public class  Athlete {

  protected Name athletesName;
  protected Double height;
  protected Double weight;
  protected String league;

  /**
  Constructor for the class Athlete.
  @param athletesName - String, representing athlete's name
  @param height - Double, representing athlete's height
  @param weight - Double, representing athlete's weight
  @param league - String, representing athlete's league
   **/
  public Athlete(Name athletesName, Double height, Double weight, String league) {
    this.athletesName = athletesName;
    this.height = height;
    this.weight = weight;
    this.league = league;
  }

  /**
 Constructor for the class Athlete.
 @param athletesName - String, representing athlete's name
 @param height - Double, representing athlete's height
 @param weight - Double, representing athlete's weight
  **/
  public Athlete(Name athletesName, Double height, Double weight) {
    this.athletesName = athletesName;
    this.height = height;
    this.weight = weight;
    this.league = null;
  }

  /**
  @return - Name, giving athlete's name
   **/
  public Name getAthletesName() {
    return athletesName;
  }

  /**
 @return - Double, giving athlete's height
  **/
  public Double getHeight() {
    return height;
  }
  /**
 @return - Double, giving athlete’s weight
  **/
  public Double getWeight() {
    return weight;
  }

  /**
 @return - String, giving athlete’s league
  **/
  public String getLeague() {
    return league;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) {
      return true;
    }
    if (o == null || this.getClass() != o.getClass()) {
      return false;
    }
    Athlete athlete = (Athlete) o;
    return Objects.equals(this.athletesName, athlete.athletesName) && Objects.equals(
        this.height, athlete.height) && Objects.equals(this.weight, athlete.weight)
        && Objects.equals(this.league, athlete.league);
  }

  @Override
  /**
   * {@inheritdoc}
   */
  public int hashCode() {
    return Objects.hash(this.athletesName, this.height, this.weight, this.league);
  }

  @Override
  public String toString() {
    return "Athlete{" +
        "athletesName=" + athletesName +
        ", height=" + height +
        ", weight=" + weight +
        ", league=’" + league + ’\’’ +
        ’}’;
  }
}

package problem1;

import java.util.Objects;

/**
 * Class Runner stores information about a runner’s best 5k run time, best half-marathon run time,
 * and their favorite running event.
 */
public class Runner extends Athlete {

  private Double best5KTime;
  private Double bestHalfMarathonTime;
  private String favoriteRunningEvent;

  /**
   * Constructor for class Runner.
   * @param athletesName runner’s name, represented as custom class Name
   * @param height runner’s height, represented as Double
   * @param weight runner’s weight, represented as Double
```

```java
   * @param league runner's league, represented as String
   * @param best5KTime runner's best 5k time, represented as Double
   * @param bestHalfMarathonTime runner's best half-marathon time, represented as a Double
   * @param favoriteRunningEvent runner's favorite running event, represented as a String
   */
  public Runner(Name athletesName, Double height, Double weight, String league, Double best5KTime,
      Double bestHalfMarathonTime, String favoriteRunningEvent) {
    super(athletesName, height, weight, league);
    this.best5KTime = best5KTime;
    this.bestHalfMarathonTime = bestHalfMarathonTime;
    this.favoriteRunningEvent = favoriteRunningEvent;
  }

  /**
   * Anothre constructor for class Runner.
   * @param athletesName runner's name, represented as custom class Name
   * @param height runner's height, represented as Double
   * @param weight runner's weight, represented as Double
   * @param best5KTime runner's best 5k time, represented as Double
   * @param bestHalfMarathonTime runner's best half-marathon time, represented as a Double
   * @param favoriteRunningEvent runner's favorite running event, represented as a String
   */
  public Runner(Name athletesName, Double height, Double weight, Double best5KTime,
      Double bestHalfMarathonTime, String favoriteRunningEvent) {
    super(athletesName, height, weight);
    this.best5KTime = best5KTime;
    this.bestHalfMarathonTime = bestHalfMarathonTime;
    this.favoriteRunningEvent = favoriteRunningEvent;
  }

  /**
   * Getter for the best 5k running time.
   * @return 5k running time, as Double
   */
  public Double getBest5KTime() {
    return best5KTime;
  }

  /**
   * Getter for the best half-marathon time.
   * @return half-marathon time, as Double
   */
  public Double getBestHalfMarathonTime() {
    return bestHalfMarathonTime;
  }

  /**
   * Getter for the favorite running event.
   * @return favorite running event, as String.
   */
  public String getFavoriteRunningEvent() {
    return favoriteRunningEvent;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) {
      return true;
    }
    if (o == null || getClass() != o.getClass()) {
      return false;
    }
    if (!super.equals(o)) {
      return false;
    }
    Runner runner = (Runner) o;
    return Objects.equals(best5KTime, runner.best5KTime) && Objects.equals(
        bestHalfMarathonTime, runner.bestHalfMarathonTime) && Objects.equals(
        favoriteRunningEvent, runner.favoriteRunningEvent);
  }

  @Override
  public int hashCode() {
    return Objects.hash(super.hashCode(), best5KTime, bestHalfMarathonTime, favoriteRunningEvent);
  }

  @Override
  public String toString() {
    return "Runner{" +
        "best5KTime=" + best5KTime +
        ", bestHalfMarathonTime=" + bestHalfMarathonTime +
        ", favoriteRunningEvent='" + favoriteRunningEvent + '\'' +
        ", athletesName=" + athletesName +
        ", height=" + height +
        ", weight=" + weight +
        ", league='" + league + '\'' +
        '}';
  }

}

package problem1;

import java.util.Objects;

/**
 * Class BaseballPlayer stores information about a baseball player, includding their team,
 * their average batting score and the number of home runs they had in a season.
 */
public class BaseballPlayer extends Athlete{

  private static final Double AVG_BATTING_SCORE_LIMIT = 0.0;
  private String team;
  private Double avgBattingScore;
  private Integer seasonHomeRuns;

  /**
   * Constructor for BaseballPlayer.
   * @param athletesName baseball player's name, as Name
   * @param height baseball player's height, as Double
   * @param weight baseball player's weight, as Double
   * @param league baseball player's league, as String
   * @param team baseball player's team, as String
   * @param avgBattingScore average batting score, as Double
   * @param seasonHomeRuns season home runs, as Integer
   */
  public BaseballPlayer(Name athletesName, Double height, Double weight, String league, String team,
      Double avgBattingScore, Integer seasonHomeRuns) throws InvalidBattingScoreException {
    super(athletesName, height, weight, league);
    if(this.validateAvgBattingScore(avgBattingScore)) {
      this.team = team;
      this.avgBattingScore = avgBattingScore;
      this.seasonHomeRuns = seasonHomeRuns;
    }
  }

  /**
   * Another constructor for BaseballPlayer.
   * @param athletesName baseball player's name, as Name
   * @param height baseball player's height, as Double
   * @param weight baseball player's weight, as Double
   * @param team baseball player's team, as String
   * @param avgBattingScore average batting score, as Double
   * @param seasonHomeRuns season home runs, as Integer
   */
  public BaseballPlayer(Name athletesName, Double height, Double weight, String team,
      Double avgBattingScore, Integer seasonHomeRuns) throws InvalidBattingScoreException {
    super(athletesName, height, weight);
    if(this.validateAvgBattingScore(avgBattingScore)) {
      this.team = team;
      this.avgBattingScore = avgBattingScore;
      this.seasonHomeRuns = seasonHomeRuns;
    }
  }

  private Boolean validateAvgBattingScore(Double avgBattingScore) throws InvalidBattingScoreException {
    if(avgBattingScore > AVG_BATTING_SCORE_LIMIT)
      return Boolean.TRUE;
    else throw new InvalidBattingScoreException("This is not a valid batting score!");
  }

  /**
   * Getter for the team.
   * @return team, as String
   */
  public String getTeam() {
    return this.team;
  }

  /**
   * Getter for the average batting score.
   * @return average batting score, as Double
   */
  public Double getAvgBattingScore() {
    return this.avgBattingScore;
  }

  /**
   * Getter for season home runs.
   * @return season home runs, as Integer.
   */
  public Integer getSeasonHomeRuns() {
    return this.seasonHomeRuns;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) {
      return true;
    }
    if (o == null || getClass() != o.getClass()) {
      return false;
    }
    if (!super.equals(o)) {
      return false;
    }
    BaseballPlayer that = (BaseballPlayer) o;
    return Objects.equals(team, that.team) && Objects.equals(avgBattingScore,
        that.avgBattingScore) && Objects.equals(seasonHomeRuns, that.seasonHomeRuns);
  }

  @Override
  public int hashCode() {
    return Objects.hash(super.hashCode(), team, avgBattingScore, seasonHomeRuns);
  }

  @Override
  public String toString() {
    return "BaseballPlayer{" +
            "team='" + team + '\'' +
            ", avgBattingScore=" + avgBattingScore +
            ", seasonHomeRuns=" + seasonHomeRuns +
            ", athletesName=" + athletesName +
            ", height=" + height +
            ", weight=" + weight +
            ", league='" + league + '\'' +
            '}';
  }
}

package problem1;

public class InvalidBattingScoreException extends Exception {

  public InvalidBattingScoreException(String message) {
    super(message);
  }
}
```

(a) Please create a file **Problem1.md**, and in this file include a code snippet showing an example of **input validation**. ***(4 pts)***

(b) In the same file **Problem1.md** include a code snippet showing **inheritance**. ***(4 pts)***

(c) In the same file **Problem1.md** include a code snippet showing **method overriding**. ***(4 pts)***

(d) In the same file **Problem1.md** include a code snippet showing **casting**. ***(4 pts)***

(e) In the same file **Problem1.md** include a code snippet of **method overloading.** ***(4 pts)***

Problem 2 *(20 pts)*

Consider the following code, given below, representing a class `GameCharacter`:

```
package problem2;

import java.util.Objects;

/**
 * Class Name represents information about a person's name - their first, middle and last name.
 */
public class Name {

  private String firstName;
  private String middleName;
  private String lastName;

  /**
   * Constructor for class Name
   * @param firstName first name, represented as a String
   * @param middleName middle name, represented as a String
   * @param lastName last name, represented as a String
   */
  public Name(String firstName, String middleName, String lastName) {
    this.firstName = firstName;
    this.middleName = middleName;
    this.lastName = lastName;
  }

  /**
   * Getter for field first name.
   * @return first name, represented as a String.
   */
  public String getFirstName() {
    return firstName;
  }

  /**
   * Getter for field middle name.
   * @return middle name, represented as a String.
   */
  public String getMiddleName() {
    return middleName;
  }

  /**
   * Getter for field last name.
   * @return last name, represented as a String.
   */
  public String getLastName() {
    return lastName;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) {
      return true;
    }
    if (o == null || getClass() != o.getClass()) {
      return false;
    }
    Name name = (Name) o;
    return Objects.equals(firstName, name.firstName) && Objects.equals(middleName,
        name.middleName) && Objects.equals(lastName, name.lastName);
  }

  @Override
  public int hashCode() {
    return Objects.hash(firstName, middleName, lastName);
  }

  @Override
  public String toString() {
    return "Name{" +
        "firstName='" + firstName + '\'' +
        ", middleName='" + middleName + '\'' +
        ", lastName='" + lastName + '\'' +
        '}';
  }
}

package problem2;

import java.util.Objects;

public class GameCharacter {

  private static final Double COMBAT_POWER_LIMIT = 0.0;
  private static final Double MIN_HEALTH_LEVEL = 0.0;
  private static final Double MAX_HEALTH_LEVEL = 100.0;
  protected Name name;
  protected Integer age;
  protected Double combatPower;
  protected Double healthLevel;
  protected Double wealthLevel;

  public GameCharacter(Name name, Integer age, Double combatPower, Double healthLevel)
      throws CombatPowerException, HealthLevelException {
    if(this.validateCombatPower(combatPower) && this.validateHealthLevel(healthLevel)) {
      this.name = name;
      this.age = age;
      this.combatPower = combatPower;
      this.healthLevel = healthLevel;
      this.wealthLevel = 0.0;
    }
  }

  private Boolean validateCombatPower(Double combatPower) throws CombatPowerException {
    if (combatPower > COMBAT_POWER_LIMIT)
      return Boolean.TRUE;
    else throw new CombatPowerException("Invalid combat power limit");
  }

  private Boolean validateHealthLevel(Double healthLevel) throws HealthLevelException {
    if ((healthLevel >= MIN_HEALTH_LEVEL) && (healthLevel <= MAX_HEALTH_LEVEL))
      return Boolean.TRUE;
    else throw new HealthLevelException("Invalid health level");
  }

  public Name getName() {
    return name;
  }

  public Integer getAge() {
    return age;
  }

  public Double getCombatPower() {
    return combatPower;
  }

  public Double getHealthLevel() {
    return healthLevel;
  }
  public Double getWealthLevel() {
    return wealthLevel;
  }

  //YOUR CODE HERE
  public void increaseWealthLevel(Double wealthLevel) throws WealthLevelException {

  }

  @Override
  public boolean equals(Object o) {
    if (this == o) {
      return true;
    }
    if (o == null || getClass() != o.getClass()) {
      return false;
    }
    GameCharacter that = (GameCharacter) o;
    return Objects.equals(name, that.name) && Objects.equals(age, that.age)
        && Objects.equals(combatPower, that.combatPower) && Objects.equals(
        healthLevel, that.healthLevel) && Objects.equals(wealthLevel, that.wealthLevel);
  }

  @Override
  public int hashCode() {
    return Objects.hash(name, age, combatPower, healthLevel, wealthLevel);
  }

  @Override
  public String toString() {
    return "GameCharacter{" +
        "name=" + name +
        ", age=" + age +
        ", combatPower=" + combatPower +
        ", healthLevel=" + healthLevel +
        ", wealthLevel=" + wealthLevel +
        '}';
  }
}

package problem2;

public class CombatPowerException extends Exception {

  public CombatPowerException(String message) {
    super(message);
  }
}

package problem2;

public class HealthLevelException extends Exception {

  public HealthLevelException(String message) {
    super(message);
  }
}

package problem2;

public class WealthLevelException extends Exception {

  public WealthLevelException(String message) {
    super(message);
  }
}
```

(a) Please provide Javadoc for the constructor in class `GameCharacter`. ***(3 pts)***

(b) In class `GameCharacter`, you will notice method `increaseWealthLevel`. Please implement this **`void`** method such that it does not allow the wealth level of a character to ever become negative, nor higher than 100. If an incorrect value of a wealth input is provided, the method should throw an exception, which has already been implemented for you. ***(7 pts)***

(c) Please test method `increaseWealthLevel`. ***(10 pts)***

Problem 3 ***(20 pts)***

(a) You are a part of a team developing a new software system to catalog and manage sailing events (regattas) around the world. Your team is tasked with the development of a directory system for regattas, referred to as `RegattaDirectory`, which will be used to store and manage information about regattas around the world. ***(5 pts)***

A class `Regatta`, which stores relevant information about regattas has already been developed for you.

Your ADT `RegattaDirectory` will need to support the following functionality:

- Check whether or not the `RegattaDirectory` is empty.
- Checks if a `Regatta` with the requested name exists in the `RegattaDirectory`.
- Remove a specified `Regatta` from the `RegattaDirectory`. If the `Regatta` does not exist in the `RegattaDirectory`, the system should throw `RegattaNotFoundException`, which you will have to implement yourself.

The `RegattaDirectory` is envisioned as a **collection** of `Regattas`, and it can be mutable or immutable. **Please write an interface for the `RegattaDirectory` ADT. Please include Javadoc for every method.**

(b) The `RegattaDirectory` is envisioned as a collection of `Regattas`. Please implement `RegattaDirectory` using either Java arrays, or your own implementation of a sequential linked list. ***(15 pts)***

```java
package problem3;

import java.time.LocalDate;
import java.util.Objects;
import problem3.RegattaLevel;


/**
 * Class Regatta stores relevant information about sailing regattas.
 */
public class Regatta {

  private String regattaID;
  private String regattaName;
  private String country;
  private String startingPoint;
  private String stoppingPoint;
  private Double mileageDuration;
  private LocalDate dateOfEvent;
  private RegattaLevel regattaLevel;
  private Boolean annualEvent;

  /**
   * Constructor for the class Regatta.
   * @param regattaID unique identifier of a regatta, represented as a String.
   * @param regattaName regatta’s name, represented as a String.
   * @param country country where the regatta is being held, represented as a String.
   * @param startingPoint starting point of the regatta, represented as a String.
   * @param stoppingPoint stopping point of the regatta, represented as a String.
   * @param mileageDuration duration of the regatta in miles, represented as a Double.
   * @param dateOfEvent date of the regatta, represented as LocalDate.
   * @param regattaLevel regatta level, represented as custom data type RegattaLevel.
   * @param annualEvent Boolean flag indicating if the regatta is annual.
   */
  public Regatta(String regattaID, String regattaName, String country, String startingPoint,
      String stoppingPoint, Double mileageDuration, LocalDate dateOfEvent,
      RegattaLevel regattaLevel,
      Boolean annualEvent) {
    this.regattaID = regattaID;
    this.regattaName = regattaName;
    this.country = country;
    this.startingPoint = startingPoint;
    this.stoppingPoint = stoppingPoint;
    this.mileageDuration = mileageDuration;
    this.dateOfEvent = dateOfEvent;
    this.regattaLevel = regattaLevel;
    this.annualEvent = annualEvent;
  }

  /**
   * Getter for the regatta’s ID.
   * @return regatta’s ID, represented as a String.
   */
  public String getRegattaID() {
    return regattaID;
  }

  /**
   * Getter for the regatta’s name.
   * @return regatta’s name, represented as a String.
   */
  public String getRegattaName() {
    return regattaName;
  }

  /**
```

```
   * Getter for the country of the regatta.
   * @return country, represented as a String.
   */
  public String getCountry() {
    return country;
  }

  /**
   * Getter for the staring point of the regatta.
   * @return starting point, represented as a String.
   */
  public String getStartingPoint() {
    return startingPoint;
  }

  /**
   * Getter for the end point of the regatta.
   * @return end points, represented as a String.
   */
  public String getStoppingPoint() {
    return stoppingPoint;
  }

  /**
   * Getter for teh duration of the regatta, in miles.
   * @return mileage of the regatta, represented as Doubles.
   */
  public Double getMileageDuration() {
    return mileageDuration;
  }

  /**
   * Getter for the date of the regatta.
   * @return date of the regatta, represented as LocalDate.
   */
  public LocalDate getDateOfEvent() {
    return dateOfEvent;
  }

  /**
   * Getter for the regatta level.
   * @return regatta level, represented as custom data type RegattaLevel.
   */
  public RegattaLevel getRegattaLevel() {
    return regattaLevel;
  }

  /**
   * Getter for a Boolean flag indicating if the regatta is an annual event.
   * @return Boolean true if the regatta is an annual event.
   */
  public Boolean getAnnualEvent() {
    return annualEvent;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) {
      return true;
    }
    if (o == null || getClass() != o.getClass()) {
      return false;
    }
    Regatta regatta = (Regatta) o;
    return Objects.equals(regattaID, regatta.regattaID) && Objects.equals(
        regattaName, regatta.regattaName) && Objects.equals(country, regatta.country)
        && Objects.equals(startingPoint, regatta.startingPoint) && Objects.equals(
        stoppingPoint, regatta.stoppingPoint) && Objects.equals(mileageDuration,
        regatta.mileageDuration) && Objects.equals(dateOfEvent, regatta.dateOfEvent)
        && regattaLevel == regatta.regattaLevel && Objects.equals(annualEvent,
        regatta.annualEvent);
  }

  @Override
  public int hashCode() {
    return Objects.hash(regattaID, regattaName, country, startingPoint, stoppingPoint,
        mileageDuration, dateOfEvent, regattaLevel, annualEvent);
  }

  @Override
  public String toString() {
    return "Regatta{" +
        "regattaID='" + regattaID + '\'' +
        ", regattaName='" + regattaName + '\'' +
        ", country='" + country + '\'' +
        ", startingPoint='" + startingPoint + '\'' +
        ", stoppingPoint='" + stoppingPoint + '\'' +
        ", mileageDuration=" + mileageDuration +
        ", dateOfEvent=" + dateOfEvent +
        ", regattaLevel=" + regattaLevel +
        ", annualEvent=" + annualEvent +
        '}';
  }
}

package problem3;

public enum RegattaLevel {

  CLUB,
  REGIONAL_GAMES,
  NATIONAL,
  AMERICAS_CUP,
  NATIONS_CUP,
  OLYMPIC_GAMES,
  SAILING_WORLD_CUP_SERIES
}
```

Problem 4 ***(20 pts)***

(a) You are a part of a team developing a new software system managing digital assets, such as video games, at your online store. Your team is tasked with specifying and developing an ADT for a catalog of video games, referred to as `VideoGamesCatalog`, which will be used to store and manage information about video games. ***(15 pts)***

A class `VideoGame`, which stores relevant information about video games your store offers, has already been developed for you.

Your ADT `VideoGamesCatalog` will need to support the following functionality:

- Count the number of `VideoGames` in the catalog.
- Add a `VideoGame` into the catalog. Please note that the system allows duplicate `VideoGames` in the catalog.
- Find and return all `VideoGames` from the `VideoGamesCatalog` that have more than 500 000 downloads, and have been created by more than five creators.

The `VideoGamesCatalog` is envisioned as a **recursive collection** of `VideoGames`. **Please write an interface for the `VideoGamesCatalog` ADT, and implement it using recursive data collections**.

(b) Please create a file Problem4.md file, and in this file include a code snippet of a: ***(5 pts)***

- Method that uses a local variable. If you are not using any local variable, please briefly explain (in a sentence or two) why that was not necessary.
- Helper method you used to implement the given ADT. If you are not using any helper methods, please briefly explain why were helper methods not necessary.

```java
package problem4;

import java.time.LocalDate;
import java.util.Arrays;
import java.util.Objects;
import problem4.VideoGameRating;

/**
 * Class VideoGame contains information about a video game - its name, creators, dates of creations
 * and revision, as well as its rating.
 */
public class VideoGame {

  private String gameID;
  private String name;
  private String[] creators;
  private LocalDate dateOfCreation;
  private LocalDate dateOfLastVersion;
  private Integer numDownloads;
  private VideoGameRating videoGameRating;
  private Boolean containsViolence;

  /**
   * Constructor for a class VideoGame.
   * @param gameID unique ID of the game, represented as a String.
   * @param name video game's name, represented as a String.
   * @param creators list of game's creators, represented as an array of Strings.
   * @param dateOfCreation date of the game's creation, represented as a LocalDate.
   * @param dateOfLastVersion date of the game's last version, represented as a LocalDate.
   * @param numDownloads number of downloads, represented as an Integer.
   * @param videoGameRating rating of the video game, represented as a custom data type
   *                        VideoGameRating.
   * @param containsViolence Boolean flag set to True if teh game contains violence.
   */
  public VideoGame(String gameID, String name, String[] creators, LocalDate dateOfCreation,
      LocalDate dateOfLastVersion, Integer numDownloads, VideoGameRating videoGameRating,
      Boolean containsViolence) {
    this.gameID = gameID;
    this.name = name;
    this.creators = creators;
    this.dateOfCreation = dateOfCreation;
    this.dateOfLastVersion = dateOfLastVersion;
    this.numDownloads = numDownloads;
    this.videoGameRating = videoGameRating;
    this.containsViolence = containsViolence;
  }

  /**
   * Getter for the unique ID of the video game.
   * @return ID, represented as String
   */
  public String getGameID() {
    return gameID;
  }

  /**
   * Getter for the name of the video game.
   * @return name, represented as String.
   */
  public String getName() {
    return name;
  }

  /**
   * Getter for the list of the game's creators.
   * @return creators, represented as an array of Strings.
   */
  public String[] getCreators() {
    return creators;
  }

  /**
   * Getter for the date of the game’s original creation.
   * @return date of creation, represented as LocalDate.
   */
  public LocalDate getDateOfCreation() {
    return dateOfCreation;
  }

  /**
   * Getter for the date of the game’s latest version.
   * @return date of the latest version, represented as LocalDate.
   */
  public LocalDate getDateOfLastVersion() {
    return dateOfLastVersion;
  }

  /**
   * Getter for the total number of downloads for the game.
   * @return number of downloads, represented as an Integer.
   */
  public Integer getNumDownloads() {
    return numDownloads;
  }

  /**
   * Getter for the game’s rating.
   * @return game’s rating, represented as a custom data type VideoGameRating.
   */
  public VideoGameRating getVideoGameRating() {
    return videoGameRating;
  }

  /**
   * Getter for a Boolean flag indicating if the game contains violence.
   * @return Boolean true if the game contains violence.
   */
  public Boolean getContainsViolence() {
    return containsViolence;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) {
      return true;
    }
    if (o == null || getClass() != o.getClass()) {
      return false;
    }
    VideoGame videoGame = (VideoGame) o;
    return Objects.equals(gameID, videoGame.gameID) && Objects.equals(name,
        videoGame.name) && Arrays.equals(creators, videoGame.creators)
        && Objects.equals(dateOfCreation, videoGame.dateOfCreation)
        && Objects.equals(dateOfLastVersion, videoGame.dateOfLastVersion)
        && Objects.equals(numDownloads, videoGame.numDownloads)
        && videoGameRating == videoGame.videoGameRating && Objects.equals(containsViolence,
        videoGame.containsViolence);
  }

  @Override
  public int hashCode() {
    int result = Objects.hash(gameID, name, dateOfCreation, dateOfLastVersion, numDownloads,
        videoGameRating, containsViolence);
    result = 31 * result + Arrays.hashCode(creators);
    return result;
  }

  @Override
  public String toString() {
    return "VideoGame{" +
        "gameID='" + gameID + '\'' +
        ", name='" + name + '\'' +
        ", creators=" + Arrays.toString(creators) +
        ", dateOfCreation=" + dateOfCreation +
        ", dateOfLastVersion=" + dateOfLastVersion +
        ", numDownloads=" + numDownloads +
        ", videoGameRating=" + videoGameRating +
        ", containsViolance=" + containsViolence +
        '}';
  }
}
```

```
package problem4;

public enum VideoGameRating {

  EVERYONE,
  EVERYONE_10_PLUS,
  TEEN,
  MATURE,
  ADULTS_ONLY
}
```

Problem 5 ***(20 pts)***

(a) You are a part of a software development company, developing a new reservation system for charter boats. At this development stage, your goal is to implement the backbone infrastructure for charter boats. ***(10 pts)***

The reservation system distinguishes between two types of **Charter boats:**

- Sail boats
- Motor boats

For every **Charter boat**, the system keeps track of the following:

- **Boat ID**, a unique charter boat's identifier, represented as a `String`.
- **Boat length**, length of the boat, represented as `Double`.
- **Price per day**, daily price of renting (chartering) this boat, represented as `Double`.
- **Manufacturing year**, manufacturing year of the boat, represented as an `Integer`.
- **Number of cabins**, number of cabins on the boat, represented as an `Integer`.
- **Skipper needed**, a `Boolean` flag indicating if the boat needs to be chartered with a skipper.

Additionally, for every **Motor Boat**, the system keeps track of:

- **Engine power**, boat's engine power, represented as a `Double`.

Please implement `CharterBoat`, and its hierarchy, and provide the UML Class diagram for your complete design.

(b) The reservation system provides a functionality of estimating an annual revenue for every charter boat in its fleet. In doing so, it relies on the following rules: ***(10 pts)***

- **Basic annual estimation:** Basic estimate is that every charter boat will be rented (chartered) for at least 80 days yearly, producing a basic annual estimate equal to 80*price per day.
- **Young motor boats increase:** Motor boats manufactured after 2010 are considered young, and more modern and sleek. They typically get chartered mode, so they get a multiplication increase of 1.35 to their basic annual estimation.
- **Large sail boats increase:** Sail boats larger than 45ft are typically chartered more, so they get a multiplication increase of 1.15 to their basic annual estimation.

Please design and implement method `estimateAnnualBoatRevenue()`, whose return type is `Double`. That means that when we call method `estimateAnnualBoatRevenue()` on some `CharterBoat`, it should return an estimated annual revenue for that charter boat as `Double`.