Programming III

Laboratory 5

Objectives

- Classes, interfaces, inheritance
- Collections
- Generics

Exercises

1. Create the following class hierarchy:

a) A class *Airplane* that has the following attributes: producer, code, number of flights, fuel capacity

b) From *Airplane* class derive classes *FightAriplain* that has the following characteristics: can or cannot camouflage and weapon capacity; *LineArplain* that has the following characteristic: maximum number of passengers and *EntertaimentAirplain* hat has the following characteristics: current owner and a list of previous owners.

d) *Line* and *entertainment* airplanes implement also the interface *LuxuryOptions* that contain

the information about the fact if a plain has: noise-cancelling headphones or personal touch screen TV for each client.

Requirements:

1. Create the class hierarchy described

2. Create a list (`LinkedList` or `ArrayList`) of airplanes and display it.

3. Create and display a map that contains the type of airplane and for each type count how many

airplanes are in the list. Display the map in the following format: planeType number of plays displayed like and arrays of stars

a) FightAirplane **

b) LineArpline *****

c) EntertainmentAirplane ***

4. Display from the list of airplanes the ones that have noise canceling headphones

5. Find and display the EntertaimentAirplain that had the most owners

6. Display the average fuel capacity of all air-plains from the list, display the average fuel capacity of fight airplanes

2. Write a generic method to count the number of elements in a collection that have a specific property (for example, odd integers, prime numbers, palindromes).

3. Write a generic method to exchange the positions of two different elements in an array.

4. Design a class that acts as a library for the following kinds of media: book, video, and newspaper. Provide one version of the class that uses generics and one that does not. Feel free to use any additional APIs for storing and retrieving the media.

5. Joe Mocha is defining an interface Appendable that includes an append method. He then defines two classes, MyString and MyList, which both implement Appendable. He wants Java's type system to allow a MyString to be appended to a MyString, and a MyList to be appended to a MyList, but not MyString to a MyList, or a MyList to a MyString. Here is his definition of Appendable:

```
interface Appendable {
        Appendable append(Appendable a);
}
```

What is wrong with this definition? What is a correct one? Also write a definition for a classes MyString and MyList and uses the revised definition of Appendable.

Homework deadline 2 weeks

Foreword.

The interfaces defined below:

- are given as part of the requirement
- will be the basis of your implementation
- are fixed and cannot be modified

Your solution will be tested by a test application that will:

- validate the behavior of your classes
- use classes and interfaces define below

Make the complete implementation of ClubFactory and ClubArchive classes presented below.

Note that:

- the documentation of the class and interface methods stand as requirements
- you will inherently need to define classes that implement below defined interfaces, as needed by ClubFactory implementation

You will also need to make use of the following imports throughout your implementation:

- java.util.Date
- java.time.DayOfWeek
- java.time.LocalDate
- java.util.Collection
- java.util.List

```
- java.util.Map

public interface Club {

        /**
          * @return name of the club
          */
        public String getName();

        /**
          * @return a number representing the year of club foundation
          */
        public int getFoundedYear();

        /**
          * @return current address of the club
          */
        public String getAddress();

        /**
          * Changes the address of the club
          * @param address: the new address of the club
          */
        public void setAddress(String address);
}


/**
  * @apiNote Interface for managing working hours (open hours -> closing hours)
  *
  */
public interface TimeTable {

        /**
          * @param dayOfWeek: one of the 7 week days, e.g. DayOfWeek.MONDAY
          * @param openHours: a string containing the opening time in "HH:mm" format, e.g. "08:30"
          * @param closingHours: a string containing the closing time in "HH:mm" format, e.g. "22:30"
          */
```

```
	public void setWorkingHours(DayOfWeek dayOfWeek, String openHours, String closingHours);

	/**
	 *
	 * @param dateTime: a given date and time
	 * @return true if the given date and time is during working hours, false otherwise
	 */
	public boolean isOpen(LocalDate dateTime);
}


public interface Gym extends Club {

	/**
	 * Obtains the working hours
	 * @return an object implementing TimeTable
	 */
	public TimeTable getTimeTable();

	/**
	 * Change the working hours
	 * @param timeTable: new time table of the gym
	 */
	public void setTimeTable(TimeTable timeTable);
}


public interface ClubMember {
	/**
	 * Obtain the full name of the club member
	 * @return string containing full member's name
	 */
	public String getName();

	/**
	 * Obtain annual salary
	 * @return number containing total amount of money this member receives in an year
	 */
```

```
        public int getSalary();

        /**
          * Obtain contract expiration data
          * @return Date object
          */
        public Date getContractExpirationDate();
}

public interface FootballClub extends Club {

        /**
          * Adds a new member to the club
          * @param object of type implementing ClubMember
          */
        public void addMember(ClubMember member);

        /**
          * Obtains the collection of current members
          * @return actual collection of existing members
          */
        public Collection<ClubMember> getMembers();

        /**
          * Changes the trainer of the football club
          * @param dedicated member that has the trainer role
          * @apiNote the trainer is not part of the member collection
          */
        public void setTrainer(ClubMember member);

        /**
          * Obtains the trainer of this club
          */
        public ClubMember getTrainer();
}


public class ClubFactory {
```

```
/**
  * Creates a new member object with the given name, salary and expiration date of the contract
  * @param name
  * @param salary
  * @param contractExpirationDate
  * @return a new object, of a type that implements ClubMember
  */
public ClubMember createMember(String name, int salary, Date contractExpirationDate) {
        // todo: Add implementation
        return null;
}

/**
  * Creates a football club
  * @param name of the football club to create
  * @return a new object, of a type that implements FootballClub
  */
public FootballClub createFootballClub(String name) {
        // todo: Add implementation
        return null;
}


/**
  * Creates a gym club
  * @param name of the gym club to create
  * @return a new object, of a type that implements Gym
  */
public Gym createGym(String name) {
        // todo: Add implementation
        return null;
}

/**
  * Creates a new time table
  * @return a newly created object of type TimeTable
  */
```

```
	public TimeTable createTimeTable() {
		// todo: Add implementation
		return null;
	}

	/**
	  * Obtains the author of the implementation
	  * @return your full name
	  */
	public String getOwner() {
		// todo: Add implementation
		return null;
	}
}

public class ClubArchive {

	/**
	  * Add an existing club to archive
	  */
	public void addClub(Club club) {
		// todo: Add implementation
	}

	/**
	  * Obtains the map of the clubs in the archive, grouped by year of foundation
	  * @return a Map object, with key indicating year of foundation, and for each key a collection of
Club ojects founded in that year
	  */
	public Map<int, Collection<Club>> getClubsByYear() {
		// todo: Add implementation
		return null;
	}

	/**
	  * Obtains the football clubs from the archive with largest number of members
	  * @return a Collection containing the FootballClub objects having the highest number of
members
	  */
	public Collection<FootballClub> getLargestFootballClubs() {
		// todo: Add implementation
		return null;
	}

	/**
	 * Obtains the gyms from the archive with the longest total opening time
	 * @return a Collection containing the Gym objects having the largest time availability
	 */
	public Collection<Gym> getMostAvailableGyms() {
		// todo: Add implementation
		return null;
	}

	/**
	 * Sorts a given list of Club objects based on their year of foundation, and those founded in the
same year by their name alfabetically
	 * @param the List of Club ojects to be sorted
	 */
	public static void sortByYearAndName(List<Club> clubs) {
		// todo: Add implementation
	}
}
```

Remark. Try to resolve the requests using stream operations