Assignment 2 specification – Inheritance, Polymorphism and Abstraction

This overall assignment is worth 55% for the Programming Fundamentals II module. The interview to assess your understanding and ability to explain of the code is very important. The interview marking will be used as a multiplier to determine your final mark.

Please read this section before starting the assignment:

UX approach:

- Your menu driven app should be user friendly, report progress to the user, robust, handle exceptions, intuitive, etc.
- The menu system should be well tested for many different user-input scenarios e.g. case sensitivity, input mismatches, invalid indexes, etc.

DX approach:

- Javadoc comments should be written for each method and class. Internal "programmer" comments should be used where required also.
- Your name should appear as the @author at the top of each java file submitted.
- Java naming standards must be followed e.g.:
    - Class names begin with capital letter – lowercase after that., e.g. **Employee**
    - Field names / local variables begin with lowercase, use uppercase to separate words in name, e,g, **numRounds**
    - For instances of collections they should follow the conventions for field names and should be plural, e.g. **dvds.**
- Java coding principles / standards should be applied throughout the project.

Note on validations:

- When you wish to update fields though methods (though constructors or setters) , and the input data/new value is not valid according to the validation rules, you deal with this differently in the two cases. In consructors, you should give the field a default value,(0 if this is allowed by the rules of validation, some other sensible value otherwise) . In the setters, you should leave the field unchanged if the new value is invalid according to the validation rules.

Submission:

- Before you submit, save your project name as your name (i.e. first character of firstname followed by second name, e.g mmeagher for Mairead Meagher's assignment). Then zip it and submit it.

A note on the marking of the assignment:

- All methods should override and extend the overridden method, where possible.

- Serialization should use the Serializer interface we discussed in class.
- You are permitted to use any applicable code from your labs / previous assignments. However if you use anything from other sources, you must reference it.
- An indicative marking scheme is as follows:
    - Structure and Principles adopted (i.e. DX approach, UX approach) 15%
    - Phase 1 (inheritance, abstraction and polymorphism) 28%
    - Phase 2 (assessment class) 3%
    - Phase 3 (GymApi class) 29%
    - Phase 4 (MenuController class) 15%
    - Phase 5 (Testing) 10%

Phase 1 – Inheritance and Polymorphism:

Create the following inheritance hierarchy in Java that defines the following inheritance hierarchy:

- Person (abstract). Stores email, name, address and gender. The email is used to uniquely identify a person in the system.
- Member(abstract). Subclass of Person. Stores a person's height, starting weight, chosenPackage and a hashmap (key: date; value: assessment details) to record all the members progress i.e. assessments performed by trainers.
- PremiumMember (concrete). Subclass of Member. Stores no additional data.
- StudentMember (concrete). Subclass of Member. Stores studentId and collegeName.
- Trainer (concrete). Subclass of Person. Stores the trainer's speciality.

The following rules should be applied to this hierarchy:

- The following validation rules apply to these fields; any fields not listed below indicates that no validation is done on the field:
    - Height is measured in metres and must be between 1 and 3 inclusive.
    - Starting Weight is measured in kgs and must be between 35 and 250.
    - The name is maximum 30 characters; any name entered should be truncated to 30 characters.
    - The gender can be either "M" or "F". If it is not specified, apply a default value of "Unspecified".
- The Person class should have a String toString() method that formats the printing of the object state and returns it. It's subclasses should override the superclass method String toString() so that they can report on the new fields defined in these subclasses.
- Each class in the hierarchy should define a constructor that initialises each instance variable based on user input data.

Specifics for class **Person**:

- No additional functionality included, after the accessors, mutators and constructors.

Specifics for class **Member**:

- `public double calculateBMI(double weight)`
  Return the BMI for the member based on the calculation: BMI is weight divided by the square of the height (https://en.wikipedia.org/wiki/Body_mass_index)

- `public String determineBMICategory(double bmiValue)`
  Return the category the BMI belongs to, based on the following values:

```
BMI less than 15(exclusive) is "VERY SEVERELY UNDERWEIGHT"
BMI between 15 (inclusive) and 16 (exclusive) is "SEVERELY UNDERWEIGHT"
BMI between 16 (inclusive) and 18.5 (exclusive) is "UNDERWEIGHT"
BMI between 18.5 (inclusive) and 25(exclusive) is "NORMAL"
BMI between 25 (inclusive) and 30 (exclusive) is "OVERWEIGHT"
BMI between 30 (inclusive) and 35 (exclusive) is "MODERATELY OBESE"
BMI between 35 (inclusive) and 40 (exclusive) is "SEVERELY OBESE"
BMI greater than 40(inclusive) is "VERY SEVERELY OBESE"
```

- ```
  public Assessment latestAssessment() {
  //Returns the latest assessment based on last entry (by calendar date).
     return getAssessments().get(sortedAssessmentDates().last());
  }
  ```

- ```
  public SortedSet<Date> sortedAssessmentDates() {
  //Returns the assessment dates sorted in date order.
        return new TreeSet<Date>(getAssessments().keySet());
  }
  ```

Specifics for class **PremiumMember**:

- `public void chosenPackage(String packageChoice)`
  Provides the concrete implementation for this method. The chosenPackage is set to the value passed as a parameter. There is no validation on the entered data.

Specifics for class **StudentMember**:

- `public void chosenPackage(String packageChoice)`
  Provides the concrete implementation for this method. The chosenPackage is set to the package associated with their collegeName. If there is no package associated with their college, default to “Package 3”.

Specifics for class **Trainer**:

- No additional functionality included, after the accessors, mutators and constructors.

Phase 2 – Assessment:

The Assessment class is a concrete class that stores weight, chest, thigh, upperArm, waist, hips, comment and a Trainer that entered the member's assessment `(i.e. private Trainer trainer)`.

This class just has the standard constructor, accessor and mutator method with no validation on any fields. It also contains the standard toString method.

Phase 3 – Gym Api:

This is a concrete class that operates between:

- the inheritance hierarchy classes and assessment class detailed above (phase 1 and 2) and
- the menu controller (phase 4).

This class stores:

- an ArrayList of members
- an ArrayList of trainers.
- a serializer object.

It contains the following, self-explanatory methods:

- `public void addMember(Member member)`
- `public void addTrainer(Trainer trainer)`
- `public int numberOfMembers()`
- `public int numberOfTrainers()`
- `public ArrayList<Member> getMembers()`
- `public ArrayList<Trainer> getTrainers()`

The class also contains these methods:

- `public boolean isValidMemberIndex(int index)`
  Returns a boolean indicating if the index passed as a parameter is a valid index for the member's array list.

- `public boolean isValidTrainerIndex(int index)`
  Returns a boolean indicating if the index passed as a parameter is a valid index for the trainer's array list.

- `public Member searchMembersByEmail(String emailEntered)`
  Returns the member object that matches the email entered. If no member matches, return null.

- `public Person searchTrainersByEmail(String emailEntered)`
  Returns the trainer object that matches the email entered. If no trainer matches, return null.

- `public String listMembers()`
  Returns a string containing all the members details in the gym. If there are no members in the gym, return a message indicating this.

- `public String listMembersWithIdealWeight()`
  Returns a string containing all the members details in the gym whose latest assessment weight is an ideal weight (based on the devine method). If there are no members in the gym, return a message indicating this. If there are members in the gym, but none have a current ideal body weight, return a message indicating this also.

- `public String listMembersBySpecificBMICategory(String category)`
  Returns a string containing all the members details in the gym whose BMI category(based on the latest assessment weight) partially or entirely matches the entered category. If there are no members in the gym, return a message indicating this. If there are members in the gym, but none have a current BMI category matching the selected category, return a message indicating this also.

- `public String listMemberDetailsImperialAndMetric()`
  List, for each member, their latest assessment weight and their height both imperially and metrically. The format of the output is like so:

  - Joe Soap: xx kg (xxx lbs) x.x metres (xx inches).
  - Joan Soap: xx kg (xxx lbs) x.x metres (xx inches).

  If there are no members in the gym, the message "There are no members in the gym" should be returned.

- `public void load() throws Exception`
  Using the serializer object in this current class, read the associated XML file and pop the members and trainers into their associated array lists.

- `public void store() throws Exception`
  Using the serializer object in this current class, push the members and trainers array lists out to the associated XML file.

Phase 4 – Menu Controller:

Create a driver class (MenuController) that uses the console I/O to interact with the user. This driver class should create an instance of the GymApi class and allow the user to navigate the system through a series of menus.

The following processing is required in this menu system:

1. On app startup, automatically load the gym data (trainers and members) from an XML file.
2. Ask the user do they want to login(l) or register (r).
3. Ask the user if they are a member(m) or a trainer(t).
   a. If the user selected to login, verify that the email entered is stored in the appropriate arraylist i.e. the members or trainers list. If the email doesn't exist, print out "access denied" to the console and exit the program.
   b. If the user selected to register, ask the user to enter the required details for the member/trainer. If a user enters an email that is already used in the system, ask let them know it is an invalid email and ask them to enter a new one.
4. Once logged in, display a trainer menu for the trainer and a member menu for the member.

a. The trainer menu should allow the trainer to:
    i. Add a new member
    ii. List all members
    iii. List members with ideal body weight
    iv. List members with a specific BMI category
    v. Search for a member by email
    vi. Add an assessment for a member
    vii. View assessments for a member (sorted by date).
b. The member menu should all the member to:
    i. View their profile
    ii. Update their profile
    iii. View their progress

5. On app exit, automatically save the gym data (trainers and members) to an XML file.

Note: Aside from the above minimum requirements, the design of the menu system and the contents you include is left open to you. This is an important area of the assignment where you can demonstrate your programming skills. Also, robustness of this menu system is an important factor (i.e. exception handling).

Note 2: The following packages can be hard coded in this class into a HashMap:

```
("Package 1", "Allowed access anytime to gym.\nFree access to all classes.
\nAccess to all changing areas including deluxe changing rooms.");
 ("Package 2", "Allowed access anytime to gym.\n€3 fee for all classes.
\nAccess to all changing areas including deluxe changing rooms.");
 ("Package 3", "Allowed access to gym at off-peak times.\n€5 fee for all
classes.  \nNo access to deluxe changing rooms.");
 ("WIT", "Allowed access to gym during term time.\n€4 fee for all classes.
\nNo access to deluxe changing rooms.");
```

Ideally, this data would be read in from a file, however, we can just hard code them for the purposes of this assignment.

Phase 5 – Testing

A JUnit test class will be provided for some classes. Having reviewed the test classes provided, create appropriate test classes for the remaining classes (except the Menu controller class) Test methods in this class should include:

- test method for getters and setters
- test method for constructor/s.
- test methods for each other method in the class.

Ensure that you test for both valid and invalid data entry. Ensure that you use the setup() method to help you. You should aim for full code coverage here.