Assignment 3 specification – Inheritance, Polymorphism and Abstraction

This overall assignment is worth 40% of the overall module grade. 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.

The Rubric for the Assignment

Grade Range	**Hierarchy (Phase 2, 3)**	**Gym Api (Phase 4)**	**Menu/UX (Phase 1, 5)**	**Testing/DX (Phase 1, 6)**
Starter	Member	Add member/trainer NumberOfMembers/ Number of Trainers Get members/ trainers	View and update members (member menu).	JavaDoc. Coding and naming standards adhered to.
Baseline	Person	Search members by name	Member login/register (unique email address). UX (robust, intuitive, user friendly, etc).	JUnit testing of models – full code coverage <u>not</u> achieved. Package structure adhered to.
Good	Trainer	Search members & trainers by email. Uses util classes to verify valid indexes for arraylists.	Trainer register/login (unique email address). Trainer menu (top level).	JUnit testing of models – full code coverage achieved. Code is DRY i.e not repeating code unnecessarily; use of utils defined in labs / lectures.
Excellent	Assessment	Load/store members & trainers in XML. Uses latest assessment weight when calculating analytics i.e. BMI, BMI Category, Ideal Body Weight.	Assessment Menu. Progress menu.	Junit testing of GymAPI – full code coverage <u>not</u> achieved.
Outstanding	Premium & Student members	Uses util classes for analytics methods (BMI, BMI category, ideal body weight), for conversion methods (kg->lb, m->in), etc.	Verification of valid email address format. Reports menu. Load / Store implemented.	Junit testing of GymAPI – full code coverage achieved. Use of your own util classes.

A note on the marking of the assignment:

- Minimal marks are going for constructors, accessors, and mutators as these can be generated in Eclipse. We also examined them heavily in Assignment 1.
- All methods should override and extend the overridden method, where possible.
- Serialization should use the Xstream component we discussed in class.
- You are permitted to use any applicable code from the Assignment 1 solution and any of our lab solutions. You are also permitted to use any applicable code from your Web assignment.
- Self-directed learning:
    - This assignment will require some self-directed learning on your part. You should use the approaches and materials discussed in class, however, for some of the requirements, you will probably need to do some research yourself.

Stack overflow is a very good resource for solving technical problems.

Phase 1: Structure and Principles

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:

- Where practicable, you should aim to write utility methods/classes located in the utils package for standard calculations, reading in valid data, etc.
- Java doc all solutions.
- Java naming and coding standards should be applied throughout the project.
- Code should follow the DRY (Don't Repeat Yourself) principle, where practicable.
- Your project should be structured using packages and source directories i.e.
    - The **src** folder should contain three packages:
        - controllers (for MenuController and GymApi classes).
        - models (for the inheritance hierarchy classes and the assessment class).
        - utils (for the utility classes – see below for more information).

    - The **test** folder's packages should be structured in the same manner as the **src** folder. A test class for each class identified above should be placed in the corresponding package in this folder.

The utils package:

- The ScannerInput class (used in labs) would be stored in here.

- Write an Analytics class that would hold the following methods:
    - `public static double calculateBMI(Member member, Assessment assessment)`

Return the BMI for the member based on the calculation: BMI is weight divided by the square of the height.

- `public static 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 static boolean isIdealBodyWeight(Member member, Assessment assessment)`
  Returns a boolean to indicate if the member has an ideal body weight based on the Devine formula:
  * For males, an ideal body weight is: 50 kg + 2.3 kg for each inch over 5 feet.
  * For females, an ideal body weight is: 45.5 kg + 2.3 kg for each inch over 5 feet.
  * Note: if no gender is specified, return the result of the female calculation.
  * Note: if the member is 5 feet or less, return 50kg for male and 45.5kg for female.

Phase 2 – Inheritance, Abstraction and Polymorphism:

Create these classes in Java to define 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 (concerete). 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(), call the superclass toString method and also report on the new fields defined in these subclasses.
- Each class in the hierarchy should define a constructor that initialises each instance fields based on user input data.
- Each class in the hierarchy should define accessors and mutators for all instance fields.

Specifics for class **Person**:

- No additional functionality included; just include the accessors, mutators and constructors.

Specifics for class **Member**:

- `public Assessment latestAssessment()`
  Returns the latest assessment based on last entry (by calendar date).

- `public SortedSet<Date> sortedAssessmentDates()`
  Returns the assessments dates sorted in date order. Note: you can choose to sort the assessment dates in any manner you wish; you do not have to stick to the SortedSet approach.

- `public abstract void chosenPackage(String chosenPackage);`
  The concrete implementation of this method will be completed in Member subclasses.

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; just include the accessors, mutators and constructors.

Phase 3 – Assessment class:

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 4 – GymApi class:

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.

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 String searchMembersByName(String nameEntered)`
  Returns a list of members whose name partially or entirely matches the entered name. If there are no members in the gym, return a message indicating this. If there are members in the gym, but none match the name entered, return a message indicating this also.

- `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`
  Read the associated XML file and pop the members and trainers into their associated array lists.

- `public void store() throws Exception`
  Push the members and trainers array lists out to the associated XML file.

Phase 5 – MenuController class:

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 cl ass 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 start-up, 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 (for

either trainers/members), 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. Search for a member by email
      iv. Search for a member by name
      v. List members with ideal body weight
      vi. List members with a specific BMI category
      vii. Assessment sub-menu
         1. Add an assessment for a member
         2. Update comment on an assessment for a member
      viii. Reports sub-menu
         1. Specific member progress (via email search). Note: brings the user to the same progress sub-menu identified below
         2. Specific member progress (via name search). Note: brings the user to the same progress sub-menu identified below
         3. Overall members' report

   b. The member menu should allow the member to:
      i. View their profile
      ii. Update their profile
      iii. Progress sub-menu
         1. View progress by weight
         2. View progress by chest measurement
         3. View progress by thigh measurement
         4. View progress by upper arm measurement
         5. View progress by waist measurement
         6. View progress by hips measurement
5. On app exit, automatically save the gym data (trainers and members) to an XML file.

Note 1: Aside from the above 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 (see extra credit phase also).

Note 3: The options on the progress sub-menu should ideally display the member's progress based on the metric chosen (e.g. weight). It should display the metric sorted by date and some form of indication as to whether the subsequent measurement was above or below the previous measurement.

Phase 6 – Testing

Provide JUnit testing classes for the classes in the models and controllers packages (except the Menu controller class). Test methods 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. You should aim for full code coverage here.

Extra Credit

Extra credit of up to 10% can be added onto your grade for this assignment (bringing your grade to a max of 100%). Extra credit would be awarded in cases where the assignment went above and beyond the assignment specified above. Some examples of extra credit opportunities include (but are not limited to):

- Persist (i.e. save) in both JSON and XML format.
- Instead of hard-coding the gym packages into the MenuController class, read them in from a file in a format e.g. XML, JSON, etc.
- The use of Enums (stored in the utils folder) e.g. for defining BMI categories:
  http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html.
  http://javarevisited.blogspot.ie/2011/08/enum-in-java-example-tutorial.html
- JUnit testing of the classes/methods in the utils package. Once again, aim for full code coverage in this area.