Encapsulation Exercises

The purpose of this exercise is to provide you the opportunity to practice writing code that's extendable, maintainable, and loosely coupled through the art of encapsulation.

Learning objectives

After completing this exercise, students will understand:

- How to write code that's loosely coupled.
- How to write code that appropriately hides the internal details of classes.
- How to create readonly properties.
- How to limit access to properties through the use of access modifiers.

Evaluation criteria and functional requirements

- The project must not have any build errors.
- Unit tests pass as expected.
- Appropriate variable names and data types are used.
- Code is presented in a clean, organized format.
- Code is appropriately encapsulated.
- The code meets the specifications defined below.

Getting started

- Import the `oop-with-encapsulation-exercises` project into Eclipse.
- Right-click on the project, and select the **Run As -> JUnit Test** menu option.
- Click on the **JUnit** tab to see the results of your tests and which passed/failed.
- Provide enough code until the test passes.
- Repeat until all tests are passing.

Step One: Implement the `HomeworkAssignment` class

Data members

Attribute	Data Type	Get	Set	Description
earnedMarks	int	X	X	The total number of correct marks submitter received on the assignment.
possibleMarks	int	X		The number of possible marks on the assignment.
submitterName	string	X	X	The submitter's name for the assignment.
letterGrade *(derived)*	string	X		The letter grade for the assignment.

Notes

- `letterGrade` is a derived attribute that's calculated using `earnedMarks` and `possibleMarks`.
  - For 90% or greater, return "A"
  - For 80-89%, return "B"
  - For 70-79%, return "C"
  - For 60-69%, return "D"
  - Otherwise, return "F"
  - *hint*: `possibleMarks` and `earnedMarks` are integers. What happens when a smaller integer is divided by a larger integer?

Constructor

The `HomeworkAssignment` class has a single constructor. It accepts a single argument: `possibleMarks`.

```
`public HomeworkAssignment(int possibleMarks)`
```

Step two: Implement the `FruitTree` class

Data members

Attribute	Data Type	Get	Set	Description
typeOfFruit	string	X		The type of fruit on the tree.
piecesOfFruitLeft	int	X		The number of remaining fruit pieces on the tree.

Methods

```
`public bool pickFruit(int numberOfPiecesToRemove)`
```

Notes

- `pickFruit(int numberOfPiecesToRemove)` is a method.
- If there are enough pieces on the tree, "picks" the fruit and updates `piecesOfFruitLeft` by subtracting `numberOfPiecesToRemove` from it.
- Returns `true` if successful (there were enough pieces to pick) or `false` if no fruit was picked (`piecesOfFruitLeft` was less than `numberOfPiecesToRemove`).

Constructor

The `FruitTree` class has a single constructor. It accepts two arguments: `typeOfFruit` and `startingPiecesOfFruit`.

```
`public FruitTree(string typeOfFruit, int startingPiecesOfFruit)`
```

Step three: Implement the `Employee` class

Data members

Attribute	Data Type	Get	Set	Description
employeeId	int	X		The employee ID.
firstName	string	X		The employee's first name.
lastName	string	X	X	The employee's last name.
fullName *(derived)*	string	X		The employee's full name.
department	string	X	X	The employee's department.
annualSalary	double	X		The employee's annual salary.

Notes

- `fullName` is a derived attribute that returns `lastName, firstName`.

Methods

```
`public void raiseSalary(double percent)`
```

Notes

- `raiseSalary(double percent)` increases the current annual salary by the percentage provided.

Constructor

The `Employee` class has a single constructor. It accepts four arguments:

`public Employee(int employeeId, String firstName, String lastName, double salary)`

Step four: Implement the `Airplane` class

Data members

Attribute	Data Type	Get	Set	Description
planeNumber	string	X		The six-character plane number.
bookedFirstClassSeats	int	X		The number of already booked first class seats.
availableFirstClassSeats *(derived)*	int	X		The number of available first class seats.

Attribute	Data Type	Get	Set	Description
totalFirstClassSeats	int	X		The total number of first class seats.
bookedCoachSeats	int	X		The number of already booked coach seats.
availableCoachSeats *(derived)*	int	X		The number of available coach seats.
totalCoachSeats	int	X		The total number of coach seats.

Notes

- `availableFirstClassSeats` is a derived attribute calculated by subtracting `bookedFirstClassSeats` from `totalFirstClassSeats`
- `availableCoachSeats` is a derived attribute calculated by subtracting `bookedCoachSeats` from `totalCoachSeats`

Constructors

The `Airplane` class has a single constructor. It accepts three arguments:

```
`Airplane(String planeNumber, int totalFirstClassSeats, int
totalCoachSeats)`
```

- `planeNumber` is the six-character plane number.
- `totalFirstClassSeats` is the initial number of total first class seats.
- `totalCoachSeats` is the initial number of total coach seats.

Methods

```
`bool reserveSeats(bool forFirstClass, int totalNumberOfSeats)`
```

Notes

- `reserveSeats()` is a method.
  - if `forFirstClass` is `true`, reserve the value for `totalNumberOfSeats` for first class.
  - if `forFirstClass` is `false`, reserve the value for `totalNumberOfSeats` for coach.
  - return `true` if the reservation can be made, `false` if it can't.

Step five: Implement the `Television` class

Data members

Attribute	Data Type	Get	Set	Description

Attribute	Data Type	Get	Set	Description
isOn	boolean	X		Whether or not the TV is turned on.
currentChannel	int	X		The value for the current channel. Channel levels go between 3 and 18.
currentVolume	int	X		The current volume level.

Constructors

The `Television` class doesn't need a constructor. It can use the **default constructor**.

A new TV is off by default. The channel is set to three and the volume level to two.

Methods

```
void turnOff()
void turnOn()
void changeChannel(int newChannel)
void channelUp()
void channelDown()
void raiseVolume()
void lowerVolume()
```

Notes

- `turnOff()` turns off the TV.
- `turnOn()` turns the TV on and also resets the channel to three and the volume level to twos.
- `changeChannel(int newChannel)` changes the current channel—only if it's on—to the value of `newChannel` as long as it's between 3 and 18.
- `channelUp()` increases the current channel by one, only if it's on. If the value goes past 18, then the current channel should be set to three.
- `channelDown()` decreases the current channel by one, only if it's on. If the value goes below three, then the current channel should be set to 18.
- `raiseVolume()` increases the volume by one, only if it's on. The limit is 10.
- `lowerVolume()` decreases the volume by one, only if it's on. The limit is zero.

Step six: Implement the `Elevator` class

Data members

Attribute	Data Type	Get	Set	Description
currentFloor	int	X		The current floor that the elevator is on.
numberOfFloors	int	X		The number of floors available to the elevator.

Attribute	Data Type	Get	Set	Description
doorOpen	boolean	X		Whether the elevator door is open or not.

Constructor

The `Elevator` class has a single constructor that takes one argument. New elevators start on floor one.

```
`Elevator(int totalNumberOfFloors)`
```

- `totalNumberOfFloors` indicates how many floors are available to the elevator.

Methods

```
    void openDoor()
    void closeDoor()
    void goUp(int desiredFloor)
    void goDown(int desiredFloor)
```

Notes

- `openDoor()` opens the elevator door.
- `closeDoor()` closes the elevator door.
- `goUp(int desiredFloor)` sends the elevator upward to the desired floor as long as the door isn't open. Can't go past last floor.
- `goDown(int desiredFloor)` sends the elevator downward to the desired floor as long as the door isn't open. Can't go past floor one.

Tips and tricks

- **Note: If you find yourself stuck on a problem for longer than fifteen minutes, move on to the next one, and try again later.**
- In this exercise, you'll create the classes specified in the requirements section of this document. The unit tests you run verify if you've defined the classes correctly.
- As you work on creating the classes, be sure to run the tests, and then provide enough code to pass the test. For instance, if you're working on the `HomeworkAssignment` class, provide enough code to get one of the `HomeworkAssignment` tests passing. Focusing on getting a single test to pass at a time saves time, as this forces you to only focus on what's important for the test you are currently working on. This is commonly called **Test Driven Development**, or **TDD**.
- Be mindful of your access modifiers.
- Keep in mind, a **loosely coupled** system is one in which each of its components has, or makes use of, little or no knowledge of the definitions of other separate components. One of your goals as a developer should be to write code that's loosely coupled.
