# Encapsulation

In this exercise, you'll create the classes specified in the Exercises section of this document. The unit tests you run verify that you defined the classes correctly.

## Learning objectives

After completing this exercise, you'll understand how to:

- Write code that's loosely coupled.
- Write code that appropriately hides the internal details of classes.
- Limit access to properties through the use of access modifiers.
- Write derived properties.

## Evaluation criteria and functional requirements

- The project must not have any build errors.
- All 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 or failed.
- Provide enough code until the test passes.
- Repeat until all tests are passing.

## Tips and tricks

Focus on one test at a time

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're currently working on. This is commonly called **Test Driven Development**, or **TDD**.

Be mindful of your access modifiers

Remember that access modifiers are a key feature of encapsulation.

Write loosely coupled code

Keep in mind that 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.

### Don't linger too long on one problem

If you find yourself stuck on a problem more than fifteen minutes, move on to the next, and try again later. You may figure out the solution after working through another problem or two.

### Notes for all classes

• An X in the set column indicates the property must have a set.

### **Exercises**

Step One: Implement the HomeworkAssignment class

#### **Instance variables**

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

#### **Notes**

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

#### Constructor

The HomeworkAssignment class has a single constructor. It accepts two arguments: int possibleMarks and String submitterName. Use these parameters to set the instance variables of the class.

Step Two: Implement the FruitTree class

#### **Instance variables**

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

#### Constructor

Create a constructor for this class that accepts two parameters: String typeOfFruit and int startingPiecesOfFruit. Use these parameters to set the instance variables of the class.

#### **Methods**

Create a method called pickFruit that accepts an int called numberOfPiecesToRemove and returns a boolean.

- If there are enough pieces left on the tree, it "picks" the fruit and updates piecesOfFruitLeft by subtracting numberOfPiecesToRemove from it.
- The method returns true if there were enough pieces left to pick. It returns false if no fruit was picked —that is, piecesOfFruitLeft was less than numberOfPiecesToRemove.

Step Three: Implement the Employee class

#### **Instance variables**

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

#### **Notes**

• fullName is a derived attribute that returns lastName, firstName.

#### Constructor

Create a constructor for this class that accepts four parameters: int employeeId, String firstName, String lastName, and double salary. Use these parameters to set the instance variables of the class.

#### **Methods**

Create a method called raiseSalary that accepts a double called percent and returns void. The method increases the current annual salary by the percentage provided. For example, 5.5 represents 5.5%.

Step Four: Implement the Airplane class

#### **Instance variables**

Attribute	Data Type	Get	Set	Description
planeNumber	string	Х		The six-character plane number.
totalFirstClassSeats	int	Х		The total number of first class seats.
bookedFirstClassSeats	int	Х		The number of already booked first class seats.
availableFirstClassSeats (derived)	int	Х		The number of available first class seats.
totalCoachSeats	int	Х		The total number of coach seats.
bookedCoachSeats	int	Х		The number of already booked coach seats.
availableCoachSeats (derived)	int	Х		The number of available coach seats.

#### **Notes**

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

#### **Constructors**

Create a constructor for this class that accepts three parameters: String planeNumber, int totalFirstClassSeats, and int totalCoachSeats. Use these parameters to set the properties of the class:

- planeNumber is the plane number assigned to the airplane.
- totalFirstClassSeats is the initial number of total first class seats.
- totalCoachSeats is the initial number of total coach seats.

#### **Methods**

Create a method called reserveSeats that returns a boolean and accepts two parameters: a boolean called forFirstClass and an int called totalNumberOfSeats.

- If forFirstClass is true, add totalNumberOfSeats to the value for BookedFirstClassSeats.
- If forFirstClass is false, add totalNumberOfSeats to the value for BookedCoachSeats.
- It returns true if there were enough seats to make the reservation, otherwise it returns false.

Step Five: Implement the Television class

#### **Instance variables**

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

#### **Constructors**

The Television class doesn't need a constructor. However, the instance variables need default values: a new TV is off by default. The channel is set to three and the volume level to two.

#### Methods

Create methods based on the following signatures:

```
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 two.
- 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

#### **Instance variables**

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

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

#### Constructor

The Elevator class has a single constructor that accepts one parameter, int numberOfLevels, which indicates how many floors are available to the elevator.

Either provide a default value or set it in the constructor so new elevators start on floor one.

#### **Methods**

Create methods based on the following signatures:

```
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. The elevator 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. It can't go past floor one.