**Problem 1**

Write a class called *Animal* with an default sound() method that does not output a specific animal sound. Instead, the generic output should be the statement, *An animal makes a sound based on the animal that it is*. The default constructor for Animal should also print *I am an animal* to standard output.

Create child classes for the following animals:

* pig
* sheep
* duck
* cow

The default constructor should print out *I am a xxx*, where *xxx* is the name of the animal. Create a sound() method for each new child class, which overrides the default method from the Animal class. The child class sound() method will print a different sound (word descriptor) for each animal, based on the child class:

* pig says “oink”
* sheep says “baah”
* duck says “quack”
* cow says “moo”

This sound() method does not generate an actual sound; it only prints the word descriptor to standard output.

Write a main() that creates several instances of each animal, and then calls each animal instance’s sound() method, so that different animals print different sounds accordingly.

Create another class called *AnimalTest* that reads the command line user input and creates the appropriate Animal child class based on the type of animal, and then calls the animal’s sound() method. Take some time to reflect on which class to create.

**Problem 2**

Define a class called *Book*. The Book class should store attributes, such as the title, ISBN number, author, edition, publisher, and year of publication. These attributes must be private.

Provide public get/set methods in this class to access these attributes. Test your Book class by creating several books and displaying the attributes.

**Problem 3**

Create a class called *Elevator* that can be moved between floors in an *N*-storey building. Elevator uses a constructor to initialize the number of floors (*N*) in the building when the object is instantiated.

Elevator also has a default constructor that creates a five- (5) storey building.

The Elevator class has a termination condition that requires the elevator to be moved to the main (i.e., first) floor when the object is cleaned up. Write a finalize() method that satisfies this termination condition and verifies the condition by printing a message to the output, *Elevator ending: elevator returned to the first floor*.

In main(), test at least five (5) possible scenarios that can occur when Elevator is used in a building with many floors (e.g., create, move from one floor to another, etc.).

**Hint**: Termination occurs when the instance is set to null. You may wish to investigate “garbage collection” for this exercise.

**Problem 4**

Create an inheritance hierarchy of *Rodent*: mouse, gerbil, hamster, guinea pig.

In the base class, provide methods that are common to all rodents based on behaviours you find with a quick Internet search. Be sure to document the behaviours you implement (e.g., eat, sleep, groom, move, etc.). Each behaviour should print its action to standard output (e.g., *rodent eating*).

Next, refine these behaviours in the child classes to perform different behaviours, depending on the specific type of rodent, but only if the behaviour is actually different (e.g., *mouse eating seeds* or *guinea pig eating grass*). If you are having difficulty coming up with various behaviours, create a discussion on the Landing to compare notes with others taking the course.

Test your Rodent classes by writing a main() class and creating instances of every rodent, and demonstrate all the behaviours for each rodent.

**Problem 5**

1. Create a *Point* class to hold *x* and *y* values for a point. Create methods show(), add() and subtract() to display the Point *x* and *y* values, and add and subtract point coordinates.

**Hint**: Keep *x* and *y* separate in the calculation.

Create another class *Shape*, which will form the basis of a set of shapes. The Shape class will contain default functions to calculate area and circumference of the shape, and provide the coordinates (Points) of a rectangle that encloses the shape (a bounding box). These will be overloaded by the derived classes; therefore, the default methods for Shape will only print a simple message to standard output.

Create a display() function for Shape, which will display the name of the class *and* all stored information about the class (including area, circumference and bounding box).

Build the hierarchy by creating the Shape classes *Circle*, *Rectangle* and *Triangle*. Search the Internet for the rules governing these shapes, if necessary.

For these three Shape classes, create default constructors, as well as constructors whose arguments will initialize the shapes appropriately using the correct number of Point objects (i.e., Circle requires a Point center and a radius; Rectangle requires four Point vertices; and Triangle requires three Point vertices). Add error-checking to the constructors, such that they print a warning if the arguments do not conform to the appropriate shape.

**Hint**: Not all four-sided shapes are rectangles.

Also add a check to Rectangle, such that it tests for the special case of a “square” and prints an appropriate message if the test is true.

In main(), create several instances of each shape object and display the information for each object. Be sure to create at least one non-rectangle shape to demonstrate your error handling, and at least one square.