**Object Oriented Programming: Problem Set 2. Writing instantiable classes. This Lab sheet may take two weeks to complete.**

1. Write a class definition for a **Book**. Please note that this class must be called **Book.java**.

A Book should have the following 4 **attributes**: *title*, *price*, *ISBN* and *number of pages*. (The ISBN is a unique identifier for a book: it can contain letters as well as numbers and is either 10 chars or 13 chars long).

Book should have **accessor** and **mutator** methods defined for each of its attributes as

well as a **toString**() method (which accesses the attributes **directly** i.e. not via accessors). It

should have a **no-argument constructor** which sets up the attributes **directly** (i.e. not via another

constructor or via calls to the set() methods) to give book objects the following initial state: [“No

Title”, 0.00, “No ISBN”, 0]. It should also have a second **constructor which takes 4 arguments**

representing the 4 attributes and **calls the mutator methods directly**.

You should code your class for **true encapsulation** and write a **minimalistic driver program** that will test all the functionality of your class.

Once you are satisfied that your minimalistic driver program is working, write a second driver program which declares and creates a number of book objects using details supplied by the user (ask the user to enter title, price etc of the user’s favourite book and least favourite book into dialogs). Get all the data into the objects, then display the details of all objects in one output dialog. Explore the different ways of doing this: using toString(), or using the individual ‘get’ methods and printf() to get control over layout, or using a JTextArea within a message dialog. You can improve your code by including some input validation that you were taught last year.

2.Design a class Message that models an e-mail message. A message has a recipient, a sender, and a message text. Support the following methods:

* A constructor that takes the sender and recipient
* A method *append* that appends a line of text to the message body
* A method *tostring* that makes the message into one long string like this: “From: James O’Donoghue\nTo: Gouch Cooper\n……..”

Write a program that uses this class to make a message and display it.

3. Another instantiable class. Write a class definition for an **Animal**. An animal should

have the following 4 **attributes**: *type*, *continents* (an **array** of String), *weight*, and *age*.

It should have **accessor** and **mutator** methods defined for each of its attributes as well as

a **toString**() method (which accesses the attributes **indirectly** via accessor methods).

It should also have a **no-argument constructor** which sets up the attributes **indirectly**

(via a call to a 4-argument constructor) to give animal objects the following initial state:

[“No Type Specified”, null, 0.0, 0]. It should also have a second

**constructor which takes 4 arguments** representing the 4 attributes and **calls the**

**mutator methods directly**. You should code your class for **true encapsulation** and

write a **minimalistic driver program** that will test all the functionality of your class.

Your **code need not handle invalid data** values in

any way.

**Hints**: This is a standard class definition except for the presence of the **array** which

complicates things just a little. The getContinents() method will return the

**continents** attribute - an array of String (which is **an object** in its own right – recall

that an array is an object, created using the **new** operator), allowing you to “query” each

element (continent) in that array via **subscript numbers**. If the array is empty then its value

will be “null” to indicate this (as in the case for the no-argument constructor call). This fact

will be used in toString() for displaying purposes. As you might expect, you’ll be (or

should be) making use of a **for-loop** when processing the array in the toString() method.

In the driver program you will need to create an array of String object, representing the

Conti

nents, to test out the 4-argument constructor.

4. And yet another instantiable class. Write a class definition MyPoint.java for a **MyPoint**

class to model points on an x-y plane. A MyPoint will have the following attributes:

xVal, yVal. It should have **accessor** and **mutator** methods defined for each of its attributes

as well as a **toString**() method (which only accesses the attributes **indirectly**). It should have a

**no-argument constructor** which sets up the attributes **directly** (i.e. not via another constructor

or via calls to the set() methods) to give MyPoint objects the following initial state: (0,0). It

should have a **second constructor** that takes **2 arguments** representing the 2 attributes and

should **call the mutator methods** to set the values.

Methods must be written which can moveHorizontally() or moveVertically() a point

through a supplied number of units, also a method which will translate() a point a specific

distance in x- and y- directions and a method which will calculate the distanceFromOrigin()

of the point using Pythagoras’ formula (the origin is at 0,0). You should code your class for true

encapsulation and write a driver program that will test all the functionality of your class.

5. Write a class definition for a class called **Bicycle.java** to model a Bicycle. It should have the following attributes: owner’s name, value, and make,with accessors and mutators for each. It should have a no-args constructor, one which takes values for all the attributes and a toString() method. Write a driver which

* Asks the user to enter a name, value and make and creates a Bicycle with these attributes
* Declares and creates a Bicycle using the no-arguments constructor, then asks the user to

enter a value for each of the attributes of this second Bicycle and sets them accordingly

* Increases the value of the first Bicycle by 10 euro by getting the value, then setting it to

an increased value

* Outputs the owner names and values of the 2 Bicycles using the accessor methods.
* Calculates and displays the total value of the 2 Bicycles

6. Write a class definition for a Fraction (nb this class must be called Fraction.java) to facilitate

arithmetic with fractions. A fractional number will have the following form:

**numerator/denominator**

Your class should define the 2 attributes above to be capable of fractional arithmetic i.e.

numerator and denominator must be positive or negative whole numbers. It should have accessor

and mutator methods defined for each of its attributes as well as a toString() method (which only

accesses the attributes directly) that displays the fraction in the form indicated above. It should

have a no-argument constructor which sets up the attributes indirectly (via its 2-argument

constructor) to give Fraction objects the following initial state: [0/1]. It should have a second

constructor that takes 2 arguments representing the 2 attributes and calls the mutator methods to

set the values. Methods must also be written which can respectively add, subtract, multiply and

divide 2 fractions. Work out on paper the relatively simple algorithms for how these methods

need to be written to achieve their tasks. Note also that these methods will take a single Fraction

as argument and return a Fraction as a result. You should code your class for true encapsulation

and write a minimalistic driver program that will test all the functionality of your class. Your

code need not handle invalid data values in any way. A possible run of your driver could be as

follows:



7. Select something which interests you to form the basis for a

small project, and write a class to model it, along the lines of Bicycle above. For example,

Games students could decide between them to model things like Warriors, Vehicles, Weapons

etc, each student taking a different class. Your class should model one entity only.

8. Using Dates and GregorianCalendar objects. Copy the sample programs DateTest1 to 4

into your own folder, examine them, then compile and run them. Have a look at the

documentation of the Date class, the SimpleDateFormat class, the GregorianCalendar

class and the Calendar class. Change the birthday date in DateTest4 to your own, and run

the program again.

9. Write a program which asks you to enter day, month and year for two dates, called

borrowDate and returnDate, corresponding to the date you borrowed a textbook from the

library and the date you returned it (both should be within the same calendar year, and the

second one should come after the first one): your program should set up two

GregorianCalendar objects corresponding to the two dates. Using these two objects you must write additional code to calculate the number of days the book was on loan from the library. One way of doing this is to use the ‘get’ method and DAY\_OF\_YEAR field (inherited by the GregorianCalendar class from the Calendar class) to find out how many days in total you have kept the book.

10. In this lab, you will implement a *vending machine* that holds cans of soda. To buy a can of soda, the customer needs to insert a token into the machine. When the token is inserted, a can drops from the can reservoir into the product delivery slot. The vending machine can be filled with more cans. The goal is to determine how many cans and tokens are in the machine at any given time.

What methods would you supply for a VendingMachine class? Describe them informally.

Now translate those informal descriptions into Java method headers

What instance variables do the methods need to do their work? *Hint:* You need to track the number of cans and tokens.

Consider what happens when a user inserts a token into the vending machine. The number of tokens is increased, and the number of cans is decreased. Be sure to check that the number of cans is greater than zero before decreasing the can count and increasing the token count.

Now implement a method fillUp(int cans) to add more cans to the machine. Simply add the number of new cans to the can count.

Next, implement two methods, getCanCount and getTokenCount, that return the current values of the can and token counts.

So far, the VendingMachine class does not have any constructors. Instances of a class with no constructor are always constructed with all instance variables set to zero (or null if they are object references). It is always a good idea to provide an explicit constructor.

After you have completed the vending machine class, write a tester class.