COMP1006/1406 – Summer 2016

Submit a single file called `assignment2.zip` to cuLearn.
There are 100 marks possible marks. The assignment it out of 100.

In assignment specifications, text appearing `like this` represents things that a user types when running or using a program. Text that appears `like this` represents output from your program.

1: Written [10 marks]

Write your solutions to both questions (A and B) in a plain text file called `Problem1.txt` or PDF file called `Problem1.pdf`. Do NOT submit a .doc, .docx, .rtf or any other kind of file.

A) You are asked to design a `Student` class that models students at Carleton University. What attributes would your class have? What methods would your class have?

B) Write a good multiple choice question based on what we have learned so far in the course. The question should have three answers: one best answer and two that are feasible but not the best (or possibly incorrect). Be sure to say which answer is the correct (best) answer for the question and why.

Here is an example (do not submit this as your question).

```
In the HelloWorld program, what is "public" that
appears on the line that defines the main method?

 a) An access modifier
 b) A member level access modifier
 c) A non-access modifier


 The best answer is b).
 a) is also correct, but b) is better as it is more specific
 c) is incorrect.
```

Some questions that are submitted may be used later in the course (in a quiz, midterm or final). If you do not want your question to be used simply add a line at the end of the question stating that you wish your question to remain private.

Include your `Problem1.txt` or `Problem1.pdf` file in your `assignment2.zip`.

2: People [20 marks]

The `Person` class models a person. Each person will have a name (consisting of a first name, middle name, and a surname), an age and a height.

➨ Complete the `Person` class provided. Your class must have the following methods and constructors: (specifications are given in the skeleton code provided)

```
public Person(String, String, String, int, int);
public Person(String, String, int, int);
public Person();

public String getName();
public int    getHeight();
public int[]  getHeightFeetAndInches();
public int    getAge();

public void   setFirstName(String);
public void   setMiddleName(String);
public void   setLastName(String);
public void   setHeight(int);
public void   setAge(int);

public String toString();

public static void main(String[]);
```

There should be no other `public` attributes or methods in your class. You may add any `private` attributes or methods that you see fit. Be sure to follow the specifications for each method (and constructor) as given in the skeleton code provided.

The conversion from cm to feet and inches is not exact. It should be the best conversion possible though. If your conversion of h cm is f feet and i inches, where $0 \le f$ and $0 \le i < 12$, then f and i when converted back to cm is the closest to h for any values of feet (≥ 0) and inches (between 0 and 11). In particular, converting f feet and $i \pm 1$ inches will give a cm value further away from the original h than converting f feet and i inches will.

Note: Pay special attention to the specification of the output of all methods. In particular, the output of the toString and getName methods must follow the specifications exactly. A misplaced comma or missing/extra space, for example, will result in significant marks lost.

Mark breakdown: 5 marks for exact output, 15 marks for Correctness

Put your `Person.java` file in your `assignment2.zip` file.

Examples:

`Person p = new Person("CHARLES", "erNEST", "Gooda", 21, 234);`

`p.toString()` $\xrightarrow{\text{returns}}$ `"GOODA, Charles Ernest: 21 years old, 2.34 metres tall."`

`p.getName()` $\xrightarrow{\text{returns}}$ `"Charles E. Gooda"`

`p.getHeightFeetAndInches()` $\xrightarrow{\text{returns}}$ `[7,8]`

Note: the quotation marks in the expected output for strings is not part of the actual output.

3: Fish [30 marks]

In the problem you will implement a `Fish` class that will represent fish in a video game.

A skeleton (partial version) of the class is given below. You will need to implement all non-getter methods and constructors. Full descriptions are given in the provided `Fish.java` file. You may add any private attributes or methods you see fit.

```
public class Fish{
  /* constants for fish bowl and fish parameters (DO NO CHANGE THESE) */
  public static final int WIDTH = 600;
  public static final int HEIGHT = 400;
  public static final int MAX_SPEED = 5;
  public static final int MAX_SIZE = 30;

  /* fish attributes */
  private int id;         // unique id for each fish
  private int size;       // 1 <= size <= MAX_SIZE
  private int health;

  private double x, y;    // x,y coordinates of the fish
  private double dx, dy;  // x,y speed values for the fish

  /* constructors (you implement these) */
  public Fish(int size, int health)
  public Fish(int size, int health, double x, double y)
  public Fish(int size, int health, double x, double y, double dx, double dy)

  /* provided getters */
  public int getID()
  public int getSize()
  public int getHealth()
  public double getX()
  public double getY()
  public double getDX()
  public double getDY()

  /* methods (you implement these) */
  public boolean eat(Fish other)
  public Fish mate(Fish other)
  public void swim()
  public boolean closeEnough(Fish other)


  /* provided helper methods */
  public boolean hasMated(Fish mater)
  public boolean beenEaten(Fish eater)
  public String toString()
}
```

In the `closeEnough` method, you will need to determine if two fish are *close enough* to each other. Two fish will be close enough if the distance between them (based on their x and y coordinates) is less than or equal the sum of their sizes. Recall that the distance d

between point (x_1, y_1) and point (x_2, y_2) is given by

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

For random values, use the `Math.random`. It generates a random number in the range $[0.0, 1.0)$. If you want a random value between 0 and 35, you can use `Math.random()*35`. Alternatively, you can use the `Random` class.

For this problem, you do not have to consider the size of the fish when hitting a wall. Just use the x, y coordinates (which we assume is the centre of the fish).

Mark breakdown: 30 marks for Correctness

Put your completed `Fish.java` file in your `assignment2.zip` file.

4: Cash Register [40 marks]

In this problem you will complete the provided `CashRegister` class. This class models a simplified cash register. Our cash register will only hold loonies, five dollar bills, ten dollar bills, twenty dollar bills, and fifty dollar bills. The functionality (behaviour) of our cash register is that it processes transactions (purchasing and returning items) and tries to ensure it has adequate money in it.

An `Item` class is provided for this problem. An item is something that is bought (and possibly returned). It has a name and a price. All prices are in dollars. An item object is immutable so once it is created it cannot be changed.

A `Money` class is provided for this problem. This is one way in which money is represented in this problem. Money objects are also immutable. Once a money object is created there is no way to change its state (you need to create a new object if you want to change it). Another way money can be represented in this problem is to use an array of integers. The array will have five (5) elements and correspond to the number of loonies, five dollar bills, ten dollar bills, twenty dollar bill and fifty dollar bills, in that order. So, $[3, 1, 0, 2, 0]$ would represent \$48.

➨ Complete the following constructors and methods in the CashRegister class. Full descriptions are given in the comments in the provided `CashRegister.java` file. You are free to add any private attributes and helper methods as you see fit.

```
/* constructors  */
public CashRegister()
public CashRegister(Money money)
public CashRegister(int[] money)
public CashRegister(int n1, int n5, int n10, int n20, int n50)

/* getters */
public int   get1()
public int   get5()
public int   get10()
public int   get20()
public int   get50()
public int   getTotalValue()
public int[] getAll()
public Money getMoney()

/* methods */
public Money purchaseItem(Item item, Money payment)
public Money returnItem(Item item)
public CashRegister updateMoney()
public CashRegister allLoonies()
```

When a purchase is being processed, you first add the payment to the register's total money and make change based on this combined amount of money. If you are unable to give exact change for the purchase, you will call the `updateMoney` method. The `updateMoney` method allows you exchange all the current money in the register with an equivalent amount (but different combination of loonies and bills). You decide how updateMoney works. If the register is still unable to make exact change after calling updateMoney, then `allLoonies` is called which exchanges all money in the register for loonies. For example, if money in the register is currently $[2, 3, 5, 1, 1]$, then after calling `allLoonies`, the money is $[137, 0, 0, 0, 0]$.

Examples:

`Item gum = new Item("Gum", 17);` (something to buy/return, expensive gum!)

`CashRegister cash = new CashRegister();` (cash register with no money)

`cash.get20()` $\xrightarrow{\text{returns}}$ `0`. (should have no twenty dollar bills)

`cash.purchaseItem(gum, new Money(3,1,1,0,1))` $\xrightarrow{\text{returns}}$ `money object with $51`. (received $68, gave $51 back)

`cash.getTotalValue()` $\xrightarrow{\text{returns}}$ `17`. (register now has 17 dollars)

`cash.get1()` $\xrightarrow{\text{returns}}$ `2`. (now has 2 loonies)

`cash.get5()` $\xrightarrow{\text{returns}}$ `1`. (now has one five dollar bill)

`cash.allLoonies().get1()` $\xrightarrow{\text{returns}}$ `17`. (convert all money in register to loonies)

`cash.get5()` $\xrightarrow{\text{returns}}$ `0`. (no five dollar bill anymore)

Mark breakdown: 40 marks for Correctness

Include your CashRegister.java file in your submission assignment2.zip file. Do NOT submit the Money or Item classes.

Submission Recap

A complete assignment will consist of a single file (assignment2.zip) with the following four (4) files included: Problem1.txt or Problem.pdf, Person.java, Fish.java, and CashRegister.java.