BIRKBECK

(University of London)

MSc EXAMINATION FOR INTERNAL STUDENTS

MSc Advanced Computing Technologies, MSc Computer Science
MSc Computing for the Financial Services
MSc Information Technology, MSc Learning Technologies
MSc Information Systems and Management

Department of Computer Science and Information Systems

Software Design and Programming

COIY062H7

DATE OF EXAMINATION: Tuesday, 26th May 2015
TIME OF EXAMINATION: 14:30–16:30
DURATION OF PAPER: Two hours

RUBRIC:

- Candidates should attempt ALL 7 questions on this paper.
- You are advised to look through the examination paper before getting started to plan your strategy.
- Answers to programming questions must be in the Java or Scala programming languages unless otherwise stated.
- Simplicity and clarity of expression in your answers is important.
- Electronic calculators are NOT allowed.
- Start each question on a new page.

Question:	1	2	3	4	5	6	7	Total
Marks:	8	4	8	18	6	26	30	100

Question 1 ..... *Total: 8 marks*

What are the four main tenets of object-oriented programming?
Explain the advantages these features provide, illustrating your answer with appropriate examples.

Question 2 ..... *Total: 4 marks*

From a design perspective, should a class expose a minimal but sufficient interface (i.e., set of public methods) to its clients or should it expose an interface that provides all methods that its clients might ever need? Briefly justify your answer by the use of an appropriate example.

Question 3 ..... *Total: 8 marks*

(a) How does dependency injection assist with *loose coupling*? Provide appropriate examples to illustrate your answer. [4 marks]

(b) In one phrase each, state the two key limitations of constructors in Java. [4 marks]

Question 4 ..... *Total: 18 marks*

(a) Provide an example of a design pattern whose use is obvious from a class diagram but not from a sequence diagram. (Don't choose one that is built into (some) programming languages, such as inheritance.) Briefly justify your answer. [4 marks]

(b) Provide an example of a design pattern whose use is obvious from a sequence diagram but not from a class diagram. (Dont choose one that is built into (some) programming languages, such as iteration.) Briefly justify your answer. [4 marks]

(c) To implement the singleton pattern often (but not always) requires using what other pattern? Provide an appropriate example to support your answer. [4 marks]

(d) Consider a wrapper object whose implementation logs each call that is made to it. Briefly explain [6 marks]

- *when* and *why* the wrapper should be considered a *decorator*, and
- *when* and *why* that same wrapper should be considered a *proxy*.

Question 5 ..... *Total: 6 marks*

Consider the definition of the method `installDoor()`, written in Java-like pseudocode, from a `Contractor` class:

```
class Contractor {
   // any necessary instance variables
   // and methods defined here
  installDoor() {
    subcontractor = YellowPages.getSubcontractor();
    carpenter = subcontractor.getCarpenter();
    doorHandle = carpenter.getDoorHandle();
    doorBody = carpenter.getDoorBody();
    screws = carpenter.getScrews();
    door = carpenter.assemble(doorHandle, doorBody, screws);
    securityExpert = subcontractor.getSecurityExpert();
    securityExpert.installDoorSensors(door);
  }
}
```

What major design flaw is evident from the definition of the method? Which design principle(s) are violated? You may assume that any errors in method calls inside `installDoor()` are handled properly.

Question 6 ..... *Total: 26 marks*

When designing a compiler for a programming language it is common to represent programs as abstract syntax trees, recursive data structures which express code hierarchically.

The mini-programming language `Prog` supports decimal numbers, e.g., 1, 42, ..., and function applications, e.g., `plus(2, -2)`, `div(1, plus(2, -2))`, ...

In `progc`, a compiler for `Prog`, abstract syntax trees are encoded using two case classes: `Number` and `Func`, which both inherit from a common superclass `Tree`.

```
sealed trait Tree
final case class Number(n: Int) extends Tree
final case class Func(fn: String, args: List[Tree]) extends Tree
```

These classes can represent `Prog` programs as follows:

Program	Abstract syntax tree
1	Number(1)
42	Number(42)
plus(2,1)	Func("plus", Seq(Number(2), Number(1)))
div(1, plus(3,-1))	Func("div", Seq(Number(1), Func("plus", Seq(Number(3), Number(-1)))))

(a) Convert the following two `Prog` programs into their abstract syntax tree representation: 6 marks

- plus(1, plus(2, 3)),
- plus(plus(1, 2), 3).

Are the resulting trees the same?

(b) With the addition of function calls to `Prog`, the developers of `progc` started experiencing problems with debugging. Help the developers by writing a pretty printing function for `Prog` abstract syntax trees with the signature: 8 marks

```
def prettyprint(tree: Tree): String
```

The pretty printer should convert `Prog` abstract syntax trees into equivalent `Prog` source code, an example of which is shown below:

```
1
42
plus(2,1)
div(1, plus(3,-1))
```

(c) A new feature the developers wish to add to `Prog` is preventing *division by zero* errors, which happen when the function named `div` is called with two arguments, the second of which is `0`. 12 marks

Write a function

```
def divsByZero(tree: Tree): Seq[Tree]
```

which when given the abstract syntax tree of a `Prog` program, returns all divisions by zero which occur in that program.

Note: You only need to detect the most obvious division by zero mistakes, i.e., just the function applications having the form of `div(..., 0)`. You do not have to consider expressions such as `div(1, plus(-2, 2))`.

Question 7 ..... *Total: 30 marks*

Consider the problem of analysing the quiz grades of students on a course. Each student is represented by his/her name — a unique `String`. Grades are represented as `Int`s from 1 to 6. For each quiz the student receives a certain grade represented by the following type declaration:

```
type Result = (String, Int)
```

where the first element of the pair is the students name, and the second is their grade.

Similarly, a quiz is defined as a sequence of results for each student:

```
type Quiz = Seq[Result]
```

A quiz will always have grades for all the students enrolled on the course.

Finally, a course is simply a sequence of quizzes:

```
type Course = Seq[Quiz]
```

Each of the following parts of this question are independent of each other – you do not have to solve part (a) to solve part (b).

For any part question you may reuse the methods defined in the earlier parts. You should answer these questions in a *functional style*, regardless of whether you use Scala or Java code.

(a) Write the method `quizMeans` which, given a course, computes the list of mean grades of all the quizzes in that course: 6 marks

```
def quizMeans(course: Course): Seq[Float] = ???
```

A mean grade of a quiz is defined as the sum of the grades of all the students divided by the number of students.

(b) Write the method `studentMeans` which, given a course, computes the mean grades of all the students in that course: 6 marks

```
def studentMeans(course: Course): Map[String, Float] = ???
```

The method should return a `Map` mapping each student to his mean grade. The mean grade of a student is defined as the sum of his grades on all the quizzes divided by the number of quizzes.

(c) Write the method `topStudents` which, given a course and the number of top students n, computes the n top students on the course. 6 marks

```
def topStudents(course: Course, n: Int): Seq[String] = ???
```

The top n students are those n students with the highest mean grades. You can assume that no two students have the same mean grade.

Hint: in any part of the question you can use one of the methods defined earlier.

(d) Write the method `passingStudents` which, given a course, computes the list of the students which passed the course: 6 marks

```
def passingStudents(course: Course): Seq[String] = ???
```

The students that passed the course are those whose mean grade is greater than or equal to 3.5.

(e) Compute the number of students per final grade. A final grade is obtained by computing the mean grade of the student and then rounding it to the nearest integer. 6 marks

The assignment from a grade to the number of students with that grade is called a grade histogram, which we model with a `Histogram` datatype:

```
type Histogram = Map[Int, Int]
```

For example, if there were three students `Dipsy`, `LaLa`, and `TinkyWinky`, and they have mean grades 3.3, 3.2 and 6.0, respectively, then the histogram is:

```
Map(3 -> 2, 6 -> 1)
```

because there were two students with a mean grade which rounds to 3, and a single student with a result of 6. If no students have a certain grade, then you do not have to include it in the histogram (e.g., no one had a grade of 5 in the above example).

Write the method histogram which, given a course, computes the grade histogram of the course:

```
def histogram(course: Course): Histogram = ???
```

Hint: consider using the `math.round(x: Float)` method from the Scala standard library which rounds a real number to an integer number.

