BIRKBECK

(University of London)

BSc EXAMINATION FOR INTERNAL STUDENTS

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

Software and Programming III

BUCI056H6

CREDIT VALUE: 15 credits

DATE OF EXAMINATION: Tuesday, 7th June 2016
TIME OF EXAMINATION: 10:00am
DURATION OF PAPER: Three hours

RUBRIC

1. **Candidates should attempt ALL questions in the paper.**
2. **The number of marks varies from question to question.**
3. **You are advised to look through the entire examination paper before getting started, in order to plan your strategy.**
4. **Simplicity and clarity of expression in your answers is important.**
5. **Electronic calculators are NOT permitted.**
6. **You may answer questions using only the Java or Scala programming languages unless specified otherwise.**
7. **Start each question on a new page.**

Question:	1	2	3	4	5	6	7	8	9	Total
Marks:	18	9	8	12	11	6	12	8	16	100

Question 1 ..... Total: *18 marks*

(a) Briefly describe the four major principles of Object-Oriented Programming. You should provide appropriate examples to illustrate your answer. 6 marks

(b) What do you understand by the term *algebraic data type*? How can algebraic data types be emulated in Scala (or Java)? Provide a (brief) example to illustrate your answer. 6 marks

(c) Briefly compare the basic elements of functional programming with those of object-oriented programming. You should provide appropriate examples to illustrate your answer. 6 marks

Question 2 ..... Total: *9 marks*

Integer division by zero is an awkward problem as it can lead to exceptions, for example:

```
scala> 1.0 / 0.0
res0: Double = Infinity

scala> 1 / 0
java.lang.ArithmeticException: / by zero
```

(a) Create an object called `divide` with an `apply` method that accepts two `Int`s and returns `DivisionResult`. `DivisionResult` should be a sealed trait (or interface) with two subtypes:

a `Finite` type encapsulating the result of a valid division, and

an `Infinite` type representing the result of dividing by 0.

Here are some examples of its usage:

```
scala> divide(1, 2)
res7: DivisionResult = Finite(0)

scala> divide(1, 0)
res8: DivisionResult = Infinite
```

(b) Write a sample function that calls `divide`, matches on the result, and then returns a sensible description of that result.

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

Given the following Scala code:

```
1  object S extends App {
2    def ss(xs: Array[Int]): Array[Int] = {
3      if (xs.length <= 1) xs
4      else {
5        val pp = xs(xs.length / 2)
6        Array.concat(
7          ss(xs filter (pp >)),
8          xs filter (pp ==),
9          ss(xs filter (pp <)))
10     }
11   }
12
13   println(ss(Array(11, 3, 17, -4, 8, 9)).mkString(" "))
14 }
```

What is the output produced from executing this program and what function does `ss` compute? You should include a trace of the execution of the program in your answer.

Question 4 ..... Total: *12 marks*

(a) Consider a wrapper whose implementation logs each call that occurs. 4 marks

In no more than a couple of sentences each, explain when the wrapper should be considered a decorator (and why), and when that same wrapper should be considered a proxy (and why).

(b) Briefly describe the two key limitations of constructors in Java or Scala. 4 marks

(c) Describe the most important difference between a library and a framework. Provide appropriate examples to illustrate your answer. 4 marks

Question 5 ..... Total: *11 marks*

(a) To implement the *singleton design pattern* often (but not always) requires using what other design pattern? Provide example code to illustrate your answer. 3 marks

(b) Provide two disadvantages of the direct instantiation model that can be solved by using the factory method or factory class patterns. Provide appropriate examples to illustrate your answer. 4 marks

(c) Describe the two most important differences between composition and aggregation. Provide appropriate examples to illustrate your answer. 4 marks

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

(a) State two reasons that pair programming may deliver code with *more* functionality code than the same two people working independently. 3 marks

(b) State two reasons that pair programming may deliver code with *less* functionality than the same two people working independently. 3 marks

Question 7 ..... Total: *12 marks*

Write a class called `Playlist` which can be run from the command line. `Playlist` should take one command-line argument, the name of a playlist file in plaintext TAB-delimited value format in which each line contains the name, artist, album, and track number for each song in the playlist.

`Playlist` should read the input file and produce an output file in which any duplicate songs have been removed and the playlist is sorted by artist, then album, then track number. The output file's name should be the input file's name with `"-sorted"` appended to the base name.

For example, if you run your program with `ClassBumperMusic.txt` then the output file would be named `ClassBumperMusic-sorted.txt`.

You should make use of the following aspects of the language: collections, lambdas, and stream data.

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

(a) Why would you prefer property based dependency injection (DI) as against constructor DI? 2 marks

(b) The dependency injection design pattern adds ("injects") a dependency. Briefly describe where and when the dependency does not exist and where and when it does exist. 3 marks

(c) Suppose that component `A` depends on component `B`. Provide Java or Scala code constructs that could cause this dependence. Your answer should be descriptive and not use code examples. 3 marks

Question 9 . . . . . Total: *16 marks*

Write a method `transform` that traverses a heterogeneous list (of integers, strings, doubles, and/or nested heterogeneous lists), and evaluates to a list with exactly the same structure, where every non-list item has been transformed by the specified function. You may assume that none of the lists are empty, and that the first argument of `transform` is a function and the second argument is a list.

For example, the following code fragment (in Scala)

```
var l = List(1, List(2, List(3), 4, List(5, List(6)), List(8, 9)))
println(transform((x) => x + 1, l))
```

prints

```
List(2, List(3, List(4), 5, List(6, List(7)), List(9, 10)))
```

and the fragment

```
l = List(4, List(List(5)), List(6, List(7)))
println(transform((x) => x * x, l))
```

prints

```
List(16, List(List(25)), List(36, List(49)))
```

