The University of Nottingham Ningbo China

SCHOOL OF COMPUTER SCIENCE

A LEVEL 1 MODULE, SPRING SEMESTER 2019-2020

PROGRAMMING PARADIGMS

Time allowed: **TWO Hours THIRTY Minutes**

Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced

Answer ALL questions.
(Each question is worth equal marks)

Only silent, self-contained calculators with a Single-Line Display are permitted in this examination.

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn your examination paper over until instructed to do so

ADDITIONAL MATERIAL: Haskell standard prelude.

INFORMATION FOR INVIGILATORS: Exam papers must be collected at the end of the exam.

1. Object-Oriented Programming/Java (25 marks)

(a) Explain, in fewer than 20 words, what it means to declare a method final.

[2]

(b) Explain, in fewer than 20 words, what it means to declare a class final.

[2]

(c) Strings are immutable in java. Explain what this means, give one benefit of this and explain why this is a benefit.

[7]

(d) What is constructor overloading in Java.

[2]

(e) What is the difference between an inner class and a sub-class.

[2]

(f) Briefly explain the difference between a syntax, runtime and logical error in java and how the compiler or JVM alerts you to these.

[3]

(g) The following code is intended to show the ratio of values above 0.5 to those below 0.5. So if the random numbers were 0.8, 0.7, 0.2, 0.3 then the ratio is 1 (1 times as many above 0.5 than below). If it were 0.6, 0.7, 0.9, 0.2 it would be 3 (3 times as many above 0.5 than below). Correct the syntax, runtime and logical error in this code.

[7]

```
public class Rand2 {
      public static void main(String[] args)
{           double x, ratio;            int
plus, minus;
            for(int i=1; i<19; i++){
      x = Math.random();                if
(x > 0.5) plus = plus +1;
                  else minus = minus +1;
            }
            ratio = plus / minus;
            System.out.println("ratio " + ratio);
      }
}
```

2. Object-Oriented Programming/Java (25 marks)

(a) Give **two** similarities and **two** differences between interfaces and classes

[4]

(b) What does method overriding do, give a simple coded example

[4]

(c) Draw a class diagram to represent the following software requirements. Provide a key which clearly shows the notation which you used in your diagram.

"We require a parent class called Vehicle that has fields for the vehicle's colour and its maximum speed. It needs methods to set the colour and maximum speed and also one to display these values to the terminal. We also need a child class of Vehicle called Helicopter, this class uses the start method from a third class called Engine. The Engine class has 2 methods called start and stop, these methods print the time the engine started or stopped."

[4]

(d) With the requirements from (2.c) write the **three** required classes that produce the following output (your start time would be different):

```
C:\work\C\javaexam>java HeliDemo
Vehicle Colour is Red and Max speed is 200
Engine started at 2020-05-13T08:34:17.396420300
```

Using the main class:

```
class HeliDemo
{
     public static void main (String[] args)
     {
          Helicopter myHeli = new Helicopter();
     myHeli.setColour("Red");
myHeli.setMaxSpeed(200);
myHeli.vehicleDetails();
          myHeli.heliStart();
     }
}
```

[9]

Hint: You will need to import java.time.LocalDateTime

(e) Discuss, using examples from (2.c) and (2.d), **two** benefits of using the object oriented approach to produce this code using multiple classes.

[4]

3. Functional Programming / Haskell (25 Marks)

(a) Write down the most general types for each of the following functions:

(i) `increment n = n+1` [2]

(ii) `xor a b = (a || b) && not (a && b)` [2]

(iii) `suffixDoc = (++ ".doc")` [2]

(iv) `applyToPair f (a,b) = (f a, f b)` [2]

The Gregory-Liebniz infinite series for π is

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} + \cdots\right)$$

or, equivalently,

$$\pi = 4\sum_{i=1}^{\infty} \frac{s(i)}{2i-1}$$

where $s(i) = \begin{cases} -1 & \text{if } i \text{ is even} \\ +1 & \text{if } i \text{ is odd} \end{cases}$

In the following questions, you may use standard Prelude library functions.

(b) Write a function for $s(i)$ with type s :: Int -> Float, using guarded equations. [2]

(c) Write a function `piSub` of type `piSub :: Int -> [Float]` that takes an argument k and is defined as the infinite list

$$\left[\frac{s(k)}{2k-1}, \frac{s(k+1)}{2k+1}, \frac{s(k+2)}{2k+3}, \frac{s(k+3)}{2k+5}, \frac{s(k+4)}{2k+7}, \cdots\right]$$

[4]

Hint: Use the library function `fromIntegral` to convert a value given as type `Int` to `Float`.

(d) Use `piSub` to write a function `getPi` with type `getPi :: Int -> Float` that takes one argument n and calculates the Gregory-Liebniz series for π, using only the first n terms in the series. [2]

(e) Explain lazy evaluation and why it is used in Haskell.
As part of your answer, show how the expression `triple (5*4)` is evaluated as an illustration of lazy evaluation, where `triple x = x+x+x`. [9]

4. Functional Programming / Haskell (25 Marks)

In the (imaginary) country of Haskellonia, a business wants to write an income and tax calculation system in Haskell for its employees. Each employee is represented by the fields: Employee id (`Int`), name (`String`), income (`Int`), employee type (value 1, 2 or 3), role (`String`). In Haskell, this data will be coded as tuple of six elements.

(a) Write a data type `Emp` to represent one employee as a tuple of the five fields given above. [2]

(b) What is the type of a list of employees? [1]

(c) Which of these records is a correct `Emp` type object? [2]

```
(1, "John Smith", 20000, 2, "Accountant")
(2, "Arnold Jones", 30000, "B", "Programmer")
```

(d) Write two functions `empType` and `empInc` to return the employee type and income, respectively, from an `Emp` type record. Include the type declaration for each function. [2]

(e) Use list comprehension to write a function `empByType` with type declaration
`empByType :: [Emp] -> Int -> [Emp]`
that takes a list of employees and an employee type as arguments and returns a list of only those employees from the list with the given employee type. [2]

You may use any standard library function or function you have already defined in answering this question.

In Haskellonia, there is a different tax calculation for each employee type, as follows:-

- If employee type = 1 then tax = (income-10000)/10,

- If employee type = 2 then tax = (income-15000)/8,
- If employee type = 3 then tax = (income-5000)/12, with tax calculations rounded down to the nearest integer, and any negative tax calculations rounded up to zero.

The following code is used to express these three tax calculations:

```
taxCalcs :: [Emp -> Int]
taxCalcs = [\x -> max (div (empInc x - 10000) 10) 0,
\x -> max (div (empInc x - 15000) 8) 0,
           \x -> max (div (empInc x - 5000) 12) 0 ]
```

(f) Explain how `taxCalcs` expresses the tax calculations.
In particular, explain the use of `max` and `div` in the code and the `\x ->` part of the code. [3]

(g) Given an employee record `e1 :: Emp` with employee type `t :: Int`, write code to apply the correct tax calculation for this employee using `taxCalcs`. [2]

(h) Using recursion, write a function `empsTax :: [Emp] -> [Int]` to calculate the correct tax for each employee in a list of employees, based on their employee type. Return the corresponding list of taxes for each employee. [2]
You may use any standard library functions or functions you have already defined in answering this question.

(i) Describe what this function is doing:

```
incTaxByType es t = (t, sum(map empInc ets), sum(empsTax ets))
where
    ets = empByType es t
```

What meaning does the output from this function have? [2]

(j) Which type definition below is consistent with the function in 4(i)? [1]

(i) `incTaxByType :: [Emp] -> Int -> (Int, Int)`

(ii) `incTaxByType :: Emp -> Int -> (Int, Int)`

(iii) `incTaxByType :: [Emp] -> Int -> (Int, Int, Int)`

(k) Use the `map` library function (*not recursion*) to write a function

```
incTaxTotals :: [Emp] -> [(Int, Int, Int)]
```

to return a list of tuples output by `incTaxByType`, one for each employee type, given a list of employees with different employee types. [2]

You may use any standard library functions or functions you have already defined in answering this question.

(l) Write a function `showIncTaxTotals :: [Emp] -> IO()` to produce a tabular formatted output for total income and tax by employee type, given a list of employees. The output should be sent to the screen and appear as, for example,

```
 Total income and tax by employee type:
Type  Income     Tax

   1  55000      3500

   2  74000      3625

   3  0          0
 End of report
```

[4]

You may use any standard library functions or functions you have already defined.

Hint: you may need to define one or more auxiliary functions to implement this. The tab is written as `\t` in Haskell.