The University of Nottingham Ningbo China

SCHOOL OF COMPUTER SCIENCE

A LEVEL 1 MODULE, SPRING SEMESTER 2020-21

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)

Questions 1(a) and 1(b) relate to the following code:

```
class  Parent{
   void PrintData() {
      System.out.println("method of parent class");
   }
}

class Child extends Parent {
                                    //A
   void PrintData() {
      System.out.println("method of child class");
   }
}

class Sample1{
   public static void main(String args[]) {
      Child obj1 = new Child();
      Parent obj2 = new Parent(); //B
      Parent obj3 = new Child();  //C
      Child obj4 = new Parent();  //D
      Child obj5 = (Child)obj2;   //E
      Child obj6 = (Child)obj3;   //F
   }
}
```

(a) The code includes a compile-time error, a run-time exception and a best practice issue. Briefly explain what the lines **B** to **F** do and what is missing at line **A**. Include useful comments describing what this line of code is doing AND highlight where the THREE errors occur. (14)

(b) If you executed Printdata() for each for each viable object created within the main method what would the output be. (2)

(c) Briefly explain what parametric polymorphism means and provide a short Java code example to illustrate how it is used. (3)

(d) What is abstraction, explain why it is an Object-Oriented concept and state how it is achieved in Java and Haskell. (6)

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

(a) What are exceptions in Java, what can cause them and give an example of what happens when they are not handled? (3)

(b) What is the difference between a checked exception and an unchecked one? (2)

(c) Discuss what ***immutable*** means in java, give an example of where it occurs and why it may be useful. (4)

(d) What does the `Scanner` class do in Java and give an example of when you would use it. (3)

(e) What are Packages in Java and why are they useful? (3)

(f) What are the differences between int and Integer in Java and what are the benefits of using each. (5)

(g) Write a short commented program with:

1. an interface called `Animal` with two methods that do not return a value, these are called `eat` and `move`
2. an abstract class called `Fish` that provides an implementation of the `move` method of the `Animal` interface that prints "I swim"
3. a subclass called `Piranha` that provides an implementation of the `eat` method on the `Animal` interface that prints "I bite" (5)

3. Functional Programming / Haskell (25 Marks)

Parts (a) and (b) of this question are about polymorphism in Haskell and Java and the remainder is about coding with polymorphism in Haskell. **(25 marks).**

(a) What is meant by polymorphism, in general? Use one Java example and one Haskell example to illustrate. (5)

(b) Describe and contrast overloading in Java and Haskell. (3)

(c) Explain why the Haskell Prelude function `length` is polymorphic. (2)

(d) Construct a function `lengthNum` so that it will only return the length of a list of numbers. (i.e. it will not work for any other type of list, e.g. a list of `String`). (2)

Consider a system that stores data as key/value pairs. The system needs to be polymorphic so that key is any type `a` and the value is any type `b`. The key/value pairs are stored as a list of tuples, i.e. as type `[(a,b)]`. For example, if keys are given as `Int` and values as `String`, then this is a key/value store:

```
store1 = [ (2, "red"), (3, "green"), (2, "blue"), (1, "black") ]
```

Notice that a store may have multiple values for the same key.

(e) Suppose two stores `s1` and `s2` are concatenated into a single store `s1++s2`. If `s1` has type `s1 :: [(Int,String)]`, what must the type of `s2` be? (1)

(f) Use list comprehension to write a polymorphic function `getValues` that given any store `s` of type `[(a,b)]` and a key `k`, will output a list of all values for the key `k`.
For example, `getValues store1 2` evaluates to `["red", "blue"]`.
Show the type definition as well as the function definition of `getValues`. (2)

(g) Does type `a` need to be constrained to type class `Eq` in the type definition of `getValues`? Briefly explain why. (1)

(h) Does type `b` need to be constrained to type class `Eq` in the type definition of `getValues`? Briefly explain why. (1)

Consider a polymorphic function `applyToLists` in Haskell that takes three arguments in order:

1. a list of elements of type `a`,
2. a list of elements of type `b` and
3. a function `f` that takes two arguments of types `a` and `b` and returns a value of type `c`.

The function `applyToLists` should take corresponding values from the two lists, apply the function `f` to them then return the results in a list. For example,

```
applyToLists ["abc", "defg", "hij"] [1,0,2] (!!)
```

evaluates to `"bdj"`.

(i) Give the formal type definition of `applyToLists`. (2)

(j) Use recursion to write the function `applyToLists`. (2)

(k) Use patterns to ensure that `applyToLists` will work even if the two list arguments are of different lengths by ignoring unmatched elements in any of the list arguments. For example, `applyToLists [3,1,2,1,4] [1,5,1] (+)` will evaluate to `[4,6,3]`. (2)

(l) Consider the function that switches the sign of an integer conditional on a Boolean argument:

```
switchSign :: Bool -> Int -> Int
switchSign False n = n
switchSign True n = -n
```

Use `applyToLists` to write Haskell code that applies `switchSign` to each element of `[3,4,-3]` based on corresponding conditions `[False, True, True]`. What will this code evaluate to? (2)

4. Functional Programming / Haskell (25 Marks)

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

(i)
```
and3 x y z = x && y && z
```
(1)

(ii)
```
square x = x*x
```
(1)

(iii)
```
concats (xs:xss) = xs ++ concats xss
concats [] = []
```
(1)

(iv)
```
zap x f g = if x then f else g
```
(1)

(v)
```
zap2 x f g y = (if x then f else g) y
```
(1)

(b) Let `nor x y = (not x) && (not y)`.
Explain how `nor False True` is evaluated using currying. (2)

(c) Use list comprehension to write a function that outputs the list of square numbers up to some integer `n` given as an argument. Remember to include the function type declaration. (2)

(d) Evaluate each of these lambda expressions.

(i) `(\x -> (\y -> x+y) ) 3 4` (1)

(ii) `(\x -> (True, 2) ) (False, 3)` (1)

(iii) `(\x -> (\y -> x (x y)) ) (\z -> z+2) 4` (2)

Consider the following Haskell code:

```
data Tree a = Leaf a
            | Node (Tree a) a (Tree a)
t :: Tree Int
t = Node (Node (Leaf 1) 3 (Leaf 4)) 5
         (Node (Leaf 6) 7 (Leaf 9))

foldt :: (b->b->b) -> (a->b) -> Tree a -> b
foldt _ g (Leaf x) = g x
foldt f g (Node t1 x t2) = f (f (g x) (foldt f g t1)) (foldt f g t2)

occurs x = foldt (||) (==x)
```

(e) Write a function `onLeaf` that takes a tree as first argument and a value as the second and outputs `True` if and only if the value is in a leaf of the tree.
Do not use `foldt` to answer this question, and include the type definition for `onLeaf`. (3)

(f) Describe the function `foldt`; what can it be used for? (3)

(g) Describe the function `occurs`; what can it be used for? (2)

(h) What is the type of function `occurs`? (1)

(i) Evaluate the expression `occurs 3 t` (1)

(j) Use `foldt` to write a short Haskell function `sumTree` that will compute the sum of all values in a `Tree Int` object. (2)