香 港 大 學

THE UNIVERSITY OF HONG KONG

Bachelor of Engineering

Department of Electrical & Electronic Engineering

ELEC1503 ELEC2543 Object Oriented Programming and Data Structures

Examination

Date: 15 May 2014 Time: 9:30am-12:30pm

Answer **ALL** questions in Section A and

ANY 2 questions in Section B.

Use the same answer book for both sections.

Use of Electronic Calculators:

"Only approved calculators as announced by the Examinations Secretary can be used in this examination. It is candidates' responsibility to ensure that their calculator operates satisfactorily, and candidates must record the name and type of the calculator used on the front page of the examination script."

Section A – All the questions are compulsory.

I. Choose only 1 option as the most appropriate answer to each question. Each correct answer carries 2 marks. Write down all your answers in the answer book provided. (20%)

1) An object of the Java *String* class is immutable since
   A. methods of the *String* class always return the same object;
   B. the object can only be changed in its length by widening conversions after creation;
   C. methods of the *String* class always return new *String* objects that are actually modified versions of the original object;
   D. the object can only be changed in its value by casting after creation;
   E. the object cannot be changed in its length nor value after creation.

2) The Java *Thread* class provides
   A. a run() method to be automatically executed when a thread is created;
   B. a special way that can extend on multiple classes;
   C. the simplest way to create a thread by extending the class and then overloading the run() method;
   D. the simplest way to create a thread by extending the class and then overriding the run() method;
   E. none of the above.

3) A Java exception is
   A. an interface in which the default method is to ignore the generated exception message;
   B. part of the language intrinsic facility;
   C. an abstract class with constructor methods that will be invoked to handle a specific type of errors;
   D. an object that will be thrown at the site of error and later caught by the *finally* block of another exception object;
   E. none of the above.

4) Static methods in Java are

A. methods that can be invoked by static objects only;

B. methods with static method signatures;

C. interface methods that are static in nature;

D. methods that can be invoked through the class name only;

E. methods that can be invoked by objects of static data structures.

5) In Java programming, overriding refers to

A. the same method signature in the same class;

B. several methods of different names but having the same set of parameters;

C. several methods of the same name but having different sets of parameters;

D. the same method signature in two different classes that are related to one another via inheritance;

E. none of the above.

6) Java adopts the singly rooted object hierarchy since

A. there is no need to implement a constructor for each object;

B. there is no need to implement a garbage collector for each object;

C. programmers can easily know about what basic operations to be performed on each object;

D. compatibility checking between objects is easier with a more complex argument passing mechanism;

E. none of the above.

7) Queues are examples of

A. classic linear data structures;

B. classic nonlinear data structures;

C. multi-dimensional data structures;

D. network data structures;

E. scalar data structures.

8) A Java applet is

A. a stand-alone program with a *main* method for execution on a web browser;

B. embedded into another bytecode file for execution on a web browser;

C. embedded into an HTML file containing a tag referring to the bytecode file of the applet;

D. extended from the *JApplet* class of the *java.awt* package;

E. none of the above.

9) The set of operations defines how to interact with the abstract data type (ADT) is called

A. automatic methods;

B. interfaces;

C. modifiers;

D. receivers;

E. none of the above.

10) A polymorphic reference is

A. a Java exception that can change to another Java exception during program execution;

B. a Java interface that can change to another Java interface during program execution;

C. a Java interface that can change to a Java class during program execution;

D. a variable that can refer to an array of objects being changed to various types during program execution;

E. a variable that can refer to various types of objects during program execution.

II. Essential Concepts on Data Structures (15%)

1) Clearly differentiate dynamic data structures from static data structures. (4%)

2) Detail the major steps and also the Java program fragment to insert a node into a singly linked and sorted list. (11%)

III. Essential Concepts on Object Oriented Programming (15%)

1) A student wrote the following Java program.

```
class iVals {
    private double num;

    iVals(double iv) { num = iv; }

    public void changeiVals(Double newiv) {
      num = newiv;
    }}

public class TestiVals {
    public static void main (String[] args) {
      iVals obj1, obj2;

      obj1 = new iVals(3.1415);
      obj2 = new iVals(276.555);
      System.out.println(obj1);
      System.out.println(obj2);

      obj2 = obj1;
      obj1.changeiVals(369.138);
      System.out.println(obj1);
      System.out.println(obj2);
    }
}
```

a) In the underlined invocation to the `changeiVals(...)` method, it is clear that the method actually requires an object of the `Double` class yet the input parameter is only a primitive value of the *double* data type. Clearly explain how the invocation is possible. (2%)

b) After executing the above program, there is some unexpected output generated as below.

```
iVals@3b5b25a1
iVals@5d038b78
iVals@3b5b25a1
iVals@3b5b25a1
```

Clearly show your code modification such that the stored double values of the concerned objects can be successfully displayed through the `println(...)` statements. All the double values should be displayed and rounded up to 2 decimal places only [hint: the `java.text.DecimalFormat` class can be used for formatting decimal numbers]. (3%)

c) State the generated output of the modified program obtained in b). (2%)

2) Carefully examine the following Java program to answer the subsequent questions.

```
import java.*;

class B { int x = 17; }

class C extends B { int y = 38; }

class D extends C {
    int x = 29;

    public static void main(String[] args) {
      System.out.println("Val_1 = "    +
                  super.y / x);
      System.out.println("Val_2 = " +
                  this.x * super.super.x );

  }
}
```

a) Name the relationship between the two variable x's declared in both class `B` and `D` respectively. (1%)

b) The above Java program still contains some errors. Clearly show all your code modification to fix all the errors such that the modified Java program can be compiled and run successfully. (5%)

c) State the generated output of the corrected program obtained in b). (2%)

Section B – Each question carries equal marks. Answer ONLY TWO questions in this section. Write down ALL your answers in the answer book provided.

1. Data Structures in Java (25%)

A. Carefully study the following Java program fragment that is aimed to display, delete or store a maximum of 3 latest records of phone calls with their calling date and time onto a stack. For simplicity, all the users' inputs are assumed to be correct. Thus, there is no need to check for the validity of the input data.

```
import java.io.*;

class stack_of_calls {
    static final int stack_size = 3;
    int stack_index, i;
    String date[], time[];

    public stack_of_calls() { ….}  // constructor method

    /* to push an item onto the stack */
    void push(String new_date, String new_time) { ….}

    /* to pop (or retrieve) an item from the stack */
    void pop() { …. }

    void show_stack() {….}


class stack_of_ph_calls {
    static BufferedReader stdin;
    static stack_of_calls st;

    /* to add date & time onto the stack
     * For simplicity, the inputted date and time are
      assumed to be correct. */
    public static void add() throws IOException { … }

    public static void main(String[] args) throws
      IOException {
        boolean        exit_menu = false;
        String         option_str;
        int            option;

        stdin = new BufferedReader(new InputStreamReader(
                System.in));
         /* Stack is constructed and initialized ! */
         st = new stack_of_calls();

         while (!exit_menu) {
            System.out.println("1. Add date & time");
```

```
System.out.println(
     "2. Delete most recent call");
System.out.println("3. Show call register");
System.out.println("4. Exit");
System.out.print("Your action: ");
option_str = stdin.readLine();
option = Integer.parseInt(option_str);

switch (option) {
    case 1 : add();
             break;
    case 2 : st.pop();
             break;
    case 3 : st.show_stack();
             break;
    case 4 : exit_menu = true;
             break;
    default: exit_menu = false;
} } } }
```

Complete the above Java program such that the following output can be generated after the latest 3 records of phone calls were stored onto the stack. All the users' inputs were boldfaced and underlined for your reference.

```
1. Add date & time
2. Delete most recent call
3. Show call register
4. Exit
Your action: 3

2009-02-22      19:24:16
2011-11-06      09:34:56
2012-07-13      03:14:36

1. Add date & time
2. Delete most recent call
3. Show call register
4. Exit
Your action: 2
1. Add date & time
2. Delete most recent call
3. Show call register
4. Exit
Your action: 3
```

```
2011-11-06      09:34:56
2012-07-13      03:14:36

1. Add date & time
2. Delete most recent call
3. Show call register
4. Exit
Your action: 4
```

(20%)

B. Clearly explain the properties of "queues" as data structures. (5%)

2. Class Inheritance and Method Invocation in Java (25%)

A. Complete the following Java program so that it will display the changes of values over successive method invocations. Besides, clearly state all the generated output after the successful execution of the completed Java program.

```
class Num { private int value;

      public Num(int update) { value = update; }
      ….
}


class PModifier
{
      public void changeValues(int f1, Num f2, Num
      f3) {
      System.out.println(
            "Before changing the values:");
      System.out.println("f1\tf2\tf3");
      System.out.println(
            f1 + "\t" + f2 + "\t" + f3 + "\n");

      f1 = 469;
      f2.setValue(486);
      f3 = new Num(175);

      System.out.println(
            "After changing the values:");
      System.out.println("f1\tf2\tf3");
      System.out.println(f1 + "\t" + f2 + "\t" + f3
      + "\n");
}}
```

```
public class GTest
{     public static void main(String[] args) {
      PModifier modifier = new PModifier();

      int x1 = 569;
      Num x2 = new Num(138);
      Num x3 = new Num(234);

      System.out.println(
            "Before calling changeValues:");
      System.out.println("x1\tx2\tx3");
      System.out.println(
            x1 + "\t" + x2 + "\t" + x3 + "\n");

      modifier.changeValues (x1, x2, x3);

      System.out.println(
            "After calling changeValues:");
      System.out.println("x1\tx2\tx3");
      System.out.println(
            x1 + "\t" + x2 + "\t" + x3 + "\n");
}}
```

(12%)

B. Carefully study the following Java program to answer the subsequent questions.

```
abstract class Actor  { abstract void act(); }
interface CanFight    { void fight(); }
interface CanRun      { void run(); }
interface CanFly      { void fly(); }

class ActionCharacter { public void fight() {
    System.out.println("I can fight well !"); } }

class  Hero  extends  ActionCharacter  implements
     CanFight, CanRun, CanFly {
    public void run() {
      System.out.println("I can run fast !");}
    public void fly() {
      System.out.println("I can fly high !"); }
}

class HappyActor extends Actor implements CanRun {
    public void act() {
      System.out.println("HappyActor"); }
    public void run() {
      System.out.println("I can run slowly !");}
}

class SadActor extends Actor implements CanFight {
    public void act() {
      System.out.println("SadActor"); }
```

```
    public void fight() {
     System.out.println("I can fight a bit !"); }}

class Stage {
    Actor a = new SadActor();
    Hero  h = new Hero();

    void change() { a = new HappyActor(); }
    void go() { a.act(); }

    void exercise(CanRun x) { x.run(); }
    void soar(CanFly y)     { y.fly(); }
    void trial(CanFight z)  { z.fight(); }
}

public class Transformer {
    public static void main(String[] args) {
Stage s = new Stage();
Hero  h = new Hero();

s.go();
s.trial(s.a);
s.change();
s.go();

s.exercise(s.a);
s.exercise(h);
s.soar(h);
s.trial(h);
} }
```

---

i) The above Java program still contains 2 compilation errors. Highlight each of these errors, and propose a solution to fix it. (6%)

ii) Clearly state the expected output after successful compilation and execution of the corrected Java program obtained in i). (7%)

3. Object Serialization and Java Threads (25%)

A. Clearly explain "object serialization" in Java. (10%)

B. The following Java program is targeted to create three threads for which each thread will display a message to identify itself, put to sleep for 3 seconds, and lastly print out a message to signal the end of the execution.

```
public class ThreadRoller extends Thread
{ String threadName;
```

```
  static float total_time = 0;

public ThreadRoller(String name){ threadName = name;}

public void run() { …. }

public static void main(String args[])
{  long start, end;

    System.out.println("Creating thread 1");
    Thread t1 = new ThreadRoller("Dummy");
    System.out.println("Creating thread 2");
    Thread t2 = new ThreadRoller("Hurry");
    System.out.println ("Creating thread 3");
    Thread t3 = new ThreadRoller("Sleepy");

    start = System.currentTimeMillis();
    t1.start(); t2.start(); t3.start();
    end = System.currentTimeMillis();

    System.out.println("Total time : " +
              (float) (total_time)/1000);

}}
```

---

i) Complete the `run()` method of the above Java program so that the following output can be produced.

```
Creating thread 1
Creating thread 2
Creating thread 3
I am Java thread named Dummy
I am Java thread named Hurry
I am Java thread named Sleepy
Total time : 0.0
Dummy is finished!
Hurry is finished!
Sleepy is finished!
```

(5%)

ii) Rewrite the complete Java program in i) so that it will create the three threads using the `Runnable` interface instead of the `Thread` class.

(5%)

iii) Show your code modification to the rewritten Java program in ii) such that the modified program can successfully display the total time (in seconds) of executing the three threads after all of them are terminated.

(5%)

*** END OF PAPER ***