CPSC 101 Winter 2013
Midterm II—08 March 2013

Name (Printed) : _____

Signature : _____

Student Number : 2 3 0 _ _ _ _ _ _

Question	Score
1	/10
2	/3
3	/7
4	/4
5	/2
6	/4
7	/2
8	/1
9	/1
10	/3
11	/2
12	/6
13	/5
Total	/50

- This is a **50** minute exam. This exam contains **10** pages of questions not including this cover page. Make sure that you have all of them.
- Put your name on the top right hand corner of each page as examination papers sometimes come unstapled.
- Non-programmable calculators and simple wrist-watches are allowed. **Cell-phones and other non-medical electronic devices are prohibed**.
- Answer all questions on the exam sheet. If you do some of your work on the back of a page, clearly indicate to the marker what work corresponds with which question.
- Partial marks shall be awarded for clearly identified work.
- *Read each question carefully. Ask yourself what the point of the question is. Check to make sure that you have answered the question asked.*
- This exam counts as **15%** of your total grade. There are **50** points total on the exam.

True False

1 each

1. Circle **TRUE** or **FALSE** as appropriate. Questions that don't clearly indicate *one* choice shall be marked wrong. If you feel that the answer depends on how you interpret the question, give a brief reason for the answer you chose.

(a) If a class contains an abstract method, the class must be abstract. **TRUE FALSE**

(b) If a class contains a final method, the class must be final. **TRUE FALSE**

(c) An overriding method must be at least as public as the method it overrides. **TRUE FALSE**

(d) A JAVA graphics program doesn't necessarily end when its `public static void main(String [] args)` exits. **TRUE FALSE**

(e) The `java.awt.*` classes are newer than the `javax.swing.*` classes. **TRUE FALSE**

(f) The JAVA Swing libraries are designed to operate on a single thread. **TRUE FALSE**

(g) JAVA explicitly supports concurrency. **TRUE FALSE**

(h) Interfaces allow for inheritance. **TRUE FALSE**

(i) Interfaces allow for run-time polymorphism. **TRUE FALSE**

(j) The `hashCode` method of the `Object` class is `final`. **TRUE FALSE**

Inheritance and Interfaces

2. When overriding a method

(1) (a) What is the rule about the return type?

(1) (b) What is the rule about the signature?

(1) (c) How should the post-condition of the method in the subclass relate to the post-condition of the method in superclass?

3. Suppose that `FrenchFries` is a direct subclass of `DietFood`, which in turn is a sub-class of `Food`. Suppose further that we are looking at the code:

```
1  Food brainFood = new FrenchFries(37) ;
2  // ...
3  System.out.println(brainFood.getCalories()) ;
```

(1) (a) What principal justifies the assignment in line 1?

(1) (b) In which class must the `.getCalories()` method exist in order for the code to compile?

(1) (c) Suppose that the `Food`, `DietFood`, and `FrenchFries` classes all have `.getCalories()` methods.

Assuming no assignments to `brainFood` between lines 1 and 3, the `.getCalories()` method of *which* class is executed at run-time?

(2) (d) Now suppose that the `Food` class and the `DietFood` class have `.getCalories()` methods, but that there is no explicit `.getCalories()` code in the `FrenchFries` class.

What happens at line 3 at compile- and/or run-time in this circumstance? Justify your answer.

(2) (e) Suppose that `greaseContent()` is a method found only in the `FrenchFries` class. How can we find the grease content of `brainFood`?

Graphics and User Interfaces

4. These questions are about Java, concurrency, and Swing.

(2) (a) What is concurrency?

(2) (b) Explain how concurrency and the design of the Java Swing library are connected.

(1) **5.** (a) In Figure 1 on page 9, what kinds of things can cause the `paintComponent` method to be called?

(1) (b) Suppose that users want to cause part of a GUI to be redrawn. Instead of calling the `paintComponent` method directly, what should they do?

(4) **6.** In Figure 1 on page 9, there are references to the imported classes `JPanel`, `Color`, and `Graphics`? Which of these classes are likely imported from `javax.swing`, and which are likely imported from `java.awt`? What else should Java-programmers using graphics know about `java.awt.*` and `javax.swing.*`?

(2) **7.** Consider the code fragment

```
1  private static void createAndShowGUI() {
2     JFrame frame = new JFrame("RadioButtonDemo");
3     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
4     ...
```

Explain what line 3 does and why it is important to put it in simple GUI applications.

Choose the *best* answer possible, and clearly indicate *one* choice. Supply reasons to the right if you are not sure, or think that the question is open to multiple interpretations.

(1) **8.** The `JFrame` class is

(**a**) a framework class for building a GUI.
(**b**) an interface.
(**c**) a top-level window class provided by the `javax.swing` libraries.
(**d**) a top-level window class provided by the `java.awt` libraries.

(1) **9.** The `javax.swing.SwingUtilities.invokeLater` is

(**a**) a static method for ensuring that your application quits when its main window does.
(**b**) a non-static method for ensuring that your application quits when its main window does.
(**c**) a static method that invokes its argument on the Swing GUI thread.
(**d**) a non-static method that invokes its argument on the Swing GUI thread.

(3) **10.** Suppose that the `CarComponent` class is written as

```
public class CarComponent extends JComponent
{
  public void paintComponent(Graphics g)
      {    /* ... */  }
}
```

(a) Consider the code below:

```
  public static void guiStuff()
      {
      JFrame aFrame = new JFrame() ;
      CarComponent aCar = new CarComponent ("VW Bug") ;
      aFrame.add(aCar) ;
      // ...
      }
```

How does subclassing `CarComponent` allow us to add `aCar` to a `aFrame`?

(b) What role does runtime polymorphism play in making this code work?

(2) **11.** (Not really about Graphics.) What are two methods of the `Object` class that are commonly overridden?

Longer answer

(6) **12.** See Figure 2 on page 10. Show how to subclass `NinjaAssassin` from `Person`, so that if the story went:

```
1   public static void main(String [] args) {
2     Person bambi = new NinjaAssassin("Bambi") ;
3     Person godzilla = new Person("Godzilla") ;
4     bambi.murders(godzilla) ;
5     System.out.println("Hi, I'm "+bambi.getName()) ;
6     return;
7   }
```

it would print out "`Hi I'm Ninja Assassin Bambi, killer of 1`".

To do this correctly, you must override `getName` and `murders`. Avoid adding another private member for names. Instead show how to use "`super`" (a) in the `NinjaAssassin` constructor, and (b) to *add* to the behaviour of `murders`. Write as much other code as you need to make the `NinjaAssassin` class work as shown above.

(4) **13.** (a) Code a `Student` class that has one `public` zero-argument constructor, and one public method

```
public long getStudentId() { ... }
```

that returns an integer like `230099999`. Every `Student` object must have a unique student identification number. Identification numbers must be larger than `230000000`, and you can assume that there are less than one million of them.

(1) (b) How do you prevent subclasses of `Student` from overrriding `getStudentId()`?

Figure 1: Sample code for Questions 5

CounterPanel.java

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /* comments that start with "///" indicate missing code */
6
7  public class CounterPanel extends JPanel {
8    private int myCounter ;
9    public void increment() { ++myCounter ; }
10   public int  getCounter() { return myCounter ; }
11
12   public CounterPanel () {
13     setBackground(Color.black) ;
14     setForeground(new Color(255,63,63)) ;
15     /// set things up so that clicking the mouse increments the count
16   }
17
18   public void paintComponent(Graphics g) {
19     super.paintComponent(g) ;
20     setFont(new Font("SansSerif",Font.PLAIN,35)) ;
21     g.drawString(""+getCounter(),getWidth()/2-10,getHeight()/2+5) ;
22   }
23
24   public static void main(String[] args) {
25     javax.swing.SwingUtilities.invokeLater(
26         /// code that calls createAndStartGui() ;
27         );
28   }
29
30   private static void createAndStartGui() {
31     /// set up a JFrame called aFrame
32     aFrame.add(new CounterPanel()) ;
33     aFrame.setVisible(true) ;
34   }
35 }
```

Figure 2: Code for Question 12.

Person.java

```
1  public class Person
2  {
3    private String      myName ;
4    private Person      myMurderer ;
5    private boolean     amAlive ;
6    private static int  thePersonCount = 0 ;
7
8    public Person murderer()              { return myMurderer ; }
9    public String  getName()              { return myName ; }
10   public boolean isAlive()              { return amAlive ; }
11   public static int getNumberLiving()   { return thePersonCount ; }
12
13   public Person(String name) {
14     myName = name ;
15     amAlive = true ;
16     ++thePersonCount ;
17     myMurderer = null ;
18   }
19
20   public void dies() {
21     if (amAlive) {
22       --thePersonCount ;
23       amAlive = false ;
24     }
25   }
26
27   public void murders(Person victim) {
28     if (victim.isAlive()) {
29       victim.dies() ;
30       victim.myMurderer = this ;
31     }
32   }
33 }
```