CPSC 101 Winter 2012
Midterm II—09 March 2011

Name (Printed) : _____

Signature : _____

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

Question	Score
1	/10
2	/8
3	/4
4	/4
5	/3
6	/3
7	/10
8	/6
9	/2
10	/3
11	/2
12	/2
13	/3
Total	/60

- 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.
- *Read each question carefully. Ask yourself what the point of the question is. Check to make sure that you have answered the question asked.*
- 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.
- Non-programmable calculators and simple wrist-watches are allowed. **No cellphones or other non-medical electronic devices**.
- This exam counts as **15%** of your total grade. There are **60** 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) Methods that throw objects from `RuntimeException` (or its subclasses) must declare this with a `throws` declaration. **TRUE FALSE**

(i) Every exception that can be thrown by an overriding method can also be thrown by the overridden method. **TRUE FALSE**

(j) The object thrown in a `throw` statement must be an `Exception` (or a subclass of `Exception`). **TRUE FALSE**

Inheritance and Interfaces

(2) **2.** Consider the code shown in Figure 1 on page 10.

(1) (a) What is the direct superclass of `Money`?

(1) (b) What does the word "`final`" on line 1 mean?

(3) (c) Provide code that you might insert in this class to override one behaviour inherited from the class `Object`.

(1) (d) Name one other behaviour inherited from the class `Object` that you might likely want to override.

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

```
1  Currency debt = new CanadianDollars(3400.15) ;
2  // ...
3  System.out.println(debt.value()) ;
```

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

(1) (b) Suppose that the `Currency`, `Dollars`, and `CanadianDollars` classes all have `.value()` methods.

Assuming no assignments to `myDebt` between lines 1 and 3, which class's `.value()` method is executed at run-time?

(2) (c) Now suppose that the `Currency` class and the `Dollars` class have `.value()` methods, but that there is no explicit `.value()` code in the `CanadianDollars` class.

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

(4) **4.** Write a short piece of code that illustrates the use of the Java keyword "`super`". The code doesn't have to be anywhere near complete, but must illustrate where to use the keyword.

Exceptions

(3) **5.** Explain in general terms why error handling is difficult.

(3) **6.** Is it the *dynamic* scope or the *static* scope of `try`-block that is important? Explain. Explain what *static* and *dynamic* scope are about more generally.

7. The following questions are about the technical aspects of exception handling in JAVA.

(3) (a) What happens when an exception is thrown and there is no `try`-block surrounding the `throw` statement? What happens when an exception is thrown and the nearest surrounding `try`-block has no `catch`-blocks that match the exception thrown?

(2) (b) What happens when an exception is thrown and the nearest surrounding `try`-block has multiple `catch`-blocks that match the exception thrown? What does this say about the order of `catch`-blocks?

(2) (c) What is the purpose of a `finally`-block? That is, when would you use one?

(3) (d) Draw a sketch of the inheritance hierarchy for `Object`s that can been thrown and caught in `Java`.

(3) **8.** (a) The formula to compute the hyperbolic arc cosine, x, of a number u is

$$x = \cosh^{-1} u = \pm \log\left(u + \sqrt{u^2 - 1}\right). \tag{1}$$

For real numbers this formula results in an error if and only if $u < 1$. Write an `arc_cosh` method to compute hyperbolic arc cosines using the positive version of formula (1). Your method should throw an `IllegalArgumentException` when it gets bad input data[1].

(3) (b) Show how to make `arc_cosh` throw a programmer-defined exception `ImaginaryResultException` that is a direct subclass of `IllegalArgumentException`.

[1] `IllegalArgumentException` is in scope without any `import`, and is a subclass of `RuntimeException`. The log and $\sqrt{x}$ functions are `Math.log` and `Math.sqrt` respectively.

Graphics and User Interfaces

(2) **9.** 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.

(3) **10.** In the code discussed in class the `NameComponent` was written as

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

(a) Why must we use subclassing in order to be able to add `NameComponent` object to a `JFrame` object?

(b) How does runtime polymorphism help make this code work?

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.

(2) **11.** 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.

(2) **12.** 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.

Longer answer

(3) **13.** Gaddis [*Starting out with Java: From Control Structures through Object* third edition] and Horstmann [*Big Java* third edition] explain exception handling in similar ways. For instance, both introduce exception handling with file i/o.

One of the differences is that Gaddis introduces `try-catch` syntax first before he shows how to throw an exception. Horstmann, on the other hand, first shows how to `throw` an exception.

Comment on which approach you prefer, and which gives the reader a better impression of how to use exception handling. What do you feel is the single most important concept for some learning exception handling to grasp?

```
1  public final class Money
2  {
3    private byte myCents ;
4    private long myDollars ;
5
6    public Money(long dollars, int cents)
7        {
8        myCents = (byte) (cents % 100) ;
9        myDollars = dollars + (cents-myCents)/100 ;
10       }
11   public float asDollars () { return (float)
12                                    (myCents/100.0 + myDollars) ; }
13   public long asCents    () { return myCents+ 100*myDollars     ; }
14   // More ...
15 }
```

Figure 1: Code for Question 2 on page 2