CPSC 101 Winter 2018
Midterm II—07 March 2018

Name (Printed) : _____

Signature : _____

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

Question	Score
1	/2
2	/6
3	/3
4	/5
5	/4
6	/4
7	/4
8	/2
9	/4
10	/3
11	/4
12	/9
13	/2
14	/2
15	/2
16	/2
17	/2
Total	/60

- This is a **50** minute exam. This exam contains **9** 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 prohibited**.
- 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 **60** points total on the exam.

Graphics and User Interfaces

(2) **1.** 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.

2. This questions is really about relations between classes, although the specifics are stated in terms of GUI classes.

(2) (a) Code for `paintComponent` often looks like

```
1      @Override
2      protected void paintComponent(Graphics g) {
3          Graphics2D g2d = (Graphics2D) g ; ...
```

Explain what this says about how the `Graphics2D` class is related to the `Graphics` class.

(2) (b) The JAVA documentation states that `Graphics2D` is in fact an `abstract` class. What does it mean to say that `Graphics2D` is an abstract class?

(2) (c) According to what we learned in Chapter 9, it is impossible to create objects of an abstract class, yet `g2d` above *is-a* `Graphics2D` variable. How can you explain this apparent contradiction?

(3) **3.** Explain what the *model-view-controller* idea is about.

- What is one possible advantage of separating the model and the view?
- Does the JAVA Swing library use this idea?

(5) **4.** Consider the code shown in Figure 2 on page 8.

(a) What layout managers do in general?

(b) What does the code on Line 22 ("`this.setLayout(...)`") do?

(c) What does line 23 ("`this.add(...)`") do? How does this relate to layout managers and painting?

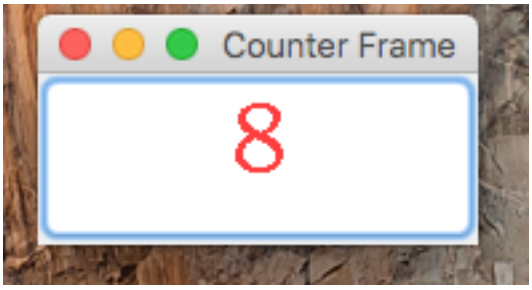

Figure 1: Sample run produced by Code in Figure 3 on page 9.

Many of the following questions refer to the code shown in Figure 3 on page 9. This code can be combined with a `JFrame` to produce an application, a Mac version of which is shown in Figure 1. This application does nothing other than increment the count shown when the Counter Frame is clicked on with the mouse.

(4) **5.** In Figure 3 on page 9, there are references to the imported classes `JButton`, `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.*`?

(4) **6.** In order that clicking the `CounterButton` causes the count to increment, what code (be as precise as possible) needs to be added at Line 13 in Figure 3 on page 9? In particular, how do you cause the updated count to display?

(4) **7.** What code needs to be added (be as precise as possible) at Line 31 in Figure 3 on page 9 in order that the `JFrame` displays a `CounterButton`?

(2) **8.** In Figure 3 on page 9, what kinds of things can cause the `paintComponent` method to be called *other than* code associated with Question 6?

9. The `paintComponent` method in `JButton` and the overriding method in `ButtonPanel` are both declared to be `protected`.

(2) (a) Explain briefly what `protected` means for a method.

(2) (b) Explain why `protected` is a good choice here.

(3) **10.** Line 17 of the `paintComponent` method shown in Figure 3 says "`super.paintComponent(g) ;`".

(a) What method of which class is called by this line of code?

(b) What do you think might happen if this line is omitted?

(4) **11.** The Java documentation says that the Swing classes should be run on their own thread. What does this mean? How do we accomplish this?

True False

1 each

12. 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) The code `Graphics2D g2 = (Graphics2D) g;` is an example of a downcast. **TRUE FALSE**

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

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

(d) The `java.awt.*` classes are often subclasses of the `javax.swing.*` classes. **TRUE FALSE**

(e) `ActionListener` is an example of a JAVA `interface`. **TRUE FALSE**

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

(g) JAVA explicitly supports multiple threads of execution. **TRUE FALSE**

(h) A `JButton` can have at most one `ActionListener` attached to it. **TRUE FALSE**

(i) A `Graphics2D` object can be assigned to a `Graphics` variable. **TRUE FALSE**

Multiple Choice

For the questions below, 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) **13.** `ActionListener` is
- (**a**) an interface.
- (**b**) an abstract class.
- (**c**) a concrete class with application specific behaviour.
- (**d**) a concrete class with default behaviours.

(2) **14.** The argument supplied to `addActionListener` is an instance of
- (**a**) a concrete class with default behaviours.
- (**b**) a concrete class with application specific behaviour.
- (**c**) an abstract class.
- (**d**) an interface.

(2) **15.** Objects of the class `MousEvent` are
- (**a**) generated by a `JButton` when it is clicked.
- (**b**) exceptions thrown when the mouse does something illegal.
- (**c**) sent to `MouseListener`s attached to `Component`s.
- (**d**) sent to `Component`s.
- (**e**) *none of the above*.

(2) **16.** The `JFrame` class is
- (**a**) an interface.
- (**b**) a framework class for building a GUI.
- (**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) **17.** *Anti-aliasing* is
- (**a**) the process of mapping from user co-ordinates to device co-ordinates.
- (**b**) another name for downcasting.
- (**c**) the use of *logical* rather than *physical* pixels on modern displays.
- (**d**) a method to help draw curved boundaries through rectangular pixels.

Figure 2: Sample code for Question 4

SimpleFrame.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  class PushMeButton extends JButton
8  {
9    public PushMeButton()
10       {
11       super("Push Me") ;
12       /// ...
13       }
14
15 }
16
17 public class SimpleFrame extends JFrame
18 {
19
20   public SimpleFrame ()
21       {
22       this.setLayout(new FlowLayout());
23       this.add(new PushMeButton()) ;
24       /// ...
25       this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
26       this.setVisible(true);
27       }
28
29   // ...
30
31   private JButton myButton ;
32   private int myFrameNumber ;
33 }
```

Figure 3: Sample code for Questions 8

CounterButton.java

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5
6  public class CounterButton extends JButton {
7    private int myCounter ;
8    public void increment() { ++myCounter ; }
9    public int getCounter() { return myCounter ; }
10
11   public CounterButton () {
12     super("") ;
13     /// cause the button to increment when pressed.
14   }
15
16   public void paintComponent(Graphics g) {
17     super.paintComponent(g) ;
18     setFont(new Font("SansSerif",Font.PLAIN,35)) ;
19     g.drawString(""+myCounter,getWidth()/2-10,getHeight()/2+5) ;
20   }
21
22   public static void main(String[] args) {
23   /// Code to call createAndStartGui()
24   }
25
26   private static void createAndStartGui()      {
27     JFrame aFrame = new JFrame() ;
28     aFrame.setTitle("Counter Frame");
29     aFrame.setLocation(55,20) ;
30     aFrame.setSize(new Dimension(80,80)) ;
31     /// Connect a CounterButton to the frame
32     aFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
33     aFrame.setVisible(true) ;
34   }
35 }
```