

Name: _____ Section: _____ CM: _____

CSSE 220—Object-Oriented Software Development

Exam 2, January 24th, 2018

This exam consists of two parts. Part 1 is to be solved on these pages. Part 2 is to be solved using your computer, and will be taken on Friday. You will need network access to download template code and upload your solution for part 2.

Resources for Part 1: You may use a single sheet of $8\frac{1}{2} \times 11$ inch paper with notes on both sides. You can also use your "UML Cheatsheet" and your "Design Principles" handouts if you brought them. Your computer *must be closed* the entire time you are completing Part 1.

Problem	Poss. Pts.	Earned
1	11	_____
2	10	_____
3	4	_____
4	10	_____
5	15	_____
Paper Part Subtotal	50	_____
C1. Recursion problems	18	_____
C2. Polymorphism problem	12	_____
C3. GUI problem	20	_____
Computer Part Subtotal	50	_____
Total	100	_____

Part 1—Paper Part

1. (11 points total. 1 point each question.)

On this page there is a UML diagram and the code it represents. Please look at the code and diagram and then answer the questions below.

a. (1 point) The coupling in this design is:

Circle one: HIGH or LOW

Why? _____

b. (1 point) The cohesion in this design is:

Circle one: HIGH or LOW

Why? _____

c. (1 point) What line(s) of code does the solid line with an open arrow going from Consumer to Producer correspond to?

```

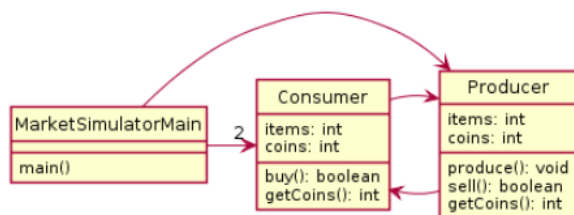
1 class Consumer {
2     private Producer p;
3     private int items=0, coins;
4     public Consumer( Producer p, int c) {
5         this.p = p;
6         this.coins = c;
7     }
8     public boolean buy() {
9         if (coins > 0 && p.sell()) {
10             items++; coins--;
11             return true;
12         }
13         return false;
14     }
15     public int getCoins() {
16         return this.coins;
17     }
18 }

```

```

19 public class MarketSimulatorMain {
20     private Producer p1;
21     private Consumer c1, c2;
22     public static void main(String[] args) {
23         new MarketSimulatorMain();
24     }
25     public MarketSimulatorMain() {
26         p1 = new Producer(null);
27         c1 = new Consumer(p1, 6);
28         c2 = new Consumer(p1, 7);
29         for (int i=0; i< 10; i++) {
30             c1.buy(); c2.buy();
31         }
32         System.out.println( c1.getCoins() ); //1
33         System.out.println( c2.getCoins() ); //2
34         System.out.println( p1.getCoins() ); //10
35     }
36 }
37 class Producer {
38     private Consumer b;
39     private int items =10, coins=0;
40     public Producer( Consumer b) {
41         this.b = b;
42     }
43     public void produce() {
44         this.items++;
45     }
46     public boolean sell() {
47         if (this.items > 0) {
48             this.items--; this.coins++;
49             return true;
50         }
51         return false;
52     }
53     public int getCoins() {
54         return this.coins;
55     }
56 }

```



```

1 class ProCon {
2     private MarketSimulatorMain sim;
3     private ProCon other;
4     private boolean isPro;
5     private int items, coins;
6     public ProCon(boolean isPro,
7         ProCon other, int coins, int items,
8         MarketSimulatorMain sim) {
9         this.other = other;
10        this.isPro = isPro;
11        this.items = items;
12        this.coins = coins;
13        this.sim = sim;
14    }
15    public void produce() {
16        if (isPro) { this.items++; }
17    }
18    public boolean sell() {
19        if (isPro && this.items > 0) {
20            this.items--;
21            this.coins++;
22            return true;
23        }
24        return false;
25    }
26    public boolean buy() {
27        if (!isPro && coins > 0 &&
28            other.sell()) {
29            items++; coins--;
30            return true;
31        }
32        return false;
33    }
34    public int getCoins() {
35        return this.coins;
36    }
37 }

```

```

38 public class MarketSimulatorMain {
39     private HashMap<String, ProCon> map =
40         new HashMap<String, ProCon>();
41     public static void main(String[] args) {
42         new MarketSimulatorMain();
43     }
44     public MarketSimulatorMain() {
45         map.put("p1",
46             new ProCon(true, null, 0, 10, this));
47         map.put("c1",
48             new ProCon(false, map.get("p1"), 6, 0, this));
49         map.put("c2",
50             new ProCon(false, map.get("p1"), 7, 0, this));
51         for (int i=0; i< 10; i++) {
52             map.get("c1").buy();
53             map.get("c2").buy();
54         }
55         System.out.println(map.get("c1").getCoins()); //1
56         System.out.println(map.get("c2").getCoins()); //2
57         System.out.println(map.get("p1").getCoins()); //10
58     }
59 }

```

On this page there is a UML diagram and the code it represents. Please look at the code and diagram and then answer the questions below.

d. (1 point) The coupling in this design is:

Circle one: HIGH or LOW

Why? _____

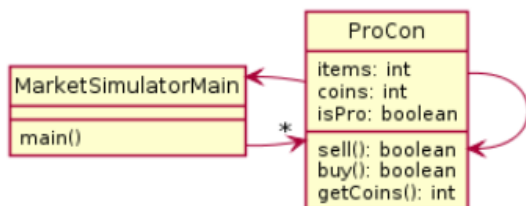
e. (1 point) The cohesion in this design is:

Circle one: HIGH or LOW

Why? _____

f. (1 point) What line(s) of code does the solid line and open arrow going from MarketSimulatorMain to ProCon correspond to?

g. (1 point) What line(s) of code does the solid line and open arrow going from ProCon to ProCon (self loop) correspond to?



On this page there is a UML diagram and the code it represents. Please look at the code and diagram and then answer the questions below.

h. (1 point) The coupling in this design is:

Circle one: HIGH or LOW

Why? _____

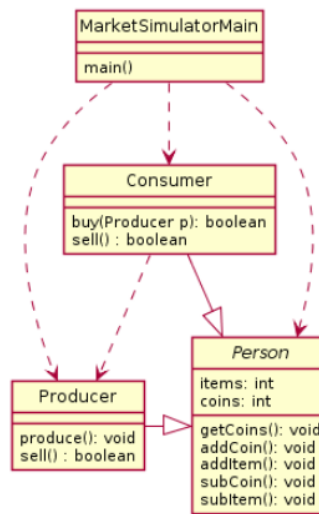
i. (1 point) The cohesion in this design is:

Circle one: HIGH or LOW

Why? _____

j. (1 point) What line(s) of code does the dashed line and open arrow pointing from Consumer to Producer correspond to?

k. (1 point) What line(s) of code does the solid line and closed arrow pointing from Consumer to Person correspond to?



```

1 public class MarketSimulatorMain {
2     public static void main(String[] args) {
3         new MarketSimulatorMain();
4     }
5     public MarketSimulatorMain() {
6         Producer p1 = new Producer(0,10);
7         Consumer c1 = new Consumer(6,0);
8         Consumer c2 = new Consumer(7,0);
9         for (int i=0; i < 10; i++) {
10             c1.buy(p1); c2.buy(p1);
11         }
12         System.out.println(c1.getCoins()); //1
13         System.out.println(c2.getCoins()); //2
14         System.out.println(p1.getCoins()); //10
15     }
16 }
17 abstract class Person {
18     private int items, coins;
19     public Person(int coins, int items) {
20         this.items = items; this.coins = coins;
21     }
22     public void subCoin() { this.coins--; }
23     public void subItem() { this.items--; }
24     public void addCoin() { this.coins++; }
25     public void addItem() { this.items++; }
26     public int getCoins() { return this.coins; }
27     public int getItems() { return this.items; }
28 }
    
```

```

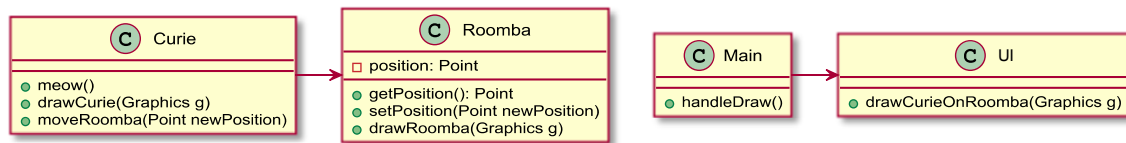
class Consumer extends Person {
    public Consumer( int coins, int items) {
        super(coins, items);
    }
    public boolean buy(Producer p) {
        if (getCoins() > 0 && p.sell()) {
            addItem(); subCoin(); return true;
        }
        return false;
    }
    public boolean sell() {return false;}
}
class Producer extends Person{
    public Producer(int coins, int items ) {
        super(coins, items);
    }
    public void produce() { addItem(); }
    public boolean sell() {
        if (getItems() > 0) {
            subItem(); addCoin(); return true;
        }
        return false;
    }
}
    
```

This page intentionally left blank.

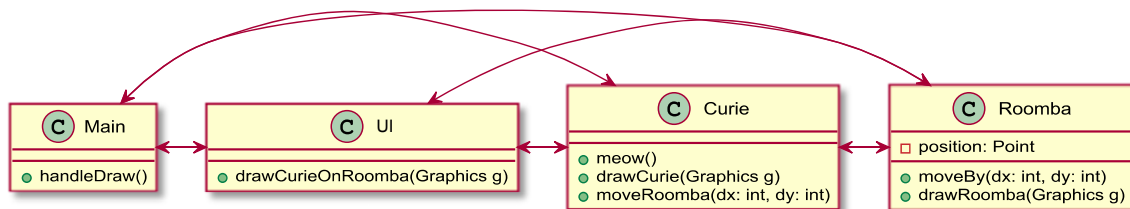
2. (10 points) This problem is a design exercise. First, read the problem description below. Then answer the questions.

Curie Cat is curious about her zoo's new Roomba, which is a small robot vacuum cleaner. She wants to reprogram it so she can ride it around the zoo. She also wants to enable her fans to watch her ride her Roomba from her User Interface (UI). Can you help Curie identify the problems with her two designs and come up with a better design?

a. (3 points) Explain the problems with this solution using your design principles.



b. (3 points) Explain the problems with this solution using your design principles.



c. (4 points) Make a UML diagram of your proposed solution to the problem.

3. (4 points) Consider the following code.

```
public static int parse(String value1, String value2 ) {
    int v1=0;
    int v2=0;
    try {
        v2 = Integer.parseInt(value2);
        v1 = Integer.parseInt(value1);
    } catch (NumberFormatException e) {
        System.out.println( "Non-number passed in!");
    }
    int result=0;
    try {
        result = exceptionalDivision(v1, v2);
    } catch (ArithmeticException e) {
        System.out.println( "Calculation error!");
        result = -99;
    }
    return result;
}

public static int exceptionalDivision(int x, int y) throws ArithmeticException {
    if (x == 0 && y== 0) {
        throw new IllegalArgumentException();
    }
    if (y == 0) {
        throw new ArithmeticException();
    }
    return x/y;
}

public static void main(String[] args) {
    String[] a = { "x", "1", "21", "0" };
    String[] b = { "5", "0", "3", "0" };
    for (int i=0; i<4; i++) {
        try {
            System.out.println(parse(a[i], b[i]) );
        } catch (IllegalArgumentException e) {
            System.out.println( "Main caught");
        }
    }
}
```

What would this code print?

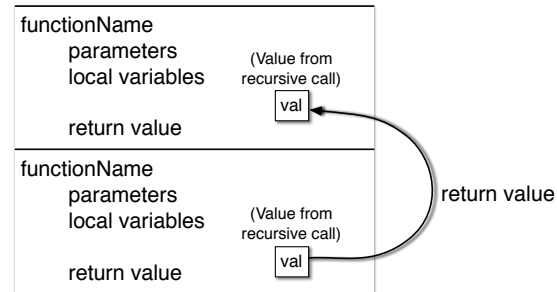
4. (10 points) For this problem, use the frame technique we practiced in the course to trace the execution of the recursive function call. Start your trace with the first call to mystery on line 14. A frame template is provided for your reference.

Once you are finished, answer the question at the bottom of the page.

```

1  public static int mystery(int x, int y, int z) {
2      x = x * 5;
3      y = y / 10;
4      if (x > y) {
5          System.out.println( "x: " + x + " y:" + y);
6          return z;
7      }
8      return mystery( x, y, z+1);
9  }
10
11 public static void main(String[] args) {
12     System.out.println( mystery(1, 10000, 0) );
13 }

```



For the code above, *what would the final output be?* _____

5. (15 points) Consider the following related declarations. @Override annotations are omitted to save space:

```
public interface Note {  
    void play();  
    void sing();  
}  
  
public class Do implements Note {  
    public void play() {  
        System.out.println("C");  
    }  
  
    public void sing() {  
        System.out.println("do");  
    }  
}  
  
public class Re extends Do  
    public void play() {  
        super.play();  
        System.out.println("D");  
    }  
  
    public void shout() {  
        this.sing();  
        System.out.println("RE!");  
    }  
}
```

```
public class Mi extends Re  
    public void play() {  
        System.out.println("E");  
    }  
  
    public void sing() {  
        System.out.println("mi");  
    }  
  
    public void harmonize(Note other) {  
        other.sing();  
        this.sing();  
    }  
}
```

- a. (4 points) Draw a UML diagram to represent the given interface and classes. Include all methods, but when writing subclass methods, only show a method on the subclass if the subclass method overrides the parent class's method, or if the method is specific only to the subclass. In places where lines representing fields are appropriate, use lines and do NOT re-list the same field in the field list.

- b. (11 points) Continuing the same problem, suppose we declare and initialize these variables:

```
Do a = new Do();
Do b = new Mi();
Re c = new Re();
Mi m = new Mi();
Note n = new Mi();
```

For each line of code below, if the line results in an error, **circle** the appropriate error; otherwise, provide the output in the provided blank. If the code works but does not print anything, write “nothing”. Consider each line of code separately. That is, if a line would give an error, then assume that line doesn’t affect any others. If the result would print on multiple lines, remove the newline from your result and show it on a single line.

Code	Either circle the error or provide the output		
b.sing();	<i>runtime error</i>	<i>compile error</i>	_____
c.sing();	<i>runtime error</i>	<i>compile error</i>	_____
n.sing();	<i>runtime error</i>	<i>compile error</i>	_____
b.play();	<i>runtime error</i>	<i>compile error</i>	_____
c.play();	<i>runtime error</i>	<i>compile error</i>	_____
m.harmonize(a);	<i>runtime error</i>	<i>compile error</i>	_____
Note x = new Note();	<i>runtime error</i>	<i>compile error</i>	_____
b.shout();	<i>runtime error</i>	<i>compile error</i>	_____
c.shout();	<i>runtime error</i>	<i>compile error</i>	_____
((Mi) n).shout();	<i>runtime error</i>	<i>compile error</i>	_____
((Mi) c).play();	<i>runtime error</i>	<i>compile error</i>	_____
