# CSSE 220---Object-Oriented Software Development

## Exam 2 -- Part 1, October 23, 2019

This exam consists of two parts. Part 1 is to be solved on these pages. If you need more space, please ask your instructor for blank paper.

*Allowed Resources on Part 1*: You may use a single sheet of 8.5” x 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.

**You will have two back-to-back class periods to complete Part 1.**

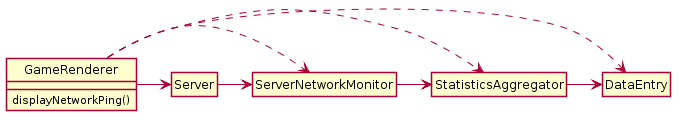
**Part 2 will be completed in the next class.**

Please, begin by writing your name on every page of the exam. We encourage you to skim the

entire exam before answering any questions.

|  |  |  |
| --- | --- | --- |
| **Problem** | **Points**  **Possible** | **Earned** |
| 1 | 6 | \_\_\_\_\_\_\_ |
| 2 | 8 | \_\_\_\_\_\_\_ |
| 3 | 10 | \_\_\_\_\_\_\_ |
| 4 | 5 | \_\_\_\_\_\_\_ |
| 5 | 15 | \_\_\_\_\_\_\_ |
| **Paper Part Subtotal** | 44 | \_\_\_\_\_\_\_ |
|  |  |  |
| **Computer Part Subtotal** | 56 | \_\_\_\_\_\_\_ |
|  |  |  |
| **Total** | 100 | \_\_\_\_\_\_\_ |

1. (6 points)



Consider the following method chain in GameRenderer's displayNetworkPing method:

double ping = server.getNetworkMonitor().getStatistics().getEntry("ping").asDouble()

The UML above shows the classes involved.

1. (2 points) Given what you know about method chains and the diagram above, would you call this primarily an issue with coupling or cohesion? Explain why in a sentence or two.
2. (4 points) Now, use a UML to propose a change that fixes the problem you identified in the previous part.

2. (8 points)

At an investment company bank there are 2 kinds of accounts, regular accounts and VIP accounts. Both accounts allow deposits and withdrawals but each does something different monthly.

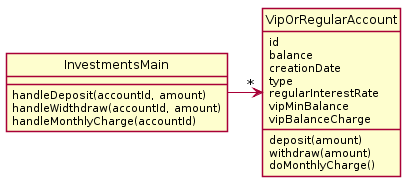
Every month, the regular account balance is decreased by a small percentage, stored as a negative interest rate which is set when the account is created. Every month, the VIP account is checked to see it is above the minimum balance - if the balance is above minimum there is no fee, but if the account balance is below minimum a fixed balance fee is subtracted from the account. Both the minimum balance and the balance fee are set when the VIP account is created.

Both accounts also charge an annual fee, which is assessed as part of the fees during the month that the account was created. For that reason, the system must store the creation date for each account. Regular accounts charge $10 per year, VIP accounts charge $100/year.

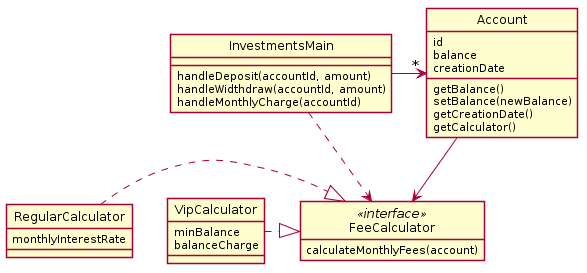
Here are 2 possible solutions. You can assume both of these designs function correctly (that

is, exclude principle 1 from your consideration).

Solution A

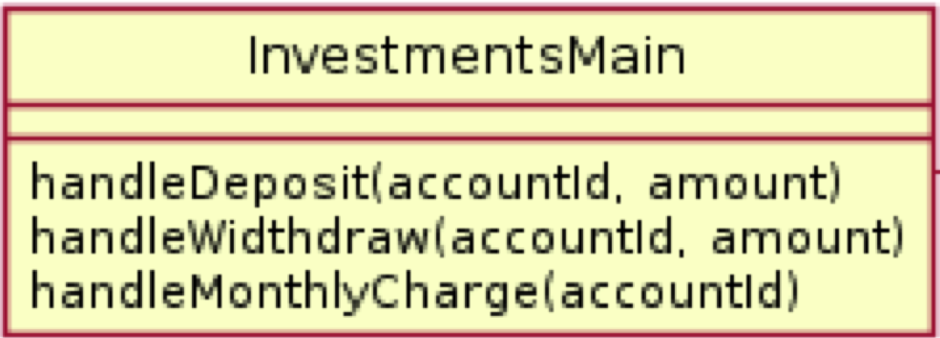


Solution B



\* Note this diagram does not show how RegularCalculator and VipCalculator objects get created. Don't worry about that issue.

1. (2 points) Explain the problems with Solution A using your design principles.
2. (2 points) Explain the problem with Solution B using your design principles.
3. (4 points) Make a UML diagram of your proposed solution to the problem. For your solution we have provided an InvestmentsMain to get you started. Feel free to omit any regular getter methods in your solution diagram as well.



3. (10 Points)

For this problem, determine the output from this program by tracing the call to operation *mystery*.

**public** **static** **void** main(String[] args) {

**int**[] a = {1, 9, 5};

**int**[] b = {2, 2, 2};

**int** result = *mystery*(a, b, **false**, 0);

System.***out***.println("Result is: " + result);

}

|  |  |
| --- | --- |
| **public** **static** **int** mystery(**int**[] a, **int**[] b, **boolean** afterNine, **int** curIndex) {  **if**(curIndex == a.length-1)  **return** a[curIndex];  **if**(a[curIndex] == 9) {  curIndex++;  **return** *mystery*(a, b, **true**, curIndex);  } **else** **if**(afterNine) {  **int** current = a[curIndex] \* b[curIndex];  **return** current + *mystery*(a, b, afterNine, curIndex + 1);  } **else** {  **int** current = a[curIndex] + 1;  **return** current + *mystery*(a, b, afterNine, curIndex + 1);  }  } |  |

4. (5 points)

Write what this code outputs.

|  |
| --- |
| **public** **class** Tester {  **public** **static** **void** main(String[] args) {  **try** {  *doSomething(*);  } **catch** (ArithmeticException e) {  System.***out***.println("No divide by 0!");  }  }  **public** **static** **int** helper(**int** a, **int** b) {  **if**(b == 0)  **throw new** ArithmeticException();  **else** **if** ((a%b) != 0)  **throw new** IllegalArgumentException();  **else**  **return** a/b;  }  **public** **static** **int** worker(**int** a, **int** b) {  **int** result = 0;  **try** {  result= *helper*(a, b);  System.***out***.println("Result is " + result);  } **catch** (IllegalArgumentException e) {  System.***out***.println("Did not divide evenly");  }  System.***out***.println("Finished worker");  **return** result;  }  **public** **static** **void** doSomething() {  *worker*(1, 3);  *worker*(9, 3);  *worker*(2, 0);  }  } |

Write your answer here:

5. (15 Points)

|  |
| --- |
| **public** **interface** Snap {  **void** print1();  **void** print2();  **void** snapper(**int** a);  }  **public** **class** Crackle **implements** Snap {  @Override  **public** **void** print1() {  System.***out***.print("1");  }  @Override  **public** **void** print2() {  System.***out***.print("2");  **this.**print1();  }  @Override  **public** **void** snapper(**int** a) {  System.***out***.print(a+1);  **this.**print2();  }  }  **public** **class** Pop **extends** Crackle **implements** Snap {  @Override  **public** **void** print1() {  System.***out***.print("One");  }  @Override  **public** **void** snapper(**int** a) {  System.***out***.print(a);  **super.**print1();  }  **public** **void** popper() {  System.***out***.print("POP!");  }  } |

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.

b. (11 points) 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**

Snap a = **new** Crackle();

Snap b = **new** Pop();

Crackle c = **new** Pop();

Crackle d = **new** Crackle();

Pop p = **new** Pop();

Output:

1. Snap s = **new** Snap(); runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Snap s2 = **new** Pop(); runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. a.print2(); runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. ((Crackle) c).print1(); runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. b.snapper(3); runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. b.print2(); runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. b.popper(); runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8. ((Pop) b).popper(); runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9. c.print1(); runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10. ((Snap) p).popper() runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11. ((Pop) d).popper(); runtime error compiler error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_