

COP 3330 – Object Oriented Programming in Java
Exam 1 Review

Date: 9/21/2026

Time: 12:00 pm – 1:15 pm (come early)

Location: HS1-119 our classroom

Outline of course material thus far:

1. Basic Program Setup
 - a. Required class name, file name
 - b. Required main method signature
 - c. Optional imports
2. Printing
 - a. System.out.println
 - b. System.out.print
 - c. Use of + with strings for string concatenation
 - d. Escape sequences
3. Variables
 - a. Primitive Types – (byte, short, int, long), (float, double), boolean, char
integer types real nums
 - b. Assignment Statement
4. Arithmetic Expressions
 - a. Usual PEMDAS
 - b. Pay attention to integer division
 - c. Pay attention to mod
 - d. Know how to force the computer to do either integer division or floating pt division.
 - e. Pay attention order of statements when solving problems.
5. If statement
 - a. Four types of syntax (if, if-else, if-else if-else if..., if-else if-else if-else)
 - b. Use of block of statements {}
 - c. Java has a boolean type and its implications
 - d. &&, || and !
 - e. Pay attention to difference between one if-else if... and separate ifs
 - f. Look at general problem solving molds with decision making
6. Loops
 - a. While loop
 - b. For loop
 - c. Do-while loop
 - d. Use of block of statements {}
 - e. Nesting of loops to solve problems
 - f. break and uses
 - g. continue and uses

7. Math class

- a. static methods only
- b. How to call static methods
- c. Don't call methods that return something on a line by themselves
- d. abs, cbrt, ceil, cos, exp, floor, log, log10, max, min, pow, sin, sqrt, tan, toDegrees, toRadians

8. Random class

- a. constructors – default, seed, corresponding behavior.
- b. How to call instance methods
- c. nextBoolean(), nextDouble(), nextGaussian(), nextInt(int bound)

9. String class

- a. Strings are immutable
- b. difference between reference and object
- c. How to make a reference point elsewhere
- d. Use of String methods
- e. charAt, compareTo, compareToIgnoreCase, concat, contains, endsWith, equals, equalsIgnoreCase, indexOf – 2 versions, replace, substring – 2 versions, toLowerCase, toUpperCase
- f. Many methods create new String object based on object method was called on.

10. Static methods

- a. How primitives are passed as parameters.
- b. How references are passed as parameters.
- c. How to incorporate method calls in code (classname.methodname if not in class)
- d. How to return different types from a method.
- e. Role in program design.

11. One Dimensional arrays.

- a. How to declare
- b. How to fill
- c. How to access elements
- d. Two ways to loop through elements
- e. How to pass to methods and return from a method
- f. Use of frequency array for letters

12. Two Dimensional arrays

- a. How to declare
- b. How to fill
- c. How to access elements, access rows
- d. How to incorporate into method calls
- e. How to look at adjacent locations
- f. Use in grid games

Exam Format

Short Answer

Coding Questions – write code segments, fill in solutions to full programs, won't have to write boring template code (public static void main...)

Other

75 points

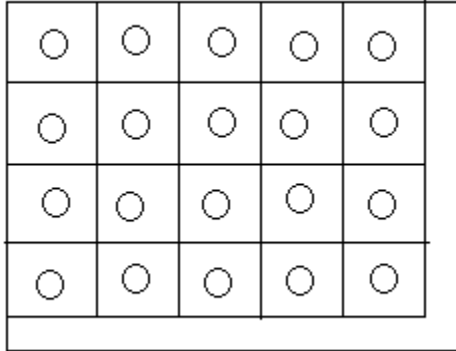
Quiz Aids

None

Math class, Random class and String class documentation that is necessary plus a couple random extra methods will be provided

Practice Quiz Questions

1) You want to have a program that calculates the number of pepperonis to place on rectangular pizzas. For each pizza, you want the user to be able to choose the “density” of pepperonis according to the following design:



Essentially, each pepperoni is roughly centered in a square of the pizza of some given size (perhaps 35 mm by 35 mm). We portion the rectangular pizza into as many rows and columns as we can, of this fixed size. Each of these square cells of the pizza get one piece of pepperoni. Depending on the dimensions of the pizza, some room may be leftover on the right side and bottom of the pizza that are too small to form the squares desired. These areas do not get any pepperonis. Write a program that takes in the length and width of the pizza in millimeters, as well as the size of the bounding square for one pepperoni, in millimeters, and calculates the total number of pepperonis needed for the pizza.

2) Write a program that determines whether or not Anna and Bob were at the restaurant at the same time. Ask the user when Anna and Bob arrived at and left the restaurant, respectively. (The user should answer in number of minutes after midnight, and assume that both arrived and left on the same day.) Determine whether or not Anna and Bob were at the restaurant at the same time, and if so, for how many minutes they overlapped. For example, if Anna arrived at 720 minutes after midnight and left 800 minutes after midnight and Bob arrived at 600 minutes after midnight and left 740 minutes after midnight, they were at the restaurant together for 20 minutes.

3) Write a program that takes in the user's first and last names and creates a single string object that stores their name in the format last comma first. For example, if they enter "Anne" and "Bridges", create a string object that stores "Bridges, Anne". For practice, explicitly call the concat method. Also create a string that has the first two letters of the first name and first letter of the last name, all uppercase as the person's monogram.

4) Write a program to play a game of marbles. The game starts with 32 marbles and two players. Each player must take 1, 2 or 3 marbles on their turn. Turns go back and forth between the two players. The winner is the person who takes the last marble. Your program should prompt each player with a message that states the current number of marbles and asks them how many they'd like to take. Continue until there is a winner. Then your program should print out the winner (either player #1 or player #2.) (Incidentally, if both players play optimally, who always wins? What is their strategy?)

5) Write a static method that takes in a String *str* and a character *c*, and returns the number of times *c* appears in *s*. For example, `count("hello", "l")` should return 2. Write the method and call it with several test cases you make on your own. Here's the method prototype:

```
public static int count(String str, char c);
```

6) One algorithm to find an approximation to the square root of a positive real number *x* is to run a binary search of the answer for 1 fixed number of iterations. The way this works is by setting initial low and high bounds for the search. If $x > 1$, then the low bound is 1 and the high bound is *x*. If $x < 1$, then the low bound is *x* and the high bound is 1. For each iteration of the search, guess that the square root is halfway in between low and high. Then see if the guess is too low or too high. For example, if $x = 4.0$ initially, the low and high start off as 1 and 4, respectively. Our first guess is $(1+4)/2.0 = 2.5$. Notice that $2.5^2 > 4$. This means that the highest possible value for the square root is 2.5. (Whatever the square root is, it's less than 2.5.) If you repeat this process 100 times, each time either changing the low bound or the high bound to the previous guess, then the low and high bounds will be extremely close to each other and either can be used as the approximation to the answer. The method prototype is given below. Write the method and then call it with various test cases to see how well it works:

```
public static double mysqrt(double x);
```

7) Write a static method that takes in a reference to an integer array, *arr*, and returns the number of inversions in the array. An inversion in an array, *arr*, is an ordered pair (*i*, *j*) such that $i < j$ but $arr[i] > arr[j]$. Test your method on several arrays.

```
public static int numInversions(int[] arr);
```

8) Write a program that reads in information about a set of pizza orders from standard input and calculates the total price of each order. In particular, the input file will be formatted as follows:

1. The first line of the input file will contain a single positive integer, *n* ($n < 100$), specifying the number of different pizza orders.
2. The next *n* lines will each contain one positive real number specifying the price for that corresponding pizza. (The first line will have the price for pizza 0, the next line for pizza 1, etc.)
3. The following line will contain a single positive integer *k* ($k < 50$) representing the number of orders you have to evaluate.
4. The next *k* lines will contain information about each of the *k* orders with one order per line.
5. Each of these lines will contain *n* non-negative integers, representing how many of those pizzas, respectively are in the order. (For example, if $n=5$ and the line contains the values 0 0 6 0 9, then the order contains 6 slices of pizza #2 and 9 slices of pizza #4.)

For each of the *k* test cases, output a line with the following format:

On day X, you will spend \$YY.YY on pizza.

where X is the test case number (starting with 1), and YY.YY is the price of the pizza displayed with exactly 2 digits after the decimal. (Note: The price may exceed 100 dollars, so YY simply represents a dollar amount and not the exact number of digits in that dollar amount.)

Sample Input File (pizza.txt)

```
5
3.00
3.50
4.50
5.00
6.00
3
1 1 1 1 1
0 0 2 1 6
1 0 3 2 3
```

Sample Output (Corresponding to Sample Input File)

```
On day 1, you will spend $22.00 on pizza.
On day 2, you will spend $50.00 on pizza.
On day 3, you will spend $44.50 on pizza.
```

9) Write a program that asks the user for the number of rows and columns in a grid, as well as the number of bombs they would like printed on the grid. Use this information to make a 2D character array where each character is set to '_' or '*'. Use randomization to set the number of '*' characters in the grid exactly equal to the number of bombs the user entered. Your program should create a static method with the prototype below and call it from the main method:

```
public static char[][] makeRndGrid(int rows, int cols, int numbombs);
```

(The method generates a 2d character array with rows number of rows and cols number of columns, filling it with exactly numbombs '*' characters and the rest equal to '_'.)

10) Add to the code from #3 so that instead of having an '_' characters, these characters are replaced with characters representing digits such that each value stored represents the number of adjacent squares storing bombs. For example, for a grid with 5 rows and 5 columns, you could print out the following grid:

```
* 3 * 2 1
2 * 3 * 1
1 2 3 2 1
0 1 * 1 0
0 1 1 1 0
```