Com S 227
Fall 2020
Topics, review problems, and process for Exam 1
Thursday, September 24, 8:15 pm

General information

This will be a 60-minute, timed exam, administered on Canvas and proctored via Zoom. No books, no notes, no calculators, no headphones, no collaboration. The problems will primarily involve writing Java code or reading and interpreting Java code.

There seems to be no perfect way to conduct exams for an online class. We want to ensure fairness, as much as possible, and we want to be available during the exam in case you have questions. Which has led us to...

Exam format

- Students will be divided up by name into groups of approximately 40. Each group will take the exam in a virtual Zoom room with two TAs or instructors. These groups and rooms will be set up and assigned a few days before the exam.

- You will need to participate in the Zoom meeting using a smartphone, with the camera and microphone on for the entire duration. You'll need to position the phone so we can see you working, and you'll need to be in a quiet room by yourself.

- You'll need to show your ID before you start.

- The exam will be taken on Canvas using the Respondus lockdown browser. Once you start the exam, you will not have access to any other websites or applications on your computer until you complete the exam.

- The exam problems will involve writing Java code. You'll be typing the code into a text window in Canvas (i.e., without the help of Eclipse or the Java compiler).

- If you have a question, raise your "hand" (in Zoom) and the TA will invite you to a breakout room.

Alternative arrangements

If you do not have access to a smartphone, or if there are other technical reasons why you cannot take the exam as described above, the main alternative will be to take an exam on campus at the testing center. This will be a slightly different exam, also on Canvas but with 50-minute time limit, and there will be no way to ask questions during the exam. The testing center can

accommodate only a limited number of our students, so *please participate in the regular Thursday night exam session if you are able.*

If you will need to use the testing center, **you must let your instructor know BY MONDAY, SEPTEMBER 21.** Likewise if you are not physically in Ames, and can't participate via smartphone, please discuss the situation with your instructor **in advance** and we will arrange something.

> *(Note: Students with an accommodation letter from Student Accessibility Services will take the exam at the Exam Accommodation Center, and will be contacted separately by the EAC to schedule it.)*

Do this immediately

1. *Check Zoom:* Make sure you have the Zoom app on your phone. Join a test meeting using your phone at https://zoom.us/test and make sure the microphone and camera are working. Make sure you know how to "raise your hand".

2. *Check Location:* Figure out where you're going to take the exam and how you will prop up your phone so that you are visible. You will need to be in a private room, at a desk or table that is clear except for your phone and computer. It needs to be a quiet place, because you'll have your microphone on during the exam. Make sure your phone is set up to use the local wifi.

3. *Check Canvas:* We will create a no-credit "sample exam" on Canvas that you should try beforehand. This is mainly to make sure you can successfully use the Respondus lockdown browser. Try it with the computer that you intend to use when you take the exam.

Summary of Exam topics

The exam covers everything we have done in class through conditionals and boolean expressions This corresponds pretty well to the first 4 chapters of the zyBook. The list below is a rough overview.

- Using variables and assignment statements
- Basic string operations and concatenation
- Primitive types, arithmetic operations, integer vs floating-point math
    - We are only concerned with primitives **`int`**, **`double`**, and **`boolean`** and **`char`** at this point
- Using constants
- Writing static methods, calling static methods
- Objects and classes
- Using constructors and methods
- Using API documentation
- Defining our own classes
    - Defining methods, parameters, and return types
    - Defining constructors

    - Using instance variables for object state, public vs private
    - instance variables vs local variables
- Unit testing a class using a simple main() method (NOT JUnit)
- Conditional statements and boolean expressions
- Boolean operators
- Reading input or strings with Scanner

The exam *does* include material from lab but will NOT include anything specifically about Eclipse, JUnit, or the debugger. You do not need to memorize anything from the Java API. You should be familiar with how to use methods of String, Scanner, Random, and Math, but we'll provide one-sentence method descriptions for methods we've used.

Strategy

When you first read an exam problem, ask yourself: What kind of problem am I being asked to solve?

- Am I tracing execution for some existing code, or writing my own code?

- Am I being asked to write an isolated static method, or define an entire class?

- If it's a static method, is it supposed to print output, or return a value? Is it supposed to be a "main" method? Is is supposed to read input?
    - When writing code to solve a problem, always start with a concrete example or "test case". Normally this involves some hand calculation, and often *the steps of the hand calculation show you how to write the code*. For example, in problem 10 below, start by hand-calculating the result for 150 copies. You can mimic the same steps when writing the code.

- If you are defining a class,
    - Distinguish which methods are *mutators* and which are *accessors*
    - What instance variables are needed? Look at the accessor methods: *What information do they need to return? How will that information be stored in the object?*
    - Remember that accessor methods should not modify the instance variables
    - Always start with a few simple test cases: *If you construct an instance and then call a mutator method, what changes do the accessor methods observe?*

After writing your code, try to read it. Does it make sense to you? Trace through it by hand. and make sure it is really doing what you intended at every step.

How do I study?

You can become a good at problem-solving and coding with practice! Lots of it!

For exam preparation, always start by typing your code in an ordinary text editor or Word, NOT in Eclipse, since during the exam you will not have access to Eclipse.

After you solve a practice problem, paste your code into Eclipse and try it, using your concrete examples as test cases.

If you weren't successful, get some help from the TA or instructor or a fellow student. But ask yourself: *What information can I stash in my brain to make it easier the next time?* In the future, how can I remember the way to solve this kind of problem? How can I recognize a similar problem? Can I make up more problems that are like this one?

More practice?

There are more problems you could try at the the end of the zyBook. There are many more in the supplemental text, if you can get your hands on an old copy of Java Concepts or Big Java.

Another entertaining way to practice Java problem-solving is the interactive site http://codingbat.com/java. The sections Warmup-1, String-1, Logic-1, and Logic-2 are probably the relevant ones for this exam. These are useful to practice working with Java and using conditional statements. *However, please note that the problems on this site will NOT help you understand how to implement classes or use instance variables, which is a major portion of Exam 1. See Chapter 3 of the zyBook.*

Some practice problems

1) Suppose that the following is a method of some class C:

```
public boolean foo(int x, int y)
{
  boolean result = false;
  if (x > y)
  {
    if (y != 0)
    {
      result = true;
    }
  }
  if (x == 0)
  {
    result = true;
  }
  return result;
}
```

Assuming that x is an instance of class C, give the return value for each of the calls:

```
x.foo(2, 1)
x.foo(0, -1)
x.foo(1, 1)
```

2) Complete the right-hand side of each assignment by writing an expression satisfying the stated requirement. (In some cases there is more than one correct answer. *Do not write any additional lines of code, just write an expression.*) The first one is done for you.

(a) A boolean value indicating whether the int variable **x** is an odd number

```
boolean isOdd = (x % 2 != 0);
```

(b) Given an int named **eggCount**, the number of full cartons of 12 eggs you can make with that many eggs

```
int fullCartons =
```

(c) Given an int named **eggCount**, the number of eggs left over after putting them into full cartons of 12 eggs

```
int leftOver =
```

(d) Given a String **str**, the last character of the string

```
char lastChar =
```

(e) A boolean value indicating whether the Strings **s1** and **s2** contain the same text

```
boolean areTheSameString =
```

(f) A boolean value indicating that the string s contains at least four characters.

```
boolean hasAtLeastFour =
```

(g) Given a integer number **d** of dollars and **c** of cents, a **double** that is the dollar value of **d** dollars and **c** cents

```
double totalValue =
```

(h) An integer i converted to a String

```
String s =
```

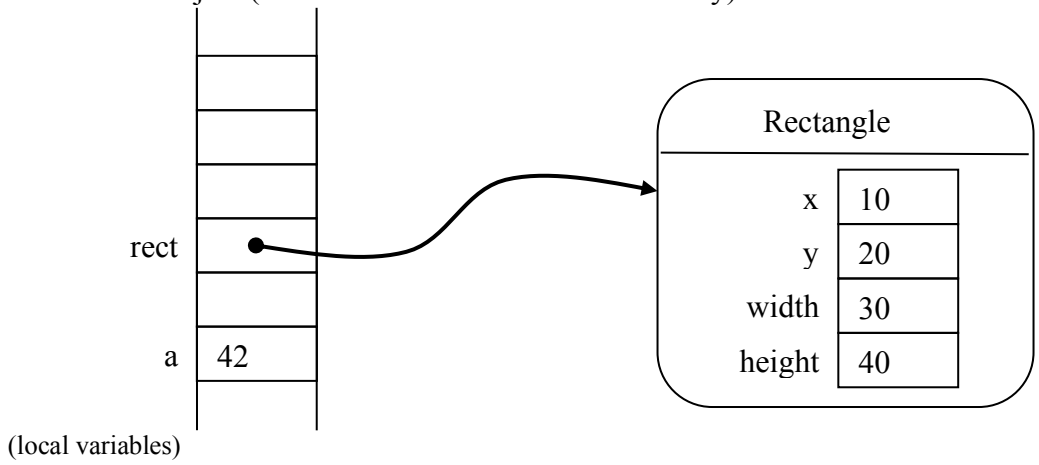
(i) An integer containing the whole number part of a **double d**

```
int i =
```

3) Assume that the following statements are executed in some method:

```
int a = 42;
Rectangle rect = new Rectangle(10, 20, 30, 40); // (x, y, width, height)
```

We can then sketch a "memory map" showing the values of the local variables at this point. For a primitive type like **`int`**, the variable stores the actual value. For an object, the variable stores a *reference* to the object (which is somewhere else in memory).



Then, trace execution of the remaining statements below. Sketch what the memory map looks like after all statements have executed, and show what the output is from the **`println()`** statements:

```
Rectangle rect2 = new Rectangle(2, 4, 6, 8);
int b = a;
Rectangle rect3 = rect;
rect3.setWidth(99);
rect2.setX(137);
b = b + 5;
System.out.println(a);
System.out.println(b);
System.out.println(rect.getWidth());
```

4) Write a method **`createID`** declared as follows:

```
public static String createID(String firstName, String lastName) {…
```

It should return a string consisting of the first initial, followed by the last name, followed by a randomly generated number between 1 and 50 (all lower case). E.g. for input "Steve Kautz" the return value would look like "skautz42". The number should be generated with an instance of java.util.Random, using the nextInt() method. (Note that since this method is static and needs no instance variables, we don't have to worry here about what class it is defined in.)

5) Consider a class **`ParkingMeter`** that models a coin-operated parking meter. We will assume it only takes quarters. Its public interface includes:

- A method **`insertCoin(int howMany)`** that simulates adding the given number of quarters. Inserting coins increases the time remaining on the meter, but not more than the maximum time.
- A method **`getTimeRemaining()`** that returns the time remaining on the meter in minutes as an **`int`** value.
- A method **`passTime(int minutes)`** that simulates the passage of time, that is, reduces the time remaining (but not below zero)
- A method **`getTotal()`** that returns the total amount of money, in dollars, collected by this meter.
- A constructor **`ParkingMeter(int minutesPerQuarter, int maximumTime)`**, where **`minutesPerQuarter`** is the number of minutes you get for a quarter, and **`maximumTime`** is the maximum number of minutes. A newly created **`ParkingMeter`** has no time on it.

a) Write a class **`MeterTest`** with a main method that tests the **`ParkingMeter`** class above under the following scenario:

A new parking meter is constructed with 15 minutes per quarter and a maximum of 60 minutes
Three quarters are inserted
20 minutes passes
(*)
Four quarters are inserted
90 minutes passes
(*)

At the points marked (*), your test should check the values returned by the accessor methods. Print out the *actual* value obtained from calling the method, and then print out the *correct* value that you expect.

b) Write a complete implementation of the **`ParkingMeter`** class described above

6) Implement a class **`Loan`** that models a loan on which monthly payments are made. A loan is created with an *annual interest rate*, given as a decimal value (e.g. "6% per year" is given as .06). At any given time the loan has a *balance*, the amount that is still owed. Each time a payment is made, the interest owed for one month is calculated and added to the balance, and then the amount of the payment is subtracted. (If the balance is negative, no interest is added.) The class should have the following as its public interface:

- A constructor allowing the interest rate and initial balance to be specified.
- A method **`makePayment`** that correctly adjusts the balance according to the given payment amount. The method has one **`double`** parameter, the amount of the payment.

(continued on next page)

- A method **`getBalance`** that returns the current balance.
- A method **`getPayoffAmount`** that returns the amount that would be required to pay off the balance at the next payment (balance plus one month's interest)

Example: A loan is constructed with an initial balance of $1000.00 and an interest rate of 0.24. After calling **`makePayment(100.0)`**, the new balance is 920.00. (One month's interest is .02 * 1000.00, or 20.00. Note that .02 is one-twelfth of the *annual* interest rate of .24.) If **`getPayoffAmount()`** is now called, it returns 938.40, which is the balance of 920.00 plus 18.40 interest (.02 * 920.00).

7) The code below is an attempt to implement the class ParkingMeter described in problem 5. It violates some best practices for using instance variables (for example, see the section "More about grading" on pp 10-11 of the hw1 pdf, and/or the September 9 entry of the Section A/B topics page (front page link #6)). As a consequence, this code sometimes works, but sometimes yields incorrect results. a) How are the guidelines for instances variables being violated? b) Give at least two different test cases in which you would get incorrect results.

```
public class ParkingMeter
{
  private int minutesPerQuarter;
  private int maxTime;
  private int timeRemaining;
  private int quartersAdded;
  private int totalQuarters;

  public ParkingMeter(int givenMinutesPerQuarter, int givenMaximumTime)
  {
    minutesPerQuarter = givenMinutesPerQuarter;
    maxTime = givenMaximumTime;
  }

  public void insertCoin(int howMany)
  {
    quartersAdded = howMany;
  }

  public int getTimeRemaining()
  {
    timeRemaining = timeRemaining + (quartersAdded * minutesPerQuarter);
    timeRemaining = Math.min(timeRemaining, maxTime);
    return timeRemaining;
  }

  public void passTime(int minutes)
  {
    timeRemaining = timeRemaining - minutes;
    timeRemaining = Math.max(timeRemaining, 0);
  }

  public double getTotal()
  {
    totalQuarters = totalQuarters + quartersAdded;
    return totalQuarters;
  }
}
```

8) The class below is supposed to model a door with a lock (can only be opened if it is not locked). What is the main problem with this implementation? Explain briefly and fix it.

```
public class Door
{
  // Locks the door (it can be locked whether open or not)
  public void lockDoor()
  {
    boolean isLocked = true;
  }

  // Unlocks the door (it can be unlocked whether open or not)
  public void unlockDoor()
  {
    isLocked = false;
  }

  // Closes the door (it can be closed whether locked or not)
  public void closeDoor()
  {
    boolean open = false;
  }

  // Opens the door, only if it is not locked.
  public void openDoor()
  {
    if (!isLocked)
    {
      open = true;
    }
  }

  // Determines whether the door is open or not
  public boolean isOpen()
  {
    return open;
  }
}
```

9) Eggs are are sold by the "flat" (30 eggs), by the dozen (12 eggs) or by the half dozen (6 eggs). Suppose they cost 6.50 per flat, 3.00 per dozen, and 2.00 per half dozen for regular eggs, and 20% more for brown eggs. Write an interactive program (with a main() method) that prompts the user to enter a number of eggs, indicate by typing "yes" or "no" whether they are brown, and then prints out the number of flats, dozens, and half dozens that should be purchased to get at least that number of eggs for the lowest price, followed by the price. (Don't worry about formatting the decimal places.) A sample interaction might be like this (user's response is in **bold**):

```
How many eggs? 100
Do you want brown eggs (yes/no)? yes
3 flats
1 dozens
0 half dozens
Price 27.0
```

10) A copy center charges 15 cents per copy for the first 10 copies, 12 cents per copy for the next 100 copies, and 8 cents per copy after that. (Example: for 150 copies it's .15 * 10 + .12 * 100 plus .08 * 40 = 16.70.) Write a static method that returns the cost in cents for a given number of copies. (Since a static method does not use any instance variables, we don't need to worry here about what class the method is in.)

```
public class AnyClass
{
  public static int findCopyCost(int numCopies)
  {
    // TODO
  }
}
```

11) Suppose you have defined variables

```
int x = 42
String y = "lunchtime"
```

a) Evaluate each expression below

```
y.equals("breakfast") || x != 42

x == 5 || (y.length() > 0 && !(x <= 0))

!((x > 0 && x > 1) || x > 2)
```

b) Using the variables x and y above, for each phrase below, write a Java boolean expression that captures its meaning. Then determine whether the expression is true or false using the values of x and y above.

x is at least 25
x is between 25 and 50, inclusive (i.e., including the endpoints of the interval)
y is either 10 or 12 characters long
x is equal to the three times the length of y
the length of y is not divisible by 3
x is negative or else x is even and divisible by 3
y is not equal to the string “dinner”

c) The method foo() from Problem 1 has three conditional statements (“if” statements). Rewrite that method so that it produces the same results, but using NO conditional statements (or any loops or tenary expressions or switch statements, etc).