Com S 227
Spring 2019
Topics and review problems for Exam 1
Monday, February 25, 6:45 pm

Exam locations by last name:

A-G	Hoover 2055	
H-Ma	Kildee 125	***** You must bring your ID to the exam *****
Mc-S	Troxel 1001	
T-Z	Lago W142	

General information

This will be a 70-minute, timed, pencil-and-paper written exam. No books, no notes, no electronic devices, no headphones, no collaboration. The problems will primarily involve writing Java code or reading and interpreting Java code.

Summary of Exam topics

The exam covers everything we have done in class through conditionals and boolean expressions and in the labs through Lab 4. This corresponds pretty well to the first 5 chapters of the textbook (not including the sections on graphics). The list below is a rough overview. Numbers in parentheses refer to the *current* edition of the text. (If you are using a previous edition, see this document: http://www.cs.iastate.edu/~cs227/section_number_mapping.pdf)

- Using variables and assignment statements (2.2)
- Basic string operations and concatenation (2.3, 4.5, and Lab 2)
- Primitive types, arithmetic operations, integer vs floating-point math (4.1, 4.2 and 5.2)
    - We are only concerned with primitives **`int`**, **`double`**, and **`boolean`** and **`char`** at this point
- Using constants (4.1)
- Objects and classes (2.1)
- Using constructors and methods (2.3, 2.4)
- Using API documentation (2.6)
- Defining our own classes (3.1 – 3.3)
    - Defining methods, parameters, and return types
    - Defining constructors
    - Using instance variables for object state, public vs private (3.1)
    - instance variables vs local variables (3.1, 3.6)
- Unit testing a class using a simple main() method (NOT JUnit) (2.7, 3.4)
- Conditional statements and boolean expressions (5.1 – 5.4)
- Boolean operators (5.7)
- Reading input or strings with Scanner (4.3)
- How to call a static method (4.2)

How to prepare

The most important thing you can do to prepare for an exam like this is to *practice solving problems and writing code*. You should practice with a pencil and paper, since that is the format of the exam. You can always check your work by typing up what you've written in Eclipse. We encourage you to post and discuss your sample solutions in Piazza. We do not post "official" solutions.

You will need to write quickly and accurately and to recognize correctly written code without the help of Eclipse. You will *not* be expected to write comments or documentation, nor define symbolic constants for numbers, nor use import statements.

You do not have to memorize methods from the Java API. You should know `System.out.print` and `System.out.println` . You should be familiar with use of `String`, `Scanner`, `Math`, and `Random`, but for specific methods from these classes that might be needed, we'll provide you with the one-sentence descriptions from the API.

For more practice

Remember that there are many, many more problems to practice on in the textbook besides the sample problems shown below. Each chapter has a number of "self-check" questions at the ends of the sections and a section of "Review exercises" at the end of each chapter.

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. *Please note that the problems on this site will NOT help you understand how to implement classes or use instance variables, which IS covered on this exam.*

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;
}
```

continued on next page

Problem 1, continued
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. Assume the following declarations:

```
double x = 2.4;
int j = 13;
String w = "food";
boolean goofy = false;
```

For each expression in the left-hand column below, give its *value* and its *type* (int, double, boolean, etc.) in the second and third columns. If what's written is not a valid Java expression, just write "error" in both columns. The first one is done for you.

Expression	Value	Type
`x > 0`	`true`	`boolean`
`j / 3`		
`j % 3`		
`j + 1 = j`		
`j / 10.0`		
`w + w`		
`j + 1 != j`		
`j + ""`		
`j.length()`		
`w.length()`		

3) 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 =
```

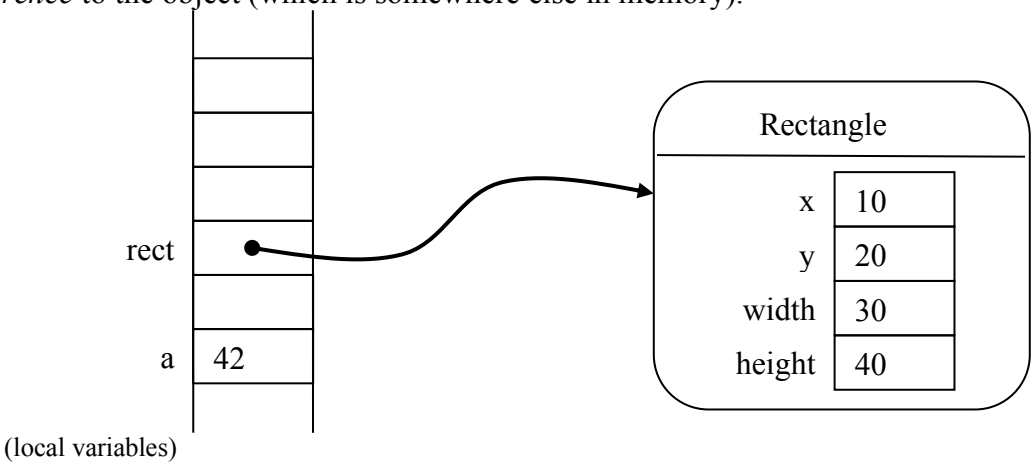
(i) An integer containing the whole number part of a **`double d`**

```
int i =
```

4) 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());
```

5) 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.)

6) 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

7) 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.
- 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).

8) The code below is an attempt to implement the class ParkingMeter described in problem 6. It violates some best practices for using instance variables (for example, see the section "More about grading" on page 9 of the hw1 pdf, and/or the February 8 entry of the topics page here: http://web.cs.iastate.edu/~smkautz/cs227s19/topics.html). 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;
  }
}
```

9) 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;
  }
}
```

10) 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
```

11) 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
  }
}
```

12) 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 without using any conditional statements (or any loops or tenary expressions or switch statements, etc).