

# Exam 1 Strategies, Tips and Practice Problems

## General information

This will be a 75-minute, timed, pencil-and-paper written exam. No books, no notes, no electronic devices, no headphones, no collaboration. The problems will primarily involve writing, reading, and interpreting Java code.

## Summary of Exam topics

The exam covers everything we have done in class through conditionals and boolean expressions (no loops). This corresponds 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 o instance variables vs local variables
- Unit testing a class using a simple main() method (NO JUnit questions will be asked)
- Conditional statements and boolean expressions
- Boolean operators
- Reading input or strings with Scanner

The exam *does* include material covered in the labs, 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 a summarized Javadocs of the the relevant Java libraries for you to refer to.

# 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 it 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 5 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 in the likely format of the exam you will not have access to Eclipse.

After you solve a 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?

One way to practice Java problem-solving is the interactive site <http://codingbat.com/java>  (<http://codingbat.com/java>). For this exam, you should be comfortable with solving the problems in:

- Warmup-1
- String-1
- Logic-1
- Logic-2

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 textbook.**

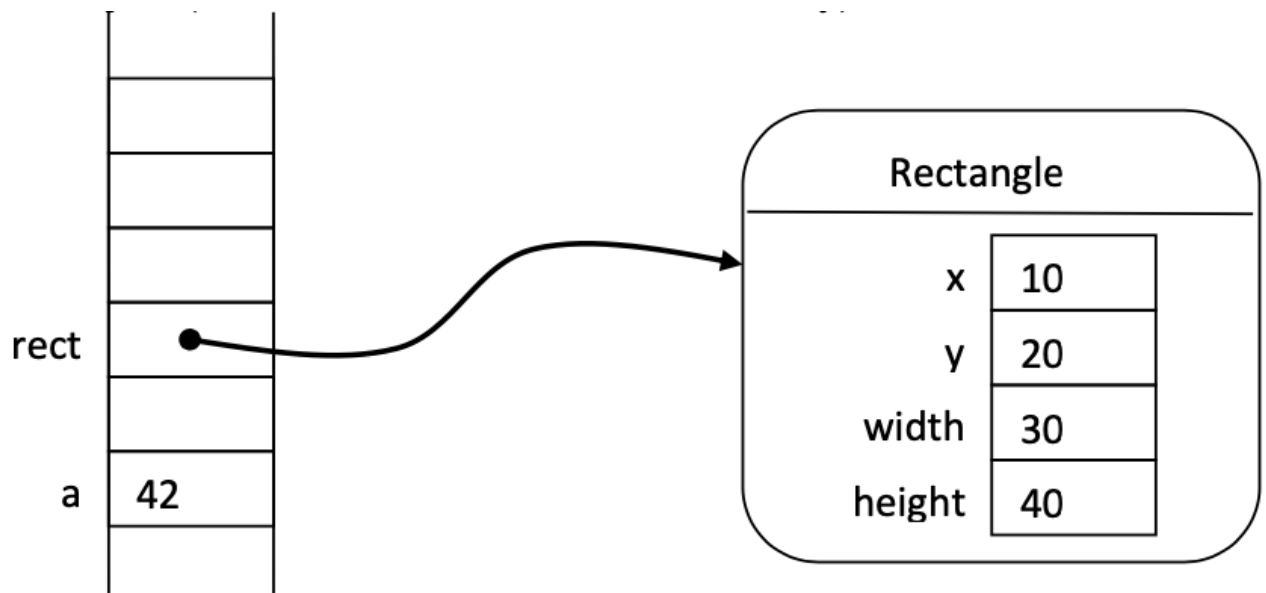
There are more problems you could try at the end of the zyBook, specifically the "extra stuff" chapters. Other good textbooks, you may be able to find an old copy of, that have plenty of practice problems are *Java Concepts* and *Big Java*.

## Some practice problems

1) 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).



(local variables)

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());
```

2) 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 (all lower case), followed by a randomly generated number between 1 and 50 (inclusive). 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.)

3) 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 ( $.02 * 920.00$ ).

4) The code below is an attempt to implement the class ParkingMeter from the practice exam. It violates some best practices for using instance variables. 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;
    }
}
```

5) 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

**6)** 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;
    }
}
```

**7)** 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
    }
}
```

**8)** 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"