APCS Problem Set 6: Flow Control via Conditionals

Michael Ferraro
mferraro@balstaff.org
Balboa High School
27 October 2015

Contents

1 Introduction

Back in Problem Set #4, you learned how to write programs that didn't run in a sequential manner. Such programs had either iterative loops using `while()` or involved recursion. In this problem set, you'll learn how to affect a program's flow using `if()` statements.

Sometimes, `if()` statements are very simple, as you've already seen:

```
if ( a == 5 ) {
    System.out.println("Hey, a is five!");
}
```

Other times, `if()` can execute a block of code to run when the condition it's testing is false; that's when `else` comes in. And then there's a specialized `if()/else` implementation in Java called `switch/case/break`. You'll be learning more about these in the required reading.

The concepts you'll learn in this problem set are crucial in Computer Science. Without the ability to write software that can act in a particular manner based on certain conditions (what we humans refer to as *decision making*), computers would be far less useful!

1.1 Objectives

By completing this problem set, students will learn to...

- use `if()/else` statements;
- evaluate and write `boolean` expressions;
- apply Java's order of operations, which includes relational and logical operators;
- use `switch/case/break`; and
- implement the rules of a system (game) in Java.

1.2 Due Date

This problem set is due **before 5th Period on 10 November 2015**. Make sure that you are keeping up with the sections you're learning about in class so that you have enough time for the programming assignments toward the end.

2 Basic Conditionals Using `if()/else`

2.1 Required Reading

→ `Litvin & Litvin,` *`Java Methods A & AB`*`: §§6.1-6.4`

2.2 Checking for Understanding

Now that you've read about how `if()/else` works, answer the following. If you're not entirely sure of an answer, try **compiling and running** the code!

1. If the code below runs, what is printed to the screen?

```
int b = -3;

if ( b == 3 ) {
    System.out.println("apple sauce");
}
```

2. What is the console output for the code below?

```
boolean b = 2 < 0; //remember:  Java evaluates RHS of an assignment,
                   //           or everything to the right of '='

System.out.print("My favorite color is ");

if ( b ) {
    System.out.println("green.");
} else {
    System.out.println("red.");
}
```

3. Consider the two snippets below and answer the questions that follow.

`int p = 2;`
`int q = (int)2.836;`
 	
`if ( p == q ) {`
`   System.out.println( p + q );`
`}`
	`int p = 2;`
	`int q = (int)2.836;`
	
	`if ( p == q )`
	`   System.out.println( p + q );`

(a) Are both syntactically correct (i.e., do they both have proper Java syntax)? If not, explain which one is malformed.

Reason, if not:

(b) Will there be printed output for the snippet on the left? If so, exactly what value is displayed?

Output, if so:

2.3 Book Problems

1. Ch. 6, #2
2. Ch. 6, #5: Before attempting to implement the `totalWages()` method, **complete the table on the paper form** so you adequately understand various workweek/pay scenarios that involve standard pay rate, overtime pay rate, and workweeks that include and exclude overtime.

save progress

3 More Advanced Conditionals

3.1 Logical Operators

3.1.1 Required Reading

→ Litvin & Litvin, *Java Methods A & AB*: §6.5

3.1.2 Relationship Between Java's Logical Operators and Symbolic Logic

In Geometry, you might have been taught *symbolic logic*, which uses symbols to represent operations like `and`, `or`, and `not` as $\wedge$, $\vee$, and $\neg$, respectively.

Using letters to represent simple statements (e.g., p means ``it will rain'' and $q$ means ``I will stay dry''), we can analyze truth values for logical statements using truth tables[1]. Consider the compound statement ``It is not true that it will rain and I will stay dry,'' which can be represented symbolically as $\neg(p \wedge q)$:

p	q	$p \wedge q$	$\neg(p \wedge q)$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

It turns out that when you make the statement ``It will not rain or I will not stay dry,'' you are logically saying the same thing. Examine the rightmost columns of the tables above and below; they show the same truth values!

p	q	$\neg p$	$\neg q$	$\neg p \vee \neg q$
T	T	F	F	F
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

DeMorgan's Rules allow for the rewriting of statements in a manner that preserves logical meaning under all circumstances. The two preceding truth tables show that the truth values of the two compound statements, (i) $\neg(p \wedge q)$ and (ii) $\neg p \vee \neg q$, are logically equivalent. We can therefore write $\neg(p \wedge q) \equiv \neg p \vee \neg q$. Another logical equivalency under DeMorgan's Rules is this one: $\neg(p \vee q) \equiv \neg p \wedge \neg q$. Each of these equivalencies can be thought of as a distribution[2] of the negation over the statement in parentheses, which negates both p and q, as well as switching logical `and` to `or` and vice versa.

The following are translations between symbols used in symbolic logic and their equivalent representations in Java:

Operator	Symbol	Java
`and`	$\wedge$	`&&`
`or`	$\vee$	`||`
`not`	$\neg$	`!`

Below is a table showing the DeMorgan's translations in symbolic form and then in the Java form that you will be using:

Symbolic Form	Java Equivalent

$\neg(p \vee q) \equiv \neg p \wedge \neg q$	`!( A || B )` $\equiv$ `!A && !B`
$\neg(p \wedge q) \equiv \neg p \vee \neg q$	`!( A && B )` $\equiv$ `!A || !B`

3.2 Order of Operators

3.2.1 Required Reading

→ `Litvin & Litvin, *Java Methods A & AB*: §6.6`

3.2.2 Relating New Operators to Prior Operators

Before learning any Java, you already had a notion of an order of operations from math, PEMDAS. With the exception of parentheses, all of those operators -- $\times$, $\div$, $+$, and $-$ -- are *binary operators*, meaning that each takes two operands (or arguments). *Unary operators* are those operators like $!$ (negation) that take only one operand.

Here is Java's order of operations, presented in a slightly different manner than in the textbook:

Priority	Operator Type	Operators
highest	**P**EMDAS	$()$
	Unary	$!$, $-$[1], *casts*[2], $++$, $--$
	P**E**MDAS	x^y
	PE**MD**AS	$\times$, $\div$ (*also modulo,* $\%$)
	PEMD**AS**	$+$, $-$
	Relational Operators	$<$, $<=$, $>$, $>=$, $==$, $!=$
	Logical `and`	`&&`
lowest	Logical `or`	$\|\|$

[1] Negation for numerical values -- not subtraction -- it is equivalent to multiplication by -1.

[2] E.g., `(int)` and `(double)`.

3.3 Short-Circuit Evaluation and More `if()/else`

6.1 Required Reading

→ `Litvin & Litvin, *Java Methods A & AB*: §§7.7-7.8`

3.4 Book Questions

1. Ch. 6, #7
2. Ch. 6, #8 (*Hint: If you want to be sure that your answer is truly equivalent to the starting statement, consider using a truth table.*)
   (a)
   (b)
3. Ch. 6, #9
   (a)
   (b)
4. Ch. 6, #10 (*Hint: You can take advantage of Short-Circuit Evaluation here. Also, consider which values of x might cause an exception to occur.*)
5. Ch. 6, #11 (*Hint: To avoid issues where ``a/b'' yields zero or results in a loss of precision -- e.g., $8 \div 3$ evaluating to 2 rather than $2\frac{2}{3}$ -- see if the products of the diagonals of $\frac{a}{b} = \frac{b}{c}$ are equal.*)
6. Ch. 6, #12a (*Hint:* `!( a > b )` $\equiv$ `a <= b`, *which is similar to multiplying both sides of an inequality by* -1.)
7. Ch. 6, #15

save progress

4 Programming Exercises

For each of the following exercises, you will need to make sure the classes can accept an integer argument from the command line. This is so that an automated tester can ``grade'' your code more easily. An example of a class taking an integer argument on the command line is provided in class `CmdLineArgsEx`.

Make sure your code is checked in to your GitHub repository in the specified project folder by the due date -- or else you may receive no credit for these exercises!

4.1 Odd or Even?

1. While at school, create a new project called `ps06a` in `workspace1`. *Capitalization matters, since the autotester won't find your code if you call your project, say,* `PS6A`*!*
2. Import `OddOrEven.java` from http://feromax.com/apcs/problemsets/PS06/downloads/OddOrEven/ into your project.
3. Perform an initial push of this project to GitHub using the `github-push.sh` script at school.
4. Modify the `main()` method in class `OddOrEven` to accept an integer argument on the command line. If the provided integer is odd, your class prints `ODD` and a newline -- i.e., `System.out.println("ODD")` -- **and nothing else**. Likewise, it prints `EVEN` for an even integer. If you print anything else, the tester might not recognize your program's response and fail you! *Precondition: Your program will be presented with integers, only, but not zero.*
5. Perform a final push of your code to GitHub. As verification that you did this OK, log in to https://github.com, check that you see a folder called `ps06a/src`, and have a file named `OddOrEven.java` within that folder.

The score from the autotester will be recorded on your paper form by the teacher.

4.2 Positive, Negative, or Zero?

1. While at school, create a new project called `ps06b` in `workspace1`.
2. Import `PosNegZero.java` from http://feromax.com/apcs/problemsets/PS06/downloads/PosNegZero/ into your project.
3. Perform an initial push of this project to GitHub using the `github-push.sh` script at school.
4. Modify class `PosNegZero` to accept an integer argument on the command line. Based on the input, print `POSITIVE`, `NEGATIVE`, or `ZERO`, in all caps, followed by a newline character. *Precondition: Your program will be presented with integers, only.*
5. Perform a final push of your code to GitHub. As verification that you did this OK, log in to https://github.com, check that you see a folder called `ps06b/src`, and have a file named `PosNegZero.java` within that folder.

The score from the autotester will be recorded on your paper form by the teacher.

save progress

5 `Craps` Lab, Part I

You are going to be writing part of a GUI application that allows a user to play the game of *Craps* (without any betting). This activity maps to Litvin §6.9 and §6.12.

This activity will be completed in two parts spanning multiple class periods. Some parts will be worked on in class, combined with teacher instruction, and other parts are to be completed by you outside of class for homework.

5.1 Know the Rules of the Game

1. Read the first page of Litvin §6.9, which describes how the game works. (You will probably need to read the

second paragraph more than once unless you're already familiar with the game!) In *words*, summarize the rules of the game.

2. Now, take the rules and arrange them into a flow chart. You enter the flowchart by making your first roll, and there are two ultimate exits: Winning or losing.

   Complete this problem on the paper form.

5.2 Let's Play! (And Learning More About JAR Files)

Litvin provides a complete Java application for you to play with. Rather than providing a set of Java class files, however, the program has been distributed as a Java JAR[3] file.

1. Download `Craps.jar`.
2. Using the terminal shell or command prompt, change to the directory containing `Craps.jar`. Then, issue this command:
   ```
   jar tvf Craps.jar
   ```
   What you'll see is a listing of the files contained within the archive. The JAR file is very similar to a ZIP file, though the contents aren't compressed.[4]
3. You can run the program by issuing this command in the directory containing `Craps.jar`:[5]
   ```
   java -jar Craps.jar Craps
   ```
   Take this opportunity to play the game a few times to see if your understanding of the rules matches what you wrote earlier.

A little later on, when you're writing your own applications with multiple classes, you'll see how you can use Eclipse to create JAR files that you can use to distribute your own programs to others (friendly, non-malware apps, of course!).

5.3 Design Considerations

The author's rationale for the design of `Craps` is explained in the subsequent parts of Litvin §6.9, but we will be discussing these ideas in class. Once you've received the lesson about the application's overall design and how the various parts ``talk'' to one another, **read the rest of Litvin §6.9**.

5.4 Implementing Your Part

As mentioned in the text, you are to implement *part* of the program, as two other software engineers have their respective parts to work on.

For any Java sources the book refers to, you may download them from http://feromax.com/apcs/problemsets/PS06/downloads/Craps/.

1. Create a new project called `ps06c`.

2. Work on Litvin §6.9, #1 (see p157). You are going to implement the basic rules of the game using `if()/else` statements. If your flowchart from earlier is correct, it may prove useful here. Once finished, demonstrate the execution of `CrapsTest1` to at least one other classmate and then demonstrate for your teacher.

   Complete this problem on the paper form.

3. Per Litvin §6.9, #2, write the `Die` class, providing the methods that Litvin tells you about earlier in the section.

   Modification: When you write your tester class, you are to use a `for()` loop to cause 100 back-to-back rolls of the die to occur. The output your tester class generates **must** look like this:

```
Roll #1: 4
Roll #2: 1
Roll #3: 6
...
Roll #100: 2
```

Once finished, demonstrate your tester class for your teacher.

Complete this problem on the paper form.

4. Now run `CrapsStats` against your classes, per Litvin §6.9, #3. If things are correct thus far, you should have ≈ 49 wins for every 100 plays. Demonstrate the `CrapsStats` program working for your teacher while the number of plays is 100,000 -- which should yield a result of ≈49,300 wins.

Complete this problem on the paper form.

save progress

6 `switch/case/break` and `enum`

6.1 Required Reading

→ `Litvin & Litvin, *Java Methods A & AB*: §§6.10-6.11, §6.13`

6.2 Programming Exercise

Write a program that does the following.

- Asks the user for an integer between 1 and 5, inclusive.
- Checks that the integer provided is really in the range from 1 to 5.
  - Integer is outside of the range? Generate an error message and ask again for an integer.
  - Integer is valid? Continue...
- Using `switch/case/break`, generate the appropriate message per the table below.

Input Value	Output Text
1	`Alice`
2	`Ben`
3	`Charlie`
4	`Denny`
5	`Ethel`

- Return to the integer-entry step.

Once the program is working, demonstrate it for your teacher.

Complete this problem on the paper form.

save progress

7 `Craps` Lab, Part II

1. Read Litvin §6.12
2. What is a `stub class`? What is it used for?
3. Copy `RollingDie.java` and implement the `drawDots()` method.
4. Download the sources you don't already have -- `ControlPanel.java`, `Craps.java`, `CrapsTable.java`, & `DisplayPanel.java` -- from `http://feromax.com/apcs/problemsets/PS06/downloads/Craps/source/` and run the `Craps` class. If everything you've done is correct, your application should run like the original `Craps.jar`.

Complete this problem on the paper form.

Footnotes

... tables[1]
If you've never worked with truth tables, please ask about how they work in class.

... distribution[2]
cf. the *Distributive Property of Multiplication over Addition*

... JAR[3]
JAR stands for *Java archive*.

... compressed.[4]
Actually, a JAR file is most like a TAR file in Unix/Linux, which stands for *tape archive*. When data is backed up from a filesystem and written to a tape, the files would basically be concatenated together into one very large file, with extra information stored -- file names; directories and names of their contents, sizes, file/directory owners; etc. - and then written to a tape in a serial manner. Remember that tapes are sequentially accessed, so *serializing* the filesystem's data is necessary for storage on tape.

...`Craps.jar`:[5]
Note that you may omit the classname `Craps` and simply type `java -jar Craps.jar` since there's a file inside the JAR called `MANIFEST.MF`, which is in the `META-INF` directory. That file has a line in it that says ``` `` ````Main-Class: Craps`'', and that tells Java which class to run if you don't specify one.

Michael Ferraro 2015-10-24