APCS Problem Set 3: Java Style and Syntax

Michael Ferraro
mferraro@balstaff.org
Balboa High School
16 September 2015

Contents

1 Introduction

Style in programming is important, but more than just for the ``look'' of your code. When your classmates or teacher read code that you write, we (should) have certain expectations: That code inside of a block is indented appropriately, that names for methods have capital letters where we expect them, etc. Proper style makes it easier for all of us (you included!) to understand your code. In the end, properly formatted code helps us to be more efficient programmers by decreasing time spent debugging and fixing errors.

1.1 Objectives

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

- what *syntax* means in the context of programs,
- syntax for the Java programming language,
- accepted Java-specific style guidelines, and
- how to use the `javadoc` utility to produces web pages explaining your code's key features.

1.2 Due Date

This problem set is tentatively due **before 5th Period on 22 September 2015**.

2 Java Keywords, Comments, & Capitalization

2.1 Required Reading

You might skim the questions in the next section (§2.2) prior to reading the sections listed below.

→ Litvin & Litvin, *Java Methods A & AB*: §§3.1-3.5

2.2 Rules of the Road

You just read a bit about the correct uses of comments, class/field/method naming conventions, reserved words in Java, etc. What follows are some checking-for-understanding questions.

1. Comments improve the [ ] of programs.
2. Java reserved words may include uppercase letters. [ ]
3. Names of classes, fields, and methods may...
   (a) contain a digit. [ ]
   (b) start with a digit. [ ]
   (c) include an underscore (_) character. [ ]

2.3 Questions

1. Ch. 3, #1: Name three good uses for comments.
   Number your response: **(1) ... (2) ... (3) ...**
   [ ]
2. Based on Ch. 3, #3: Consider the source code for `MovingDisk.java`, below.
   - There are 16 *reserved words* in the class; **box** all instances of them.
   - Put an **oval** around all *constants* you find.
   - Draw a **triangle** around all *method* names. Remember that constructors and `main()` are methods, too.
   - Any time the `new` keyword is used, **underline** the type of the object that is created. For example, you would underline the following statement like so:
     `String str = new String("hello.");` ← `String()` *needs to have a triangle around it!*

   Complete this problem on the paper form.

Code listing for `MovingDisk.java` from the Litvin textbook

```
import java.awt.*;
import java.awt.event*;
import javax.swing.*;

public class MovingDisk extands JPanel implements ActionListener {
    private int time;

    public MovingDisk() {
        time = 0
        Timer clock = new Timer(30, this);
        clock.start;
    }

    public void paintComponent(Graphics g) {
        int x = 150 - (int)(100 * Math.cos(0.005 * Math.PI * time));
        int y = 130 - (int)75 * Math.sin(0.005 * Math.PI * time));
```

```
            int r = 20;

            Color sky;
            if (y > 130) sky = Color.BLACK
            else sky = Color.CYAN;
            setBackground(sky);
            super.paintComponent(g);

            g.setColor(Color.ORANGE);
            g.fillOval(x - r, y - r, 2*r, 2*r);
        }

        public void actionPerformed(ActionEvent e) {
            time++;
            repaint();
        }

        public static void main(String args) {
            JFrame w = new JFrame("Moving Disk);
            w.setSize(300, 150);
            w.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            Container c = w.getContentPane();
            c.add(new movingDisk());
            w.setResizable(false);
            w.setVisible(true);
        }
    }
```

3. Ch. 3, #4: Select *syntax* or *style* for each.
   (a) [ ]
   (b) [ ]
   (c) [ ]
   (d) [ ]
   (e) [ ]
   (f) [ ]
   (g) [ ]
4. According to *information theory*, what is *redundancy*?
   [ ]
5. Ch. 3, #7: Select *syntax* or *style*.
   - Parentheses: [ ]
   - Curly Braces: [ ]

save progress

3 Statements and Blocks

3.1 Required Reading

→ Litvin & Litvin, *Java Methods A & AB*: §3.6 & §3.8

3.2 A Note on Curly Braces

When opening a block of code using an opening curly brace -- `{` -- you have two options, as mentioned in §3.6 of the textbook:

- The older, C-style way that you've seen in this course thus far:

```
public class Person {
    // code omitted...

    public void setWeight(int w) {
        // code omitted...
    }
}
```

- The more modern, Java way:

```
public class Person
{
    // code omitted...

    public void setWeight(int w)
    {
        // code omitted...
    }
}
```

Which you use is a matter of personal preference. Your teacher prefers the former since it makes fitting code onto slides (and the previous page) easier, as the code occupies fewer lines. Others prefer the latter since it makes blocks easier to identify quickly. To the trained eye, it makes little practical difference which way you choose -- just so long as you're consistent about which style you subscribe to!

3.3 Questions

1. Ch. 3, #9.

2. Ch. 3, #10: Indicate *true* or *false*, no explanation necessary.
   (a)
   (b)
   (c)
   (d)
   (e)

save progress

4 Javadoc: Create (Nearly) Self-Documenting Code!

By now, you should understand the different ways to add comments (and to ``comment out'' code so the compiler ignores it). There's ``//'' for use with a single line (or the rest of a line) and ``/* *comments here... */*,'' which spans one or more lines.

With a slight modification, the multi-line comment tags can be recognized by Javadoc as descriptions of what a method does, the arguments it takes, the value or object it returns, and its side effects. Javadoc creates web pages documenting your code for use by others (and by you, when you inevitably forget the details of your classes!).

4.1 MathGenie Example

4.1.1 Obtain, read, and run the code

Create a new project in workspace0 called ps03a. Download the files below and import them into your new project's src folder but **do not run** the driver class yet!

- MathGenie.java
- MathGenieDriver.java

Examine the sources you've imported, paying attention to the Javadoc comments. Once you're reasonably familiar with the code, run it.

4.1.2 Generating Javadoc documentation

Included here are instructions for running Javadoc at school using Linux (or Mac OS/X at home) and Windows (for those using Windows at home).

Linux & Mac OS/X Instructions: Do the following in a terminal shell.

```
cd <workspace0_location>/ps03a/src
mkdir doc
cd doc
javadoc ../*.java
ls

# Linux users only:
nautilus .

# Mac users only:
open .
```

Windows Instructions: At a command prompt, change to your ps03a project's src directory and run the commands given below.

```
cd <workspace0_location>\ps03a\src
md doc
cd doc
javadoc ..\*.java
dir
explorer .
```

Since the doc subdirectory of src is the *working directory* when calling javadoc, the files generated by javadoc will reside in doc. This way, your src directory won't have files that don't belong there, keeping your file hierarchy tidy.

Using your favorite web browser, open up the generated `index.html` file. As you look at the pages, pay attention to (a) the content of the `Javadoc` comments from the source code, and where those comments appear on the web pages; and (b) how the links at the top of the pages make it convenient for you to find *fields*, *constructors*, and *methods*.

Now, edit the source for the `MathGenie` class, making the field variable `mainNumber private`. Save the file and re-run `Javadoc`.

- After making `mainNumber` private and regenerating the Javadoc API page for class `MathGenie`, what change do you notice on the API page?

4.2 `Javadoc`: A Doorway to the Rest of Java

Now that you've seen what `Javadoc` produces, it's time to show you why knowing how to navigate the results pays off. On the *resources* page of the course website are two links:

- *APCS Java Subset Specification* by Owen L. Astrachan at Duke University:
  http://www.cs.duke.edu/csed/ap/subset/doc/
- *Java 8 API* by Oracle:
  http://docs.oracle.com/javase/8/docs/api/

Browse to the first link (and the second, time permitting). Find the API[1] for `String`. Once there, click on the *methods* link at the top of the page.

1. For a `String` object, there exists a method called `length()`.
   (a) What parameters must this method be called with?
   (b) What does `length()` return?
2. What's printed when the code snippet below runs?

```
String str = "This is a test.";
int strLen = str.length();
System.out.println(strLen);
```

4.3 Revisiting Problem Set #2

Your task now is to go back and add `Javadoc`-style comments to the class definitions and methods for these source files from PS #2:

- `BankAccount.java`,
- `CheckingAccount.java`, and
- `SavingsAccount.java`.

Create a new project in `workspace0` called `ps03b`. Populate the `src` folder with your source files from `ps02`. Alternatively, if you did not have working versions of the PS #2 classes, you may download and use the versions from here.

Make sure that all methods returning something have `Javadoc` comments that include `@return`. Likewise, make sure all methods taking parameters have `Javadoc` comments that include `@param` *`variable_name`*.

Once you've added the comments, run `Javadoc` to produce the docs. Show the completed `Javadoc` web pages to your

teacher for a sign-off.

Complete this problem on the paper form.

Footnotes

... API[1]
Recall from the course syllabus, API stands for *Application Programming Interface*.

Michael Ferraro 2015-09-14