**CS200 Java Documentation and Style Guidelines**

**Dean Zeller**

**Fall 2018**

***File Documentation (original work)***

Every program file submitted must have block documentation at the start of the file. This summarizes information about the overall program and contains your name and the name of the assignment. Any file without documentation will not receive full credit. Include all character formatting, including capitalization, indent, and the column of asterisks on the right of the block. Any fields with dashes-between-words indicates a field that must be entered by the programmer.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* title-of-assignment \*

\* \*

\* PROGRAMMER: your-name \*

\* CLASS: CS200 \*

\* ASSIGNMENT: Assignment assign-# \*

\* INSTRUCTOR: Dean Zeller \*

\* TA: Robert Carver \*

\* SUBMISSION DATE: date-of-submission \*

\* \*

\* DESCRIPTION: \*

\* description-of-program \*

\* \*

\* COPYRIGHT: \*

\* This program is copyright (c) 2018 your-name) and Dean Zeller. This is \*

\* original work, without use of outside sources. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

***File Documentation (citing references)***

If any external code is used, it must be properly documented within the CREDITS sections shown below, and with a modified copyright statement. If any external code is used and not documented, it could be considered plagiarism.

\* \*

\* CREDITS: \*

\* The following sources were used in completion with this program. \*

\* PROGRAM: program-name \*

\* PROGRAMMER: programmer-name \*

\* LOCATION: original-code-location \*

\* MODIFICATIONS: \*

\* modifications-to-original \*

\* \*

\* COPYRIGHT: \*

\* This program is copyright (c) 2018 your-name and Dean Zeller. This is \*

\* original work, making use of the above sources. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

***Method Documentation***

Every method needs its own block documentation. It is similar in format to the file documentation, but different information is necessary. The template for method documentation is as follows:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* method-name \*

\* \*

\* Purpose: purpose-of-program \*

\* Parameters: \*

\* list-parameters-and-purpose \*

\* Return Value: describe-return-value \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

***Syntax vs Style***

Syntax is very strict, and not up for interpretation. However, every programmer will develop their own “style” of programming, within the given program syntax. This is both a blessing and a curse. Programmer style allows for creativity in problem solving, but it can also cause problems when working with others. This course will differentiate between syntax and style. Even though code may be syntactically correct, it must also follow certain rules of style. While students can use any programming method to solve problems, their code must follow the rules of style listed below for full credit.

***Naming Conventions***

1. Variable Names

Use meaningful variable names to illustrate what is contained within. For multiple word variable names, use camelCaseNames, small letters interspersed with capital letters and no spaces. Do no start variable names with a capital letter, as that is reserved only for classes.

1. Class Names

Start all class names with a capital letter, to denote it is a class and not a variable.

1. Parameter Names

Oftentimes, parameters are immediately stored in attributes and then never used again. If a parameter name and purpose match an attribute, use the same name, but end with an underscore. For example, if an attribute is named this.startingCash, use startingCash\_ as the parameter name. Consequently, if a parameter is not used for this purpose, do not use the underscore.

1. Indentation and code blocks

In programming, visual organization is important. Always use indentation of 4 spaces per level, and make sure it is consistently used. When curly braces for code blocks, do so as shown below.

|  |  |
| --- | --- |
| Correct | Incorrect |
| public void xxxxx  {  if(xxxx)  {  xxxxx;  xxxxx;  }  else  {  xxxxx;  xxxxx;  }  } | public void xxxxx{  if(xxxx){  xxxxx;  xxxxx;}  else{  xxxxx;  xxxxx;}  } |

1. Use of the this command

Java has a this command that allows reference of attributes or methods within the object. In Java the use of this is syntactically optional, but is required for purposes of CS200. Be specific and use this whenever referring to any attribute or method. It is more readable and illustrates the point that it is referring to an attribute or method of that object, rather than a local variable.

***DemoObject***

public class DemoObject

{

//attributes

private int attribute1;

private double attribute2;

private String attribute3;

//constructor, parameters for all attributes

public DemoObject(int attribute1\_,double attribute2\_, String attribute3\_)

{

this.attribute1 = attribute1\_;

this.attribute2 = attribute2\_;

this.attribute3 = attribute3\_;

}

//constructor, parameters for all attributes

public DemoObject(int attribute1\_)

{

this(attribute1\_, 2.5, "Dean");

}

//constructor, parameters for all attributes

public DemoObject()

{

this(10, 2.5, "Dean");

}

//Get Methods

public int getAttribute1()

{

return this.attribute1;

}

public double getAttribute2()

{

return this.attribute2;

}

public String getAttribute3()

{

return this.attribute3;

}

//Set Methods

public void setAttribute1 (int attribute1\_)

{

this.attribute1 = attribute1\_;

}

public void setAttribute2(double attribute2\_)

{

//error checking

if (attribute2\_<0.0)

{

System.out.println("Warning: cannot set attribute2 to below 0.0.");

System.out.println("Attribute not changed");

return;

}

this.attribute2 = attribute2\_;

return;

}

public void setAttribute3(String attribute3\_)

{

this.attribute3 = attribute3\_;

}

//toString method

public String toString()

{

String result = "";

result += "+-----------------------+\n";

result += String.format ("| attribute1 %10d |\n", this.attribute1);

result += String.format ("| attribute2 %10.3f |\n", this.attribute2);

result += String.format ("| attribute3 %10s |\n", this.attribute3);

result += "+-----------------------+";

return result;

}

//other methods

public void addToAttribute2 (double addto)

{

this.attribute2 += addto;

}

public void clearAttribute3()

{

this.attribute3 = "";

}

public void countToAttribute1()

{

for(int i=0;i<this.attribute1;i++)

System.out.println(i+ "");

}

}

***DemoObjectTester***

public class DemoObjectTester

{

public static void main(String args[])

{

DemoObject demoA = new DemoObject (5, 2.3, "Steve");

DemoObject demoB = new DemoObject (10);

DemoObject demoC = new DemoObject ();

System.out.println(demoA.getAttribute1());

System.out.println(demoB.getAttribute2());

System.out.println(demoC.getAttribute3());

demoA.setAttribute1(7);

demoB.setAttribute2(5.1);

demoC.setAttribute3("Joanne");

demoA.countToAttribute1();

demoB.addToAttribute2(1.8);

demoC.clearAttribute3();

System.out.println(demoA);

System.out.println(demoB);

System.out.println(demoC);

}

}