Metropolitan State University, St. Paul, MN
ICS 372 Object-Oriented Design and Implementation
Coding Standards

1 Introduction

It is important that you properly document and format your programs. Every language has its own coding conventions and tere are several reasons why it is important to follow them. There is the high cost of software maintenance. It is rare that the original author would maintain a piece of software for its entire life. It is quite difficult for most people to remember their code if they haven't seen it for a long period of time. Programmers should create code that they are proud of.

In your ICS 372 programs, you must obey the following subset of conventions. As we write more complicated programs, I may ask you to follow more requirements.

2 Files

Every interface and class (unless it is internal) must be put in a separate file.

3 Documentation

Java programs can have three types of comments:

1. implementation comments that span multiple lines; they are enclosed between /* and */
2. implementation comments till the end of line; they begin with //
3. documentation comments, which are enclosed between /** and */

Implementation comments are meant for programmers who need to read and understand your code as opposed to documentation comments, which are meant to describe the specification of the code. Documentation comments can be extracted to HTML files using the javadoc tool.

You should provide a discussion of non-trivial or non-obvious design, but do not explain stuff that is clear from the code.

Every class and interface must have documentation comments before any Java code. Every method that is not a setter or a getter of a field must have documentation comments before the method header. These comments must explain the purpose of the method and the input parameters and the return type/value.

Do not put **unncessary** implementation comments between lines of code. But do put them where essential.

4 Identifiers

Variable Names: Variable names must begin with a lower-case letter and be in full words. Words after the one must have their first letter capitalized.

Here are examples of acceptable variable names:

```
count costOfItem gallonsPumped testSucceeded
```

Here are examples of unacceptable variable names:

```
Count itmCst gallonspumped tstSucceeded
```

Class and interface names must begin with a capital letter. For example, `Customer` is an acceptable class name, but `customer` and `Cstmr` are not.

Method names must begin with a lower-case letter.

All identifiers (variable, class, interface, method names) must employ full words and must have more than one character. The identifiers must convey the meaning of what they are used for in a very clear manner. So don't use variable names such as `n1` and `i` and `ijkl`.

5 Indentation and Alignment

1. Do not type very long lines. Try to limit them to 80 characters
2. If an expression does not fit on a single line, break it according to these general principles:
3. Break after a comma or before other operators.
4. Do not put blank lines between lines of code within a method or between fields.
5. Put one blank line between methods.
6. Put a single space before and after operators. There are exceptions: no space after the unary minus and no space before a comma. Do not put a space a left parenthesis or before a right parenthesis.
7. Prefer higher-level breaks to lower-level breaks: For example, if you have to break the expression

   ```
   (a + b * (c + d)) * (e + f)
   ```

   you should try to break it as

   ```
   (a + b * (c + d)) *
    (e + f)
   ```

8. Attempt to align the new line with the beginning of the expression at the same level on the previous line.
9. If the application of the above rules results in ugly-looking code, just indent with enough tabs that make the code look reasonable.
10. Indent code within a class by one tab.
11. Indent code within a method by one tab.

12. Indent code within a block by one tab.

13. Always follow an `if`, `else`, `while`, and `for` with a {.

14. Always put { in the same line as its associated `if`, `else`, `while`, `for`, `do`, `switch`, method name, class name, interface name, etc. **Here note that C and C++ conventions are different**. So code as follows.

```
public class MyClass {
   private int[] values =  new int[5];
   public MyClass(int value) {
       for (int index = 0; index < 5; index++) {
          values[index] = value;
       }
   }
}
```

and not like the following.

```
public class MyClass
{
   private int[] values =  new int[5];
   public MyClass(int value)
   {
       for (int index = 0; index < 5; index++)
       {
          values[index] = value;
       }
   }
}
```

You can simplify matters for yourself by having Eclipse format the code using the Eclipse standard Java formatting feature. For this, do the following. (See Figure 1 and the posted video.)

1. In Eclipse, click the menu Window and then click Preferences.

2. Click Java, Editor, Save Actions.

3. Click the options as shown in Figure 1 and click Apply.

```
int thatValue;
int thisValue = 10;
int count;
thatValue = 0;
count = Math.min(thatValue, thisValue);
```

Put a space before and after a binary operator other than comma. Separate tokens properly with a space. Refer to the code below for an example of a program with respect to these and other requirements.

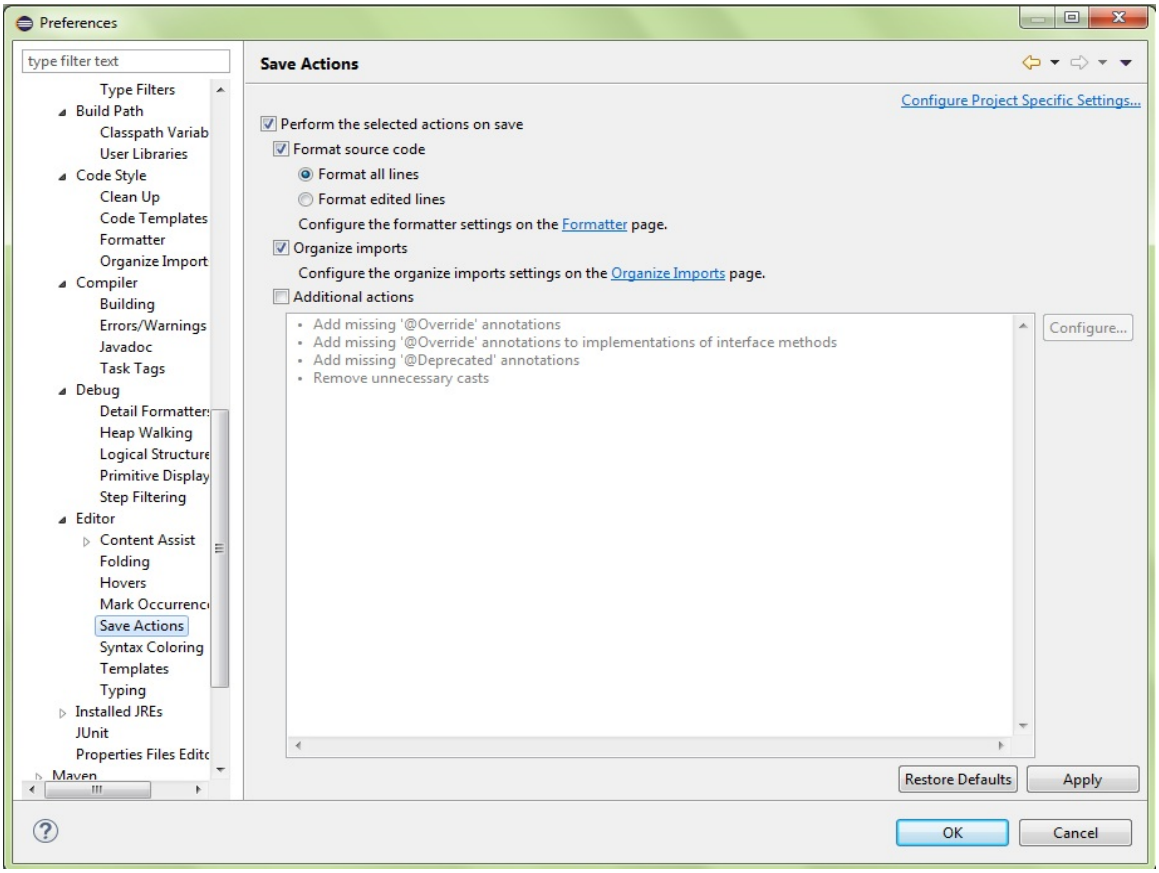

Figure 1: Auto-formatting using Eclipse

```
/**
    * This class implements a Set, which is a set.
    * The elements are integers. The purpose is to illustrate the
    * following:
    *     The notion of an ADT
    *     Efficient memory usage
    *     clone, toString, and equals methods.
    * @author Brahma Dathan
*/
import java.util.LinkedList;
public class LinkedSet implements Cloneable, Set {
/**
    * The elements are stored in a linked list.
*/
    LinkedList list = new LinkedList();

/**
    * Inserts an element into the Set object.
    * @param value the element to be inserted.
    * @returns true
*/
    public boolean insert(int value) {
        if (isMember(value)) {
            return false;
        }
        return list.add(new Integer(value));
    }
/**
    * Removes an element from the Set object.
    * @param the element to be removed
    * @returns success of operation
*/
    public boolean remove(int value) {
        return list.remove(new Integer(value));
    }
/**
    * Checks whether a given element is in the Set object
    * @param the element to be searched
    * @returns true if the element is in the Set; false otherwise.
*/
    public boolean isMember(int value) {
        return list.contains(new Integer(value));
    }
}
```