Embarrassing Ways to Lose Points on Programming Assignments

By Roger Whitney
San Diego State University

While grading I come across programs that a quick glance are clearly a mess. These programs stink in ways that are extremely obvious and trivial to avoid, yet the authors seem oblivious to the problems. I hope that by reading this students can avoid these problems.

Keep in mind that even large class assignments are tiny compared to code written in industry. As a result software development in industry is different than completing school assignments. One can survive school using practices that are unworkable in industry, so it is important to pay attention to how you develop code. One bid difference between school and industry is that in industry a program is read many more times than in school. In industry other programmers read your code to use it, to find and fix bugs, to modify it and to maintain the code. Often the only ones to read a student program is the instructor or grader. As a result students often don't realize what issues are important in making a program readable.

Indentation

Programmers use indentation to indicate the block structure of code. Done correctly indentation shows the reader the block structure of the program. If done poorly the program can be nearly unreadable. The IDEs and editors that I use (VisualWorks browser, Eclipse, BBEdit, Xcode) format code in a consistent way if asked. At times I see code like the following. How hard is it to see the block structure of the code. Imagine reading 10 pages of code like this and then having to assign a grade to the work.

```
  public void commandAction(Command c, Displayable d) {
    if (c == restartCmd) {
  theGame.restart();
    } else if (c == levelCmd) {
    Item[] levelItem = {
   new Gauge("level", true, 9 theGame.getLevel())};
  Form f = new Form("Change Level", levelItem);
      f.addCommand(OkCmd);
   f.addCommand(cancelCmd);
  f.setCommandListener(this);
    Display.getDisplay(this).setCurrent(f);
    } else if (c == exitCmd) {
 destroyApp(false);
 notifyDestroyed();
  }
}
```

The following is at least uniform, but still not very readable.

```
public void commandAction(Command c, Displayable d) {
if (c == restartCmd) {
theGame.restart();
} else if (c == levelCmd) {
Item[] levelItem = {
new Gauge(“level”, true, 9 theGame.getLevel())};
Form f = new Form(“Change Level”, levelItem);
f.addCommand(OkCmd);
f.addCommand(cancelCmd);
f.setCommandListener(this);
Display.getDisplay(this).setCurrent(f);
} else if (c == exitCmd) {
destroyApp(false);
notifyDestroyed();
}
}
```

Now notice how much easier it is to see the block structure in the following code. The point is not that you should adopt this particular indentation style, but that a reasonable and consistent indentation style greatly improves the readability of code.

```
public void commandAction(int c, Displayable d)
        {
        if (c == restartCmd)
            {
            theGame.restart();
            }
        else if (c == levelCmd)
            {
            Item[] levelItem = { new Gauge("level", true, 9 theGame.getLevel())};
            Form f = new Form("Change Level", levelItem);
            f.addCommand(OkCmd);
            f.addCommand(cancelCmd);
            f.setCommandListener(this);
            Display.getDisplay(this).setCurrent(f);
            }
        else if (c == exitCmd)
            {
            destroyApp(false);
            notifyDestroyed();
            }
        }
```

Tabs & Spaces

A common cause of inconsistent indentation is the mixing of tabs and spaces. Different programs treat tabs and spaces differently. As a result code that is perfectly lined up on the screen may be a mess when printed out like the code below.

```
{
  int a;
     int b;
  int c;
     int d;

  a = 12 * 3;
    c = a + 10;
```

Never mix tabs and spaces in aligning code. Always uses all spaces or all tabs in your code. I don't care which you use just never use them together in the same program. Smalltalkers should note that VisualWorks uses tabs to indent code, so you should always use tabs. The IDEs and editors that I use (VisualWorks browser, Eclipse, BBEdit, Xcode) handle this for me.

Line Wrap

I often get papers that look something like the following, which is hard to read.

```
public void setOfficeData(Vector officeData) {

   if (_facultyInformationView.getCancelButton().
isEnabled()) {
      if (officeData.size() == 3) {
      boolean clearTable = false;
      int rows = _facultyInformationView.
getOfficeDataTable().getRowCount();

      for (int i=0; i<rows; i++) {
        if ( ((String)_facultyInformationView.
getOfficeDataTable ().getValueAt( i, 0)).equals("")) {
          _facultyInformationView.getOfficeDataTable().
setValueAt( officeData.get(0), i, 0);
          _facultyInformationView.getOfficeDataTable().
setValueAt( officeData.get(1), i, 1);
          _facultyInformationView.getOfficeDataTable().
setValueAt( officeData.get(2), i, 2);
          break;
        }

        if (i == 9)
          clearTable = true;
      }
```

The cause of this problem is clear, an edit window can hold lines much longer that paper. If you know that you need to print your program out do the following. Using the same editor you use on your programs create a long ling of text and print it out the same way you would print your programs. Now count the number of characters you can get on a row before the line wraps to the next row. While you write your program don't create lines longer than that. I do this by adding a comment (shown below) that shows the maximum row width to each file. Some IDEs (VisualWorks browser and Eclipse) handle this problem for you.

```
//
3456789012345678901234567890123456789012345678901234567890123456789012345678
90123456789
```

Blank Lines

Blank lines are useful in improving the readability of code. I have seen programs that don't use blank lines even between methods. This makes it hard to find beginning and ending of methods. Some people do the opposite and place a blank line after every line of code, which is also hard to read. The code gets to spread out. Compare the code below with the more condensed version that appeared earlier.

```
public void commandAction(int c, Displayable d)
    {

    if (c == restartCmd)

        {

        theGame.restart();

        }

    else if (c == levelCmd)

        {

        Item[] levelItem = {

            new Gauge("level", true, 9 theGame.getLevel())};

        Form f = new Form("Change Level", levelItem);

        f.addCommand(OkCmd);

        f.addCommand(cancelCmd);

        f.setCommandListener(this);

        Display.getDisplay(this).setCurrent(f);

        }

    else if (c == exitCmd)

        {

        destroyApp(false);

        notifyDestroyed();
```

Name Structure

Programs are full of names of variables, methods, arguments and classes. Using a naming standard makes it easier to read code as:

a. The format of the name can convey information
b. Consistency always makes reading code easier

Languages like C#, Java and Smalltalk come with large class libraries, which use a naming convention. These means that if you are to be consistent and use the class libraries you must use the same naming convention used in the class library. It does not matter if you like the style used in that language or if you have a better naming convention. Mixing conventions in a program makes them harder to read. So use the convention of the language in all your programs in that language. The following table contains a brief description of the conventions of C#, Java and Smalltalk.

Type	C#	Java	Smalltalk
Class	PascalCase	PascalCase	PascalCase
Constant	PascalCase	ALL_CAPS	NA
Method	PascalCase	camelCase	camelCase
Namespace	PascalCase	all lower case	PascalCase
Property	PascalCase	NA	NA
Instance Field	camelCase	camelCase	camelCase
Parameter	camelCase	camelCase	camelCase
Local variable	camelCase	camelCase	camelCase

Where PascalCase indicates capitalizing the first character of each word of a name as in PascalCase and camelCase indicates Capitalizing the first character of each word except the first word as in camelCase. In both PascalCase and camelCase abbreviations are to be avoided.

Summary

I call the problems discussed here embarrassing because they are so easy to spot and so easy to correct. There are many other issues to address in producing code, these are just one that one can spot in a few seconds. Students in my classes are graduate students and seniors. I should never see such problems in the code produced by graduate students and seniors. At that stage of training students should be well beyond these issues. There are a lot more complex issues to resolve in producing quality code. Any student beyond the sophomore level that produces code with the problems discussed above should worry. Whatever they are doing to prepare for their profession is not working well for them. I strongly urge such students to read the book Code Complete 2nd edition by Steve McConnell. This book covers may such issues in depth. Reading this book can make any student a better programmer.