Computer Science at U of P – Coding Standard

Any code you write for CS courses must follow the standards defined in this document.

Your Assignment Must Compile and Run via a Shiley Lab Computer
While you may use your personal computer to complete homework assignments, you take a risk when you do so. First the instructor will not assist you with the installation or configuration issues that arise from differences between how your computer is configured and what software is installed. Second, you are responsible for guaranteeing that any project you create will work on a computer in a Shiley Hall computer lab (room 206, 208 or 249). The instructor/grader is not responsible for figuring out why your project doesn't work and a zero score is a likely result.

General Code Requirements

- **Line width**: Limit line width to 80 characters. Exceptions can be made in unusual circumstances where it improves code clarity.
- **Meaningful and Consistent Names**: Choose meaningful and consistent names for variables, methods, etc. Abbreviations are fine but avoid over-abbreviating variables to the point of incomprehensibility. Views in your layout (.xml) files should have appropriate ids. Exception: You may use single letter variables as for-loop iterators without explanation.
- **Constants**: Avoid using literals when it's not clear what the literal means. Use a constant variable instead.
- **Modular Design:** Use helper methods to break your program's logic into digestible chunks. As a rule of thumb, a method is too long if can't all be on the screen at once.
- **Object Oriented:** Your code makes appropriate use of classes, objects and arrays.
- **Clean:** Keep your code spaced and indented properly. Do not leave stray chunks of commented-out code and similar detritus in your files.
- **Avoid Complexity:** Avoid programming constructs that make your code difficult to read. Just because you can cram everything into only a few lines of code doesn't mean you should. Avoid unnecessary "clever" tricks that just make things harder on a human reader.
- **Grammatically Correct:** Messages that will be seen by the user and comments in code exhibit correct spelling and grammar.
- **"The One True Brace Style"**: Much energy has been wasted on debate among programmers about whether curly braces should appear at the end of the line or by themselves at the beginning of the following line. Please pick a style and stick with it. It's okay to leave starter code in its original format even if it's different.
- **Encapsulation:** Where relevant, your code should make appropriate use of the public, private and protected reserved words with regard to classes, methods and instance variables.
- **Bad Programming Habits**: If your code should not have unnecessary copied/pasted blocks, misuses of global variables (e.g., the static reserved word in Java), do-nothing code, or similar bad programming.

Using External Sources

Coding is a highly collaborative activity. As you work on your project you will certainly get assistance from others.

Possible external sources include: the instructor, an upperclassman, a textbook, the Android SDK and, of course, the Internet. You are welcome to use external sources in CS301 as long as you cite and explain them in your code.

Each time you use an external source, add a comment to your code in this format:

```
/**
External Citation
  Date:     14 September 2015
  Problem:  Could not get the background color of my button
            to change
  Resource:
     http://stackoverflow.com/questions/7957494/change-
     background-color-of-a-button-in-an-android-application
  Solution: I used the example code from this post.
*/
```

All five components (External Citation, Date, Problem, Resource and Solution) are required.

The limits of this collaboration will vary from course to course. Please consult your syllabus. A citation does not excuse a violation of course policy. Copying code without a citation is plagiarism and will be dealt with seriously.

Comment-Related Requirements

Comments in your source code should always be helpful. Helpful comments explain what is going on at a level that will be meaningful to the reader. Helpful comments often explain *why* the code is doing something. Helpful comments occur where needed but not so frequently that they clutter the code. Here is an example of code with bad comments:

```
//If unblock me is not null
if (unblockMe != null)
{
    //set sp equal to the unblockMe device's stack pointer
    int sp = unblockMe.getRegisterValue(CPU.SP);
    //increment sp
    sp++;
    //write the "data" to the sp address
    m_MMU.write(sp, data);
    //increment sp
    sp++;
    // write SC_RET_SUCCESS to the sp address
    m_MMU.write(sp, SC_RET_SUCCESS);
    //set the SP in unblockMe to be equal to sp
    unblockMe.setRegisterValue(CPU.SP, sp);

    //Call the unblock method on unblockMe
    unblockMe.unblock();
}
```

For comparison, here are good comments for the same code:

```
//If there is a process whose I/O is completed, push the
//requested data onto its stack
if (unblockMe != null)
{
    //Push the read data and a success code onto
    //the process' stack
    int sp = unblockMe.getRegisterValue(CPU.SP);
    sp++;
    m_MMU.write(sp, data);
    sp++;
    m_MMU.write(sp, SC_RET_SUCCESS);
    unblockMe.setRegisterValue(CPU.SP, sp);

    //The process can begin running again so unblock it
    unblockMe.unblock();
}
```

Class and Method Headers

Any Java source file you create or modify must have a header comment that describes what the class does. This header should include an @author tag with your name.

Any method that contains more than two lines of code needs a method header. The method header should be in JavaDoc/DOxygen format as follows:

```
/**
 * attempts to take over the world by means of a given plan.
 * The caller must supply a sidekick.
 *
 * CAVEAT:  This method has yet to complete successfully.
 *
 * @param worldID a unique identifier for the world to be taken over.
 *                If this value is set to 0, planet Earth is assumed.
 * @param plan    the current plan for taking over the world.
 * @param pinky   a sidekick is required for success
 *
 * @return true if the plan is successful and false otherwise
 */
public boolean takeOverTheWorld(int worldID, Plan plan, Mouse pinky)
{
    etc...
```

The method header comment *must* contain a description of the method, any caveats, a description of each parameter and the return value. Additional information is welcome where appropriate. Also note the use of the extra asterisk in the opening sentinel ("`/**`" instead of just "`/*`").

Humor

For an example of what not to do, check out How to Write Unmaintainable Code at:
http://mindprod.com/jgloss/unmain.html
and here:
https://github.com/Droogans/unmaintainable-code#how-to-write-unmaintainable-code