The 10 coding commandments for concurrent programming

For other classes, you are expected to turn in a r*eport*. In this class, ***your code is the report****.* As such, your "report" must meet **strict** criteria, listed below.

1. A comment will indicate the purpose of the class. This comment precedes the class declaration.

2. An identifier's name indicates as closely as possible its purpose, with the usual exception of the standard "loop" or iterative variables: `i`, `j`, and `k`. Identifier names only use the letters: `a-z`, `A-Z`, `0-9`, and underscore (_).

3. Comments: each variable or constant shall have a comment (indicating why it is declared, for what purpose, how it will be used) on the same line as the variable declaration, as in the following examples:

   ```
   int curpos = 0;            // current position in the array that is used
                              // to contain all producer messages
   Random mix = new Random(); // used to mix up thread execution
   ```

   When possible, a variable is initialized upon declaration as in the above example.

   Each process, method, or operation shall contain a comment indicating its purpose, immediately preceding its declaration, as in the following example:

   ```
   // Expects up to 4 arguments: #processes, #iterations, bridge_len, max_size
   public static void main(String args[]) { /* … */ }
   ```

   A process, method or operation should not be longer than **50** lines.

4. Code shall be **consistently** indented, **no less than 3 spaces**, but **no more than 8 spaces**. The "tab" character **should** be avoided to preserve indentation across editors, machines, etc.

5. **"Magic"** numbers shall not appear in the code, apart from obvious uses of `0` (zero) and `1` (one). An exception is granted when collecting program arguments.

6. **Constants** shall be declared in **UPPERCASE** letters, to be easily distinguishable from other identifiers.

7. No line shall be longer than **120** characters. Watch out for trailing whitespace.

8. Each source file contains the CVS keywords **`$Header$`** and **`$Log$`**, so that source files contain the complete revision history. These lines are included in your file in the following manner (careful, they are **case sensitive**):

   The **first** line in your source file is

   ```
   // $Header$
   ```

   The **last** lines in your file are

   ```
   /*
   ** $Log$
   */
   ```

   CVS comments indicate what has changed, and why. You include a description of bugs (symptoms and fixes) and enhancements. The log is the bulk of your "report". Each file should be checked in separately when appropriate. The log contains information concerning how you tested your program – **what you specifically tested**, which program parameters you used, **what bugs were revealed** with your tests.

9. Source files assume that all the codes, necessary for compilation, are in the local directory; there are no `import` statements of your own code. **Each lab is a separate project, located in its own directory**. Within a directory, all source is written in the same programming language.

10. Exceptions are **not** ignored. For each `catch` statement, there is a message and stack trace printed (or your comment on why a particular exception can be ignored):

```
try {
      /* Something… */
}
catch (Exception e) {
      System.err.println("Relevant information goes here");
      e.printStackTrace();
}
```

PENALTIES

- Failure to follow any of the above rules results in a penalty of **1/2** point of your lab, regardless of the working state of your program, for each rule violation. In case of gross negligence, this penalty may be increased.
- A turned-in program that does not compile **for whatever reason** is penalised 3 points.

Questions regarding any rule are addressed **before** you turn in your program.