Testing Reading

Prior to, during, and after writing code, testing should be considered. This reading will give you an overview of things to consider, when designing a testing plan for specific software, as well as when considering the design of complex programs.

Every program should:

- Meet the requirements outlined when it was assigned.
- Be designed based on good programming practices.
- Be clear to the user.
- Behave as expected based on given examples.
- Behave as expected based on unacceptable input (both in range and data type) and sentinel values (values that cause all or part of the program to terminate).
- Be versatile enough to work with a range of accepted values, not just the given examples.

Furthermore, programs can be designed to:

- Meet user expectations (or at least expectations of a target audience).
- Be intuitive.
- Be appealing, feel useful, and subsequently be interpreted as valuable to the user.

It is important to realize that all software has room for improvement. Not every user will understand it, and not every error may be handled. However, a programmer makes decisions during the design of the software, to try to restrict issues, and should document faults or limitations with the code being written, as issues are realized. Ignored issues are more problematic than documented ones, as a programmer is judged by the code (s)he writes, and that includes documentation and pseudocode.

A program testing plan for user input tests for:

- Compatibility: Are all data types able to be used?
  In cases where Scanner is used, an error may automatically be thrown if an incorrect data type is used. For instance, if *nextInt()* is used, inputting a decimal value may cause the program to terminate and throw an InputMismatchException. In this case, the Scanner already responds to the unacceptable input, so the programmer need not add a structure to address this. However, the prompt for input should be overtly in favor of integers only, so as to avoid this issue.

  Similarly, *nextInt()* doesn't read integers exceeding a specific size. If it is reasonable to expect a user to try to enter that number, *nextLong()* would have been a better choice during the design stage.

- Reliability: Are the same appropriate answers provided each time in the same appropriate way?
  Just because a program works once doesn't mean it will work a second time, particularly for programs that are iterative (repeat several times before finishing). Be sure to test results after unacceptable input, as well as acceptable input.

- Usability:
  Each user-interface program needs to be clear to the user. This starts with a clear prompt clarifying what the user should do (and perhaps should not do), in a concise, polite way. Language used should be at the

appropriate level for the intended audience, and spacing should be added to make it easy for the user to follow the request and see their subsequent input, particularly if the input is unacceptable, and they mean to try again.

For complex programs, during the design stage of programming or early in programming, it is in the programmer's best interest to approach an intended user and watch how they respond to a prompt (or series of prompts, with layout, for complex programs). During this user test, it is important for the programmer to simply watch – not offering any guidance whatsoever. What the user does or does not do when prompted provide ample evidence to fuel further design, if needed.

- Performance: Every program should be designed to be quick to run. Significant delays will cause the user to question the reliability, usability, and potentially the scalability of the program.

- Scalability: For advanced programmers, considering the ability to reuse the software being tested in different contexts, or add additional features to improve the software, can be very important.

In our class, we will put comments about testing as a last part in a program header's pseudocode. Include in your comments tested data types and values, as well as any issues/unaccepted values that your software does not yet address, that cannot be handled yet. You can also comment on unaccepted values that are handled by other software that you are using, such as the Scanner class.

<u>Reading Questions:</u>

1. What are three good programming practices?
   a)

   b)

   c)

2. What is a sentinel value? _____

3. What is the difference between compatibility and reliability? Define each term, then contrast the two.

4. Why is it important not to offer any guidance whatsoever, when testing usability?

5. Where should a testing plan go, in a program? _____

6. *Hello.java* accepts input of a name, then prints "Hello" and the name. For instance:
   What's your name?
   Sally
   Hello Sally.

Write a testing plan: a list of inputted data type and values, and any comments about values that won't be handled (you can assume *Hello.java* has no control structures to handle unacceptable input).