Programmer Thinking = Creating a Custom Project

Dr. William L Honig, Loyola University Chicago

Let's say you have a good custom project idea, approved, and are ready to start work. What do I do first?

1. **Don't Code**. NEVER, NEVER, start a big programming project by writing "just a bit" of code.

2. Two things to get in hand at the very beginning:

   a. **Understand the user interface**, what it looks like, how it is used. The fancy computer science talk for this is "**Use Case**". One use case covers one complete interaction with a system. For example, if it were a To Do List, likely there would be use cases for creating/modifying the list, and marking something complete.

   b. DESIGN, DESIGN. What are the pieces of your program going to be. You know I like UML for this step**. Find the classes you will need. Set up key data (attributes) and methods (behaviors) for each one.**

3. Now you are ready to begin! But DON'T CODE YET.

   a. Talk through, with a partner, a friend, or yourself, how something will work in the code. See if, using the UML, you can "**See the Steps the code will follow**". To add a new item to the existing to do list, I will use this constructor of the Item class, and then the addItem method of the ToDoList class. Understand where the information will come from, what will be in parameters, what will be in return values.

   b. Make up one or two (or three) **important test cases for your code**. Create a new ToDO list with three entries called "Buy eggs, 1 doz", "Cash pay check before run out of money", "Register for next semester". Mark the middle one done. Delete the first one even if not marked done… Fancy computer science talk would be "test driven" development. You plan Now how you will test it. This helps you understand what the code will need to do. It also ensures that your tests will be better when it is time to really do them.

4. Ready to code, almost.

   a. Pick one key class, create at least some of the attributes, one or more constructors. Write one of the harder methods. Of course **start with pseudo code**. You might pick the ToDoList addItem method.

b. Switch between classes as your understand how to get the most complex methods written. Add get and set methods as you discover needs for them.

c. Something hard to do? If it is unique to your program, "**Make the problem go away**" – make up a function to do it for you and come back to it later. Be sure you understand what this function needs and what it returns. The computer science term for this is "**abstraction**" or "**step wise refinement**".

d. Something hard to do that seems general purpose? See if there is any java built in classes that will help! Take time to explore the methods and constructors provided. This can save you lots of time and give you a guaranteed correct solution!

5. Wrap Up the Coding

    a. Finish all the parts, **compiling as often as possible**. Don't wait until the end to try to compile. Even when you cannot run your code it should compile, or have error messages that you understand.

    b. Test, Test, Test.

    c. Touch up your code, remove any TODO lines, add comments if helpful.

    d. **Test again and again** to make sure it still works

    e. Have someone else test it.

    f. Test it some more (about this time you will remember what Dijkstra and Kernighan said about debugging)….