Three examples of different kinds of abstraction for the 8-puzzle:

1. **Example A**: puzzle state is passed into/out of methods (no objects!)

   ```java
   PuzzleA {
       int Scramble();
       int Tile(int state, int r, int c);
       int Move(int state, char d);
   }
   ```

   • Data abstraction?
   • Functional abstraction?
   • Creation on demand?

2. **Example B**: static variable stores puzzle state inside class (no objects!)

   ```java
   PuzzleB {
       private static int state; // static means one per class!
       void Scramble();
       int Tile(int r, int c);
       boolean Move(char d);
   }
   ```

   • Data abstraction?
   • Functional abstraction?
   • Creation on demand?

3. **Example C**: instance variable stores puzzle state (objects!)

   ```java
   PuzzleC {
       private int state; // one per object!
       void Scramble();
       int Tile(int r, int c);
       boolean Move(char d);
   }
   ```

   • Data abstraction?
   • Functional abstraction?
   • Creation on demand?
How to write programs

The following is a good process to follow whenever you’re writing a program. This is good software design and engineering practice! If you follow it, you’re likely to a) spend less time wrestling with the compiler and b) reduce the number of mistakes in your code.

1. Read the assignment.

2. Think about the assignment (don’t skip this step!).

3. Come up with example inputs and decide what the output should look like. If you understand how to process the input, you have a better chance of writing a program that can also process the input. Identify any “tricky” examples and figure out how you would deal with them.

4. Write a general algorithm to solve the problem. Write it out in steps, like a recipe. (For example, this list of steps is an algorithm!)

5. Decide what classes you’ll need (pictures are helpful here).

6. Decide what methods you’ll need.

7. “Stub” your methods. This means that you write out the method header (name, return type, input arguments) and have it just return a simple value. For example:

   ```java
   int tile(int r, int c) {
     return 0;
   }
   
   boolean move(char d) {
     return true;
   }
   ```

   This allows you to put together a “skeleton” program that has all of the pieces it will eventually have. Of course, it won’t run properly, but you should be able to get it to compile. Later you can go back and fill in the details for each method.

8. Write a test program (or test input files), using what you came up with in step 3.

9. Compile/run your (stubbed) program with the tests. The answers won’t be correct (because the methods are only stubs), but you’ll get to make sure all the pieces are in place.

10. Implement one method: first, decide which parts of the algorithm it involves. Then copy your algorithm into your program in the form of a set of comments. Then implement each comment in Java code. Test this method thoroughly.

11. Proceed to the next method. Test it thoroughly. And so on.

12. Once you’re satisfied, remove all debugging output and submit.