Managing Complexity

Programming is more than just syntax
Learning things that matter

- Things change rapidly in computer science
  - Languages go in and out of popularity, operating systems change, even programming styles change
  - Half of what you learn in my class will be outdated in five years
- CIT591 is primarily an introductory programming course
  - It uses this year’s popular programming language
  - Unfortunately, that language is so complex that learning the syntax takes time away from learning to program
- In this slide set I’m talking about things that will not be outdated in five years
  - But my examples will be from Java
Programs should work correctly

- Many programs must be *highly* reliable
  - Medical programs, space vehicle control programs, sales programs, income tax programs
  - Household robots, self-driving automobiles
- The need for correctness isn’t going to change any time soon

Programs should continue to work correctly after they are modified, or updated, or had new features added

- Thus, it is important to be able to modify programs *safely*

This means:

- Clear, concise, readable programs
- Good tests, especially regression tests
Can we *read* a program, or do we have to *decipher* it?

Here’s a method I would consider readable:

```java
public boolean isLeapYear(int year) {
    if (year % 400 == 0) return true;
    if (year % 100 == 0) return false;
    return year % 4 == 0;
}
```

At this point, you may feel that *all* programs have to be deciphered

I feel the same way when I try to read German

With practice, deciphering changes to reading—mostly 😊
void playGame() {
    boolean playAgain = true;

    while (playAgain) {
        int computersScore = 0;
        int usersScore = 0;
        boolean nobodyHasWonYet = true;
        while (nobodyHasWonYet) {
            computersScore = computersScore + resultOfComputersTurn();
            usersScore = usersScore + resultOfUsersTurn();
            printCurrentScores(computersScore, usersScore);
            nobodyHasWonYet = computersScore < WINNING_SCORE &&
                             usersScore < WINNING_SCORE;
        }
        printFinalScores(computersScore, usersScore);
        playAgain = askUser("Do you want to play again?");
    }
}
A less readable method

- private static int giveRandomNumber(int minValue, int maxValue) {
  if (minValue > maxValue) {
    int temp = maxValue;
    maxValue = minValue;
    minValue = temp;
  }
  Random random = new Random();
  int temp;
  if (maxValue < 0 && minValue < 0) {
    temp = 0 - random.nextInt(minValue) - 1;
    while (maxValue < temp) {
      temp = 0 - random.nextInt(minValue) - 1;
    }
  } else {
    temp = random.nextInt(maxValue + 1);
    while (minValue > temp) {
      temp = random.nextInt(maxValue + 1);
    }
  }
  return temp;
}
What makes a method “readable”?

- Short enough to see the entire method at once, without scrolling
- Does a single thing
- Has a meaningful, descriptive name
- Is properly formatted, and follows established conventions
- Has comments that further clarify what the method does
- Calls methods with meaningful, descriptive names
- Uses established idioms
  - Very idiomatic: `for (int i = 0; i < array.length; i++)`
  - Less idiomatic: `for (row = 0; row <= array.length - 1; ++row)`
- Has a short, memorable parameter list, with well-chosen parameter names
- Doesn’t do “weird” things
  - Doesn’t change parameter objects unnecessarily
  - If available outside the class, works for any valid object
    - That is, it doesn’t depend on some other method being called first
What makes a program “modifiable”?

- Good tests are essential
  - More bugs are introduced when “correcting” a program than at any other time
    - If you have a complete set of tests, you can do this much more safely
  - Frequently, in order to introduce new features, you have to refactor (reorganize) a program
    - If you have a complete set of tests, you can do this much more safely

- You can add features, but you cannot change features that other people (that is, other parts of the project) depend upon
  - At least, not without an extremely convincing reason
  - You can’t change what methods do, but you can change how they do it
  - You can only change how methods work if nothing else depends on it
    - This is why you must hide as much as possible of your implementation
Example

- A modifiable program:
  ```java
  public class Lexicon {
      private String[] words;
      private int[] counts;
      private int numberOfWords = 0;
      // etc.
  }
  ```

- An *un*modifiable program:
  ```java
  public class Lexicon {
      String[] words;
      int[] counts;
      int numberOfWords = 0;
      // etc.
  }
  ```
Information hiding

- When you provide a class to a project,
  - You should provide everything that is needed by the project
  - You should **not** provide anything that isn’t needed
    - If you do, someone, somewhere, will take advantage of it
    - If you then change it, **you** will get the blame

- There is a lot more to be said on the topic of information hiding, but I don’t have the time right now to say it all

- I will add this much:
  - Information hiding *also* applies to your JUnit tests
  - If you don’t want your tests to break when you make *correct* changes to your program, don’t depend on features that should be hidden
The End