# Testing and Debugging

## Current testing procedure (maybe)

- I just wrote a `squareIt` method let me test it…..

```
public static void main(String [] args)
{
       System.out.println(squareIt(2));
}


astudent: java SquareIt
          4
```

- Well,  it didn’t give me an error and the answer is correct
  It must work correctly

## The problem…

```
public static int squareIt(int n)
{
     return n * n;
}
public static int squareIt(int n)
{
     return n + n;
}
public static int squareIt(int n)
{
     return 2 * n;
}
```

## Testing

- Multiple types of testing
  - Unit testing
  - Integration testing
  - System testing
  - Acceptance testing
- Focus right now will be on unit testing
  - Verify one section of code, usually a function/method or a class
  - Can not verify the entire piece of software will work, just that the pieces work correctly independent of each other

## Unit testing

- Want to test:
  - Several ‘normal’ cases
  - Corner or boundary conditions – inputs that often require special handling in the code
- But you don’t need to go overboard
- Helpful hint:
  - Create a test class that extends class you want to test

## Unit testing case study: Fly World

- Some of the key pieces
  - FlyWorld constructor:  use the eye-ball test
  - FlyWorld isValidLoc:
    - Row = -1, column = 0 → Out of bounds should return false
    - Row = 0, column = -1 → Out of bounds should return false
    - Row = numRows, column = 0 → Out of bounds should return false
    - Row =0, column = numCols → Out of bounds should return false
    - Row/Column are valid → In bounds should return true

## isValidLoc

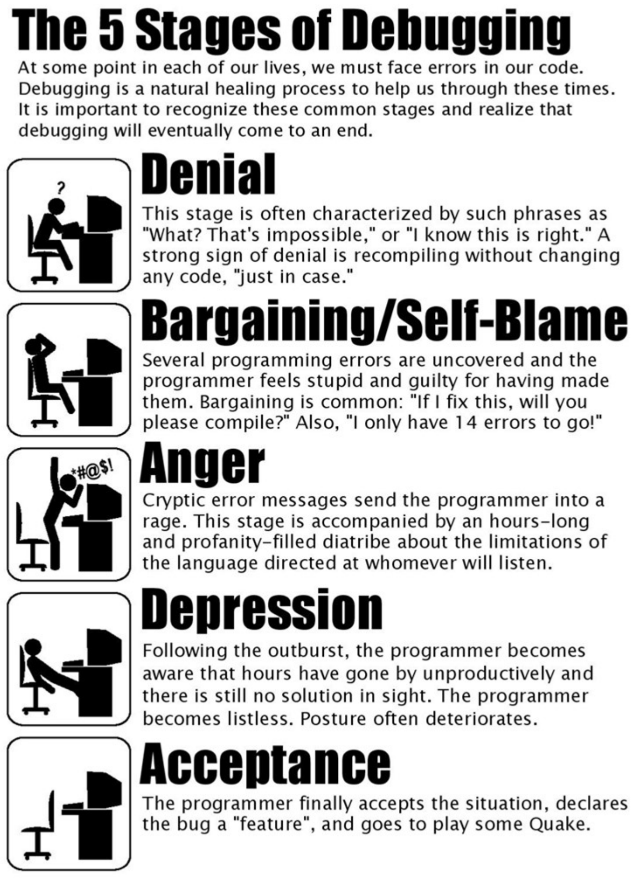
```
protected void testValidLoc()
{
    System.out.println("\n\tTesting isValidLoc...");
    int r = -1;
    int c = 0;
    if (isValidLoc(r, c))
    {
        String l = "Location [" + r + "," + c + "]";
        System.out.println(l + " is not valid, but student isValidLoc returned
valid/true");
        System.out.println("Row index checking is probably off");
        System.out.println("Test failed");
        System.exit(0);
    }

    r = numRows;
    if (isValidLoc(r, c))
    . . . . .
```

## Unit testing case study: Fly World

- Key pieces
  - Frog
    - generateLegalMoves
    - update
    - eatsFly
  - FlyWorld
    - moveFly
    - movePredators

??? Questions ???

## Current debugging procedure (probably)



**The 5 Stages of Debugging**

At some point in each of our lives, we must face errors in our code. Debugging is a natural healing process to help us through these times. It is important to recognize these common stages and realize that debugging will eventually come to an end.

**Denial**

This stage is often characterized by such phrases as "What? That's impossible," or "I know this is right." A strong sign of denial is recompiling without changing any code, "just in case."

**Bargaining/Self-Blame**

Several programming errors are uncovered and the programmer feels stupid and guilty for having made them. Bargaining is common: "If I fix this, will you please compile?" Also, "I only have 14 errors to go!"

**Anger**

Cryptic error messages send the programmer into a rage. This stage is accompanied by an hours-long and profanity-filled diatribe about the limitations of the language directed at whomever will listen.

**Depression**

Following the outburst, the programmer becomes aware that hours have gone by unproductively and there is still no solution in sight. The programmer becomes listless. Posture often deteriorates.

**Acceptance**

The programmer finally accepts the situation, declares the bug a "feature", and goes to play some Quake.

## Debugging… a better way

- Preparation before coding
  - Have test cases worked out (unit testing helps here)
  - Work through algorithms by hand
- Code -> test -> code
- When problems do arise
  - Initial focus should be on what you just changed or added --or--
  - If there is an error (compile or run-time) start there

## Debugging… a better way

- Finding the problem
  - Start with a few strategically placed and **<u>INFORMATIVE</u>** print statements
    - Bad:

```
for (int i = 0; i < 23; i++)
{
    System.out.println(i);
    int sum = i + 1;
    System.out.println(sum);
}
```

Good:

```
for(int i = 0; i < 23; i++)
{
    int sum = i+1;
    System.out.println("i: " + i + " sum: " + sum);
}
```

## Debugging…a better way

- Some "try this first" cases
  - Index out of bounds errors → Figure out the index value causing problem
    - Is the index invalid, problem with logic of the code/loop running too long/index calculation incorrect?
    - Is the array/ArrayList/etc. to small?
  - Null pointer errors
    - Did I initialize the variable (i.e., myVariable **=** <SOMETHING>;)?
    - Am I actually storing things in my variable (applies especially to ArrayList, HashMap, etc)?
    - If the variable is an instance variable – Did I accidentally re-declare the variable, usually in the constructor (e.g., **int numRows** = <something> vs. **numRows** = <something>)

## Debugging…a better way

- Use a debugger (jdb)
  - Unlike adding print statements, don't have to recompile every time
  - Can examine variables and much more
  - Can control execution of program
  - Can change values of variables
- Before using compile code with special flag
  - javac **–g** *.java
- Instead of invoking: java <SOMETHING>
  - You use: jdb <SOMETHING>

## Using jdb

- Controlling program execution
  - Set a **breakpoint**
    - stop at <class>:<line number>
    - stop in <class>.<method>
    - stop in <class>.**<init>** : sets breakpoint for constructor
  - Moving through the code
    - run : start executing code as normal
    - step : execute current line. If it involves a method call goes into the method
    - next : execute current line. If it involves a method, does not go into method
    - cont : continues execution until it hits a breakpoint, error, program ends normally

## Using jdb

- Examining the code
  - list : shows line of code about to be executed as well as code around it
  - where : shows program stack
    - up : moves to a previous stack frame
    - down : moves to next stack frame
  - print <expr> : print out value of an expression or variable
  - dump <obj> : prints out information about an object such as its contents (e.g. for array)
  - methods <class> : prints out all methods a class has
  - fields <class> : prints out all instance variables a class has
  - watch <class>.<field name> : keeps track every time field is modified
  - set <var> = <new value> : assigns new value to some variable
  - locals : prints value of all local variables includes method parameters
- help

## Demo: Fibo.java