

Testing and Debugging

CS 1027

1

Program Errors

- Compiler errors (Syntax errors)
- Runtime errors
- Logic errors

2

Compiler Errors

- **Syntax error**
 - Error in usage of Java
 - Caught by compiler
 - A program with compilation errors cannot be run
- **Syntax warning**
 - Warning message generated by compiler
 - Program can be run

3

Compiler Errors_

- Very common (but sometimes hard to find) syntax errors are:
 - Forgetting a semicolon
 - Leaving out a closing bracket }
 - Redeclaring a variable
 - Others?

4

Compiler Errors_

- Hints to help find/fix compiler errors:
 - Make sure your program has **line numbers**
 - Compiler errors are cumulative: when you fix one, others may go away
 - Realize that the error messages from the compiler are often not very helpful
 - The compiler does not know what you intended to do, it is merely scanning the Java code

5

Runtime Errors

- **Runtime error**: program runs but gets an **exception** error message
 - Program may be terminated
- Runtime errors can be caused by
 - **Program bugs**
 - Bad input
 - Hardware or software problems in the computer system

6

Runtime Errors

- Very common runtime errors are:
 - **null reference** (`NullPointerException`)
 - no object is referenced by the reference variable, i.e. it has the value `null`
 - **array index out of bounds** (`ArrayIndexOutOfBoundsException`)
 - Running out of memory
 - e.g. from creating a new object every time through a loop

7

Runtime Errors

- Hints to help find/fix runtime errors:
 - Check the exception message for the **method** and **line number** from which it came
 - Note that the line in the code that caused the exception may **not** be the line with the error
 - Example: consider the code segment

```
int [] nums = new int[10];
for (int j=0; j<=10; j++)
    nums[j] = j;
```
 - The exception will be at the line

```
    nums[j] = j;
```

but the error is in the *previous* line

8

Logic Errors

- **Logic error**: program runs but results are not correct
- Logic errors can be caused by:
 - incorrect algorithm
 - incorrect usage of operators
 - etc.

9

Logic Errors

- Very common logic errors are:
 - using `==` instead of the *equals* method
 - using `=` instead of `==`
 - infinite loops
 - misunderstanding of operator precedence
 - dangling *else*
 - off-by-one error (e.g. starting or ending at the wrong index of an array)
 - If index is invalid, you would get an exception
 - misplaced parentheses (so code is either inside a block when it shouldn't be, or vice versa)

10

Logic Errors

- Be careful of where you declare variables!
- Keep in mind the scope of variables
 - Instance variables?
 - Formal parameters?
 - Local variables?
- Example:

```
private int numStudents; // an attribute, to be
                        // initialized in some method

...
public void someMethod(){
    int numStudents = ...; // not the attribute!
    ...
}
```

11

Testing vs Debugging

- **Testing**: to identify any problems before software is put to use
 - *“Testing can show the presence of bugs but can never show their absence”.*
- **Debugging**: locating bugs (found either in testing or by user of software) and fixing them

12

Hints for Success

- When writing code:
 - **Understand the algorithm before you start coding!**
- Start small!
 - Write and test the simpler methods (e.g. getters, setters, toString)
 - Then write and test each of the more complex methods individually
- Check your code first by a preliminary hand trace
- *Then* try running it

13

Debugging Strategies

- **Trace** your code by hand
- **Add main method** to the class
- **Add print statements** to your code

14

Tracing by Hand

- **Tracing by hand**
 - Good starting point with small programs or simple methods
 - Problem: sometimes you do what you think the computer will do, but that is not what it actually does
 - Example: you may write that $9/5$ is 1.8, but it is really 1
- **Hint: draw diagrams of reference variables and what object(s) they are pointing to!**

15

Adding a main Method

- **Adding a main method** to the class
 - Conventionally placed at the end of the class code, after all the other methods
 - Then the class can be tested by simply running it.

16

Using Print Statements

- **Using print statements**
 - Insert `System.out.println()` statements at key locations
 - To show values of significant variables
 - To show how far your code got before there was a problem
 - In the print statement, it's a good idea to specify
 - The location of the trace (what method)
 - The variable name as well as its value

17

Debuggers

- All Integrated Development Environments have an **interactive debugger** feature
 - You can **single-step** step through your code (one statement at a time)
 - You can see what is stored in variables
 - You can set **breakpoints**
 - You can **"watch"** a variable or expression during execution

18

Defensive Programming

- Write *robust programs*
 - Include checking for exceptional conditions; try to think of situations that might reasonably happen, and check for them
 - Examples: files that don't exist, bad input data
- Generate appropriate error messages, and either allow the user to reenter the data or exit from the program
- Throw exceptions (see [Introduction to Exceptions](#) notes)
 - Can aid in finding errors or in avoiding errors
 - Example: invalid arguments (IllegalArgumentException)