Lecture Feb 7 - More Strings and Errors

Bentley James Oakes

February 8, 2018

This Lecture

Section 1

Iterating Through Strings

String Methods

Here is some examples of calling methods on `Strings`

```
String x = "this IS a STRING!";
String y = "THIS is a String!";

boolean eq = x.equals(y);
System.out.println("Equals: " + eq); //false

boolean eqIgnoreCase = x.equalsIgnoreCase(y);
System.out.println("Equals Ignoring Case: " + eqIgnoreCase); //true

System.out.println("X Length: " + x.length()); //17

System.out.println("X charAt(3): " + x.charAt(0)); //t
System.out.println("X charAt(7): " + x.charAt(3)); //s

System.out.println("X in lowercase: " + x.toLowerCase()); //this is a string!
System.out.println("X in uppercase: " + x.toUpperCase()); //THIS IS A STRING!
```

Double Letters

- Let's write a method to determine if a `String` contains the same two letters in a row
- For example, ``book'' and ``keeper'' have doubles of `o` and `e`
- `public static boolean doubleLetters(String s)`

Checking for Double Letters

```java
//checks to see if the String has two of the same character in a row
public static boolean doubleLetters(String s){

    //iterate through the String. Note the -1 in the condition
    for (int i=0; i < s.length() - 1; i++){

        //get each character and the next one as well
        char a = s.charAt(i);
        char b = s.charAt(i+1);

        System.out.println(a + " " + b);

        //if the characters are equal, return true
        if (a == b){
            return true;
        }
    }
    return false; //if we can't find double letters, return false
}
```

- Be careful with the condition in the *for loop*

Section 2

Testing Characters

ContainsA

```java
//returns true if the String contains
//upper or lowercase 'a'
public static boolean containsA(String s)
{
    //loop through the String
    for (int i=0; i < s.length(); i++){

        //put each character in c
        char c = s.charAt(i);

        //if this character is c
        //return true
        if (c == 'a' || c == 'A'){
            return true;
        }
    }

    //if we made it through the loop, return False
    return false;
}
```

Note

```
if ('a' <= c && c <= 'z'){
    numLetters ++;
}
```

- We can compare characters using $<$ and $>$
  - This test sees if c is a lower-case letter
- Note that chars literals have a single quotation mark ', not "

Binary Conversion

- With comparing characters, we can convert binary to decimal
  - Showed this last time
  - Try to write your own version
- Decimal to binary is very similar

Section 3

Errors

Four Types of Errors

1. Logic error
2. Style error
3. Compiler error
4. Run-time error

- Often comes up on tests...
- We'll show you code, and ask what kind of error (if any) will occur

1) Logic Error

- **Logic errors** occur when the program does something different than what the programmer expects
- Example:
  - Taking the average of two numbers
  - Should be `int average = (a + b) / 2;`
  - But accidentally wrote `int average = a + b / 2;`
- Division happens first
- Java has no idea there's an issue
  - Instructions are valid, but the result is 'wrong'

Debugging

- The way to find logic errors is to think about what the code is doing
- And to **debug your code**
- Two ways to debug your code

1. The print statement
   - Print variable values at specific points in the program
   - Are the values what they should be?
2. Use a debugger

Debugger

- The debugger in Dr. Java and Eclipse are very similar
- Check these links for info on Eclipse's debugger
  - `http://www.vogella.com/tutorials/EclipseDebugging/article.html`
  - `https://www.eclipse.org/community/eclipse_newsletter/2017/june/article1.php`
- The next slides show the debugger in Dr. Java

Dr. Java Debugger

1. Go into *Debug Mode*. This is the first option in the Debugger drop-down menu.
2. Once in this mode, toggle a **breakpoint**. Click on a line of code and select the *Toggle Breakpoint on Current Line* option
   - A breakpoint is where code execution will stop in debug mode
   - Need at least one breakpoint to debug or the execution won't stop
3. Run your code from inside *Debug Mode*
4. While you are debugging, you can *step over*, *step into*, *resume*

Debugger

- **Step over:** Clicking this button will execute the current line of code and step to the next line in the current method
- **Step into:** If there is a method call on the current line, the debugger will step into the first line of the method
- **Resume:** Continue execution until the next breakpoint, or until the end of the program

Watches

- **Watches** let you keep trace of the values of variables
- Add the names of variables to the *Watches* list
- Very helpful if you think there's an *infinite loop* in your code
  - Infinite loop: where the loop never stops

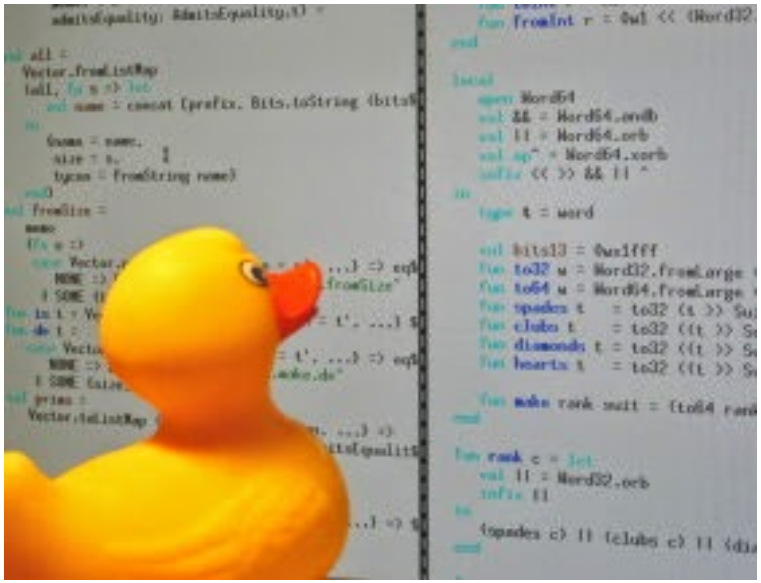
Watches

Name	Value	Type
s	This is a senten...	java.lang.String
i	1	int or Integer
c	h	char or Char...

Tips for Programming

- Always write code incrementally
  - Write some code, then test it
  - Keep adding small pieces and testing them
  - Try to write lots of methods
    - Every method should have one well-defined purpose

Rubber-duck Debugging



Explaining the problem to someone else is surprisingly useful.
https://en.wikipedia.org/wiki/Rubber_duck_debugging

2) Style Error

A **style error** is where the program is correct, but is hard to understand

Examples:

- Bad variable names
- Incorrect indentation
- Inconsistent braces
- Lack of comments
- Too many calculations on one line

- Java doesn't care, but it makes reading the program difficult
- Marks will be taken off for this on future assignments

3) Compiler error

- **Compiler errors** happen during compilation
- The compiler examines your code and raises errors
- Mostly related to variable types, syntax errors, or typos
  - eg. Sorting or returning the wrong variable type

```
5        int a = 3.5;
6
7    }
```

Interactions | Console | Compiler Output

```
1 error found:
File: /home/dcx/Dropbox/COMP 202/Lecture 9 - Errors/ErrorTest.java  [line: 5]
Error: incompatible types: possible lossy conversion from double to int
```

Dealing with Errors

Tips for helping with compiler errors:

1. Always fix the errors from the top to the bottom
   - If a line has a problem, then the compiler gets confused
   - And might report following lines as having problems
2. Use the Internet to search for the error message

4) Run-time Error

- **Run-time errors** occur during the running of the program
- Java complains that something unexpected happened
- Most run-time errors depend on values of variables
- The compiler doesn't check the values for you

Run-time Error Example

Dividing by zero is a run-time error

```
3     public static void main(String[] args)
4     {
5         divide(10, 0);
6     }
7
8     public static int divide(int a, int b)
9     {
10        return a/b;
11    }
```

Interactions | Console | Compiler Output

```
java.lang.ArithmeticException: / by zero
      at ErrorTest.divide(ErrorTest.java:10)
      at ErrorTest.main(ErrorTest.java:5)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
      at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
      at java.lang.reflect.Method.invoke(Method.java:498)
      at edu.rice.cs.drjava.model.compiler.JavacCompiler.runCommand(JavacCompiler.java:272)
```

Run-time Error Example

Accessing the wrong index in a `String` is also a run-time error

```
7         String s = "Hello";
8         System.out.println(s.charAt(100));
9
10
11      }
```

Interactions | Console | Compiler Output

```
> run ErrorTest
java.lang.StringIndexOutOfBoundsException: String index out of range: 100
      at java.lang.String.charAt(String.java:658)
      at ErrorTest.main(ErrorTest.java:8)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
      at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
      at java.lang.reflect.Method.invoke(Method.java:498)
      at edu.rice.cs.drjava.model.compiler.JavacCompiler.runCommand(JavacCompiler.java:272)
```

Stack Trace

```
java.lang.StringIndexOutOfBoundsException: String index out of range: 100
      at java.lang.String.charAt(String.java:658)
      at ErrorTest.main(ErrorTest.java:8)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
      at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
      at java.lang.reflect.Method.invoke(Method.java:498)
      at edu.rice.cs.drjava.model.compiler.JavacCompiler.runCommand(JavacCompiler.java:272)
```

- Luckily, Java will give you a **stack trace** where the run-time error occurred
- Every stack trace is different, but in this one:
- The top line is the precise error: `StringIndexOutOfBoundsException`
- The next line is the spot in Java code where the error occurred
- The next line is where in your code the error happened - `at ErrorTest.main(ErrorTest.java:8)`

Section 4

Catching Exceptions

Handling Bad Data

There are two ways we can handle bad values:

1. Check for a bad value before we do an operation
   - Example: Check for division by zero before doing it
2. *Try* the operation and *catch* the error

Try/Catch

Here we will have a **try/catch** block for division

```
public static int divide(int a, int b)
{
    try{
        //try to execute this code
        int c = a/b;
        return c;
    }catch(ArithmeticException e){
        //if this error occurred, then execute this code
        System.out.println("Error! You tried to divide by zero!");
        return 0;
    }
}
```

- If there's an `Exception`, Java will look to see if that `Exception` is caught.
- If so, then that block of code runs.
- We are providing instructions to run when an *ArithmeticException* occurs in the *try* block

Try/Catch

Here we are using `Integer.parseInt()` to parse input arguments

```
39      public static int getIntegerNumber(String arg)
40      {
41          try
42          {
43              return Integer.parseInt(arg);
44          }catch(NumberFormatException e)
45          {
46              System.out.println("ERROR: " + e.getMessage() +
47                                 " This argument must be an integer!");
48          }
49
50          //error, return 1
51          return 1;
52      }
```

Interactions | Console | Compiler Output

```
> run FinishedCalculator 12.3 12.3 5
Welcome to the Calculator program!
ERROR: For input string: "12.3" This argument must be an integer!
ERROR: For input string: "12.3" This argument must be an integer!
The first argument a is: 1
The second argument b is: 1
```

- This was done for you in the `Calculator` program in the first assignment

printStackTrace

We can also add the statement `e.printStackTrace()` to the *catch block*

```java
public static int divide(int a, int b)
{
    try{
        //try to execute this code
        int c = a/b;
        return c;
    }catch(ArithmeticException e){
        //if this error occurred, then execute this code
        System.out.println("Error! You tried to divide by zero!");
        e.printStackTrace();
        return 0;
    }
}
```

Interactions | Console | Compiler Output

```
Welcome to DrJava.  Working directory is /home/dcx/Dropbox/COMP 202/Lecture 9 - Errors
> run ErrorTest
Error! You tried to divide by zero!
java.lang.ArithmeticException: / by zero
      at ErrorTest.divide(ErrorTest.java:32)
      at ErrorTest.main(ErrorTest.java:9)
```

- Adds a stack trace for the user to understand the problem
- Note that because we caught the Exception, the program can continue

Catching all the Exception

- You can make your code less crashy with try/catch
- But you can't fix code with it

- For instance, you could always wrap your `String` iterations with a try/catch so it doesn't crash
- But you should instead make sure your code is right

Section 5

Throwing Exceptions

Throwing Exceptions

- Sometimes we want to intentionally crash the program
- For example, if the input data is not valid

No Error Checking

```
3       public static void main(String[] args)
4       {
5           System.out.println(getPercentage(100));
6           System.out.println(getPercentage(25));
7           System.out.println(getPercentage(-450));
8       }
9
10      public static String getPercentage(int grade)
11      {
12          return "Grade: " + grade + "%";
13      }
```

Interactions | Console | Compiler Output

```
Welcome to DrJava.  Working directory is /home/dcx/Dropbox/C
> run ErrorTest2
Grade: 100%
Grade: 25%
Grade: -450%
```

- Here we print a message even if the grade is not from 0 to 100
- Let's change this to crash the program if an invalid grade is entered

Error Checking

```java
        System.out.println(getPercentage(100));
        System.out.println(getPercentage(25));
        System.out.println(getPercentage(-450));
    }

    public static String getPercentage(int grade)
    {
        if (grade < 0 || grade > 100){
            String errorMessage = "This grade is invalid: " + grade;
            throw new IllegalArgumentException(errorMessage);
        }

        return "Grade: " + grade + "%";
    }
```

Interactions | Console | Compiler Output

```
Welcome to DrJava.  Working directory is /home/dcx/Dropbox/COMP 202/Lecture 9 -
> run ErrorTest2
Grade: 100%
Grade: 25%
java.lang.IllegalArgumentException: This grade is invalid: -450
      at ErrorTest2.getPercentage(ErrorTest2.java:14)
      at ErrorTest2.main(ErrorTest2.java:7)
```

- We specify an Exception to throw
- And providing a message (optional but recommended)
- `throw new IllegalArgumentException(message)`

Error Checking

- Here's a method that crashes the program if you enter the wrong username or password

```
public static void main(String[] args){
    //this login will be successful
    login("Bentley", "cats-4-ever");
    //this login will crash the program
    login("Giulia", "something-funny");
}
public static void login(String username, String password){
    System.out.println("Username: " + username);
    System.out.println("Password: " + password);

    boolean success = username.equals("Bentley") && password.equals("cats-4-ever");

    //if the login wasn't successful, throw an exception
    if (!success){
        throw new IllegalArgumentException("You entered the" +
                                            " wrong username/password: " +
                                        username + " " + password);
    }
}
```

```
> run ErrorTest
Username: Bentley
Password: cats-4-ever
Username: Giulia
Password: something-funny
java.lang.IllegalArgumentException: You entered the wrong username/password: Giulia something-funny
     at ErrorTest.login(ErrorTest.java:20)
     at ErrorTest.main(ErrorTest.java:8)
```

Exception Types

- There are lots of *Exceptions* that you could throw
- Try to be specific and give a good error message
- Here are some Exceptions you'll see in the course
  - `StringIndexOutOfBoundsException`
  - `NumberFormatException`
  - `ArithmeticException`
  - `IllegalArgumentException`
  - `ArrayIndexOutOfBoundsException`
  - `NullPointerException`

Crashing vs Not-Crashing

- Try not to throw an `Exception` if you can
- Instead fix the input
- For example, if the grade is outside the range 0 to 100, maybe print a message and fix the value

- It's much better to crash than return an invalid value
- For example, if you are writing a boolean-to-decimal converter, and the input is not a binary number, then you should crash
  - Don't want the program continuing with an invalid value
  - Force the user to fix their input