Review of Command Line Input

- Use BufferedReader and InputStreamReader classes to read input
- Use System.in to specify that input comes from standard input
- Syntax:
  ```java
  BufferedReader console =
  new BufferedReader(new InputStreamReader(System.in));
  ```
Review of Command Line Input (2)

- Use `readLine()` method to read a line from the terminal window
- `readLine()` method may cause exceptions so wrap in a try-catch block

```java
try {
    String input = console.readLine();
}
catch (IOException e) {
    System.out.println("Error: "+ e);
}
```

Reading More Than One Line

- You can use a `BufferedReader` to read a list of input by using a while loop
Reading More Than One Line (2)

• Ex:
  BufferedReader console = new BufferedReader(new InputStreamReader(System.in));
  boolean done = false;
  while (!done) {
    String line = console.readLine();
    if (line == null) {
      done = true;
    } else {
      System.out.println(line);
    }
  }

StringTokenizer

• Sometimes you may want to process the input of a string by looking at pieces of it
• The StringTokenizer class allows you to break apart a string on a delimiter
  – Default delimiter is a space
  – You may specify any number of delimiters that you want
• Import java.util.StringTokenizer to use
StringTokenizer Methods

- StringTokenizer(String line)
  - Constructor
- StringTokenizer(String line, String delimiters)
  - Tokenizes the String in line with the given delimiters
- StringTokenizer.hasMoreTokens()
  - Returns true if the StringTokenizer has more tokens
- StringTokenizer.countTokens()
  - Counts the remaining tokens
- StringTokenizer.nextToken()
  - Returns a String that contains the next token

StringTokenizer Example

```java
String tokenizer = new StringTokenizer(line);
int tokenCount = tokenizer.countTokens();
for (int i=0; i < tokenCount; i++) {
    String token = tokenizer.nextToken();
    System.out.println(token);
}
```
**StringTokenizer.countTokens()**

- Return the number of tokens that are left in the Tokenizer
- Whenever a token is retrieved from the list, it is deleted!
- Do not use this method in a for loop
  - Create a variable to store the total number of tokens before processing the tokens

---

**Formal and Actual Parameters**

- Formal Parameters – include *this* and all declared parameters (the data types and names in the parenthesis)
- Actual Parameters – values that you supply a method
- Actual parameters copy their values to the formal parameters memory locations
  - *this* gets the location in memory of the calling object
  - The parameter gets the value passed into it
Return Statement

- The return statement causes an immediate exit.
- If a return statement is reached, the code after it will not be executed!
- Methods that do not contain an explicit return statement return back to the calling line when all the code has been executed.

Execution Flow Example

```java
amt = yConvert.fromD(200); // public double fromDollar(double d) {
    double amt, fee;
    fee = er - fr;
    amt = d * fee;
    return amt;
}
```
Execution Flow Example

```java
public double fromDollar(double d) {
    double amt, fee;
    fee = er - fr;
    amt = d * fee;
    return amt;
}
```

Memory

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>amt</code></td>
<td></td>
</tr>
<tr>
<td><code>d</code></td>
<td>200.0</td>
</tr>
<tr>
<td><code>amt</code></td>
<td></td>
</tr>
<tr>
<td><code>fee</code></td>
<td></td>
</tr>
</tbody>
</table>

amt=yConvert.fromD(200);
Execution Flow Example

```java
amt=yConvert.fromD(200);

public double fromDollar(double d) {
    double amt, fee;
    fee = er - fr;
    amt = d * fee;
    return amt;
}
```

Memory

| amt | 24846.3 |

Execution Flow Example (2)

```java
int x = 10;
int y = 20;
tester.myMethod(x,y);

public void myMethod(int one, double two) {
    one = 25;
    two = 35.4;
}
```

Memory

<table>
<thead>
<tr>
<th>x</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>20</td>
</tr>
</tbody>
</table>
Execution Flow Example (2)

```java
int x = 10;
int y = 20;
tester.myMethod(x, y);

public void myMethod(int one, double two) {
    one = 25;
    two = 35.4;
}
```

Memory

<table>
<thead>
<tr>
<th>x</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>one</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>two</td>
<td>20</td>
</tr>
</tbody>
</table>

© 2005 Sarah E. Smith
Execution Flow Example (2)

```java
int x = 10;
int y = 20;
tester.myMethod(x, y);
```

```
public void myMethod
(int one, double two) {
    one = 25;
    two = 35.4;
}
```

<table>
<thead>
<tr>
<th>Memory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>20</td>
</tr>
</tbody>
</table>

References

- Jason Schwarz’s Lecture 14 and 15 slides: [http://courses.ncsu.edu/csc116/](http://courses.ncsu.edu/csc116/)
- Example of execution flow from Chapter 4.4 in Wu p 172 and 174