CIS 110: Introduction to Computer Programming

Lecture 6

Flexible Methods

(§ 3.1-3.2)

Announcements

- Homework 2 due at 23:59:59 tonight!
  - Watch Piazza submission page status.
  - Office hours from 110 staff throughout the day.
- Homework 3 + lab 3 will be out tonight.
- Exam #1 is on 10/5 (Wednesday after next)
  - Practice exam #1 is out.
  - Review session day before the exam.

Outline

- Parameter passing
- Return values

Method Mystery Live!

```
public static void doWork() {
  for (int i = 1; i <= 2; i++) {
    doSomething(i);
    System.out.println(”
      We did it!");
  }
}
```

```
public static String magic(
    int x, String msg) {
  msg += “: “ + x;
  x *= 2;
  msg += x;
  return msg;
}
```

```
public static void doSomething(int x) {
  x = x * 2;
  String s = magic(x, x + “ magic”);
  System.out.println(s);
}
```

What is the output when I call doWork()?

doWork

```
public static void doWork() {
  for (int i = 1; i <= 2; i++) {
    doSomething(i);
    System.out.println(”We did it!");
  }
}
```

doSomething

```
public static void doSomething(int x) {
   x = x * 2;
   String s = magic(x, x + " magic");
   System.out.println(s);
}
```

magic

```
public static String magic(int x, String msg) {
  msg += ": " + x;
  x *= 2;
  msg += x;
  return msg;
}
```

Method Mystery Output

```
2 magic: 24
We did it!
4 magic: 48
We did it!
```

Parameter Passing

Recall: Drawing a Cone

```
public static void drawCone() {
  // Draw the 5 lines of a cone
  for (int i = 0; i < 5; i++) {
    // Draw the spaces
    for (int j = 0; j < 4 - i; j++) {
      System.out.print(" ");
    }
    System.out.print("/");
    // Draw the dashes
    for (int j = 0; j < i * 2; j++) {
      System.out.print("-");
    }
    System.out.print("\\");
    System.out.println();
  }
}
```

```
   /\
  /--\
 /----\
/------\
/--------\
```

- Unsatisfactory!
  - Doesn't reflect our decomposition.
  - Too verbose as a result.
- Static methods to the rescue!

An Attempt At Refactoring

```
public static void drawCone() {
  // Draw the 5 lines of a cone
  for (int i = 0; i < 5; i++) {
    // Draw the spaces
    drawSpaces();
    System.out.print("/");
    // Draw the dashes
    for (int j = 0; j < i * 2; j++) {
      System.out.print("-");
    }
    System.out.print("\\");
    System.out.println();
  }
}
```

Scope of `i`

```
public static void drawSpaces() {
  for (int j = 0; j < 4 - i; j++) {
    System.out.print(" ");
  }
}
```

- `i` isn't in scope in drawSpaces!
- How can we *pass* the value of `i` from drawCone to drawSpaces?

Introduction to Parameters

```
public static void drawCone() {
  // Draw the 5 lines of a cone
  for (int i = 0; i < 5; i++) {
    // Draw the spaces
    drawSpaces(i);
    System.out.print("/");
    // Draw the dashes
    for (int j = 0; j < i * 2; j++) {
      System.out.print("-");
    }
    System.out.print("\\");
    System.out.println();
  }
}
```

```
public static void drawSpaces(int i) {
  for (int j = 0; j < 4 - i; j++) {
    System.out.print(" ");
  }
}
```

1. We *declare* that drawSpaces takes a parameter.
2. When we call drawSpaces, we *pass in* the value that we want the parameter to take.

Declaring Method Parameters

A *(formal) method parameter*.
"To call me, you must provide an int"

```
public static void printInt(int x) {
  System.out.println(x);
}
```

Inside a method, a parameter is just another local variable!

Passing in Values to Methods

```
public static void printInt(int x) {
  System.out.println(x);
}

public static void doWork() {
  printInt(5);
}
```

Passing the value 5 to printInt. To call a method that requires a parameter, you must pass a value of the correct type (here, int)

On each method call, the formal parameter variable is initialized with the *actual value* passed in.

Example: Executing Statements

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

Output

Example: Executing Statements (1)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 9)

Output

Example: Executing Statements (2)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 10)

i = 1

Output

Example: Executing Statements (3)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 10)

printAmps (line 3)

n = 1

Output

Example: Executing Statements (4)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 10)

printAmps (line 4)

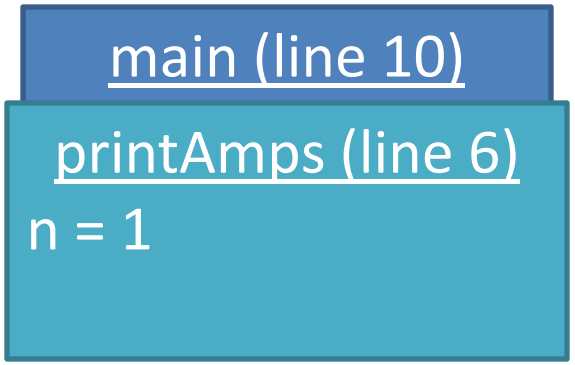
n = 1

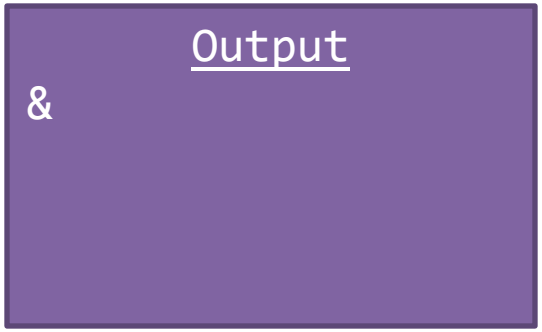
i = 0

Output

Example: Executing Statements (5)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```





Example: Executing Statements (6)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 11)

i = 1

Output

```
&
```

Example: Executing Statements (7)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 10)

i = 2

Output

&

Example: Executing Statements (8)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 10)

printAmps (line 3)

n = 2

Output

&

Example: Executing Statements (9)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 10)

printAmps (line 4)

n = 2

i = 0

Output

&

Example: Executing Statements (10)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 10)

printAmps (line 6)

n = 2

Output

```
&
&&
```

Example: Executing Statements (10)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 11)

i = 2

Output

&
&&

Example: Executing Statements (1)

```
1   public class Example {
2     public static void printAmps(int n) {
3       for (int i = 0; i < n; i++) {
4         System.out.print("&");
5       }
6     }
7
8     public static void main(String[] args) {
9       for (int i = 1; i <= 5; i++) {
10        printAmps(i);
11        System.out.println();
12      }
13    }
14  }
```

main (line 13)

Output

```
&
&&
&&&
&&&&
&&&&&
```

Reduce That Redundancy

```
System.out.println("Remove the cap from the peanut butter");
System.out.println("Scoop out some peanut butter.");
System.out.println("Spread it on a piece of bread.");
System.out.println("Remove the cap from the jelly");
System.out.println("Scoop out some jelly.");
System.out.println("Spread it on a piece of bread.");
```

```
public static void spread(String item) {
  System.out.println("Remove the cap from the " + item);
  System.out.println("Scoop out some " + item);
  System.out.println("Spread it on a piece of bread.");
}
```

```
spread("peanut butter");
spread("jelly");
```

- New opportunities for reducing redundancy!

Multiple Parameters

```
public static void repeat(String s, int n) {
  for (int i = 0; i < n; i++) {
    System.out.print(s);
  }
}
```

Multiple parameters can be specified with a comma-separate list of declarations.

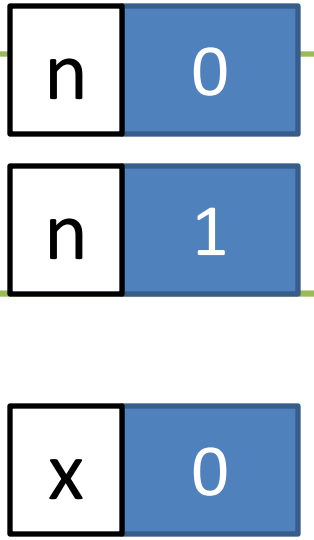
```
public static void main(String[] args) {
  repeat("+=", 3);
  repeat("*-", 5);
}
```

Likewise, each parameter requires a value when you call that method.

Passing in Values = Passing Copies

- We pass *copies of values* to methods.

```
public static void tryIncrement(int n) {
  n = n + 1;
}
public static void main(String[] args) {
  int x = 0;
  tryIncrement(x);
}
```



- Result: can't use a parameter to *change* an outside value.

Return Values

Recall Another Example

```
public class Cubes {
  public static void main(String[] args) {
    for (int i = 0; i < 5; i++) {
      System.out.println(i + “^3 = ” + i * i * i);
    }
  }
}
```

- Unsatisfactory (again)!
  - Ideally, i * i * i would be in its own method.
  - No way to have a method produce a value (yet!).

Return Values

```
public class Cubes {
  public static int cube(int i) {
    return i * i * i;
  }

  public static void main(String[] args) {
    for (int i = 0; i < 5; i++) {
      System.out.println(
        i + "^3 = " + cube(i));
    }
  }
}
```

Return type.
Specifies that cube(i) returns an int.

Return statement.
Tells the method to stop executing and *produce* the given value.

Now that cube(i) returns a value, it can be used as an expression!

Methods That Produce Values Are Expressions

- If a method returns a value, then it may be used as an expression of that type!

```
public static void main(String[] args) {
  int x = cube(1) + cube(2) + cube(3);
}
```

- Contrast with `println`:
  - `println` sends a value off to the screen.
  - Methods w/ return values can be used in computations.

Return Statements End Execution

```
public static int cube(int n) {
  return n * n * n;
  System.out.println("hello!");
}
```

Bad!

return statements end the execution of a method, so it makes no sense to have more statements afterwards!

Syntax of Methods Summary

```
public static <type> <name>(<type> <name>, ...) {
  <statement>;
  <statement>;
  …
  <statement>;
}
```