Introduction to Computer Programming

Lecture 5 – Methods and Parameters

Redundant Recipes

• Recipe for baking 20 cookies:
  – Mix the following ingredients in a bowl:
    • 4 cups flour
    • 1 cup butter
    • 1 cup sugar
    • 2 eggs
    • 40 pounds chocolate chips ...
  – Place on sheet and Bake for about 10 minutes.
Redundant Recipes (continued)

• Recipe for baking 40 cookies:
  – Mix the following ingredients in a bowl:
    • 8 cups flour
    • 2 cups butter
    • 2 cups sugar
    • 4 eggs
    • 80 pounds chocolate chips ...
  – Place on sheet and Bake for about 10 minutes.

  *Is this really different than the recipe we had for 20 cookies?*

What are parameters?

• A *parameter* is a value or a variable that is used to provide information to a function that is being called.

• If we are writing a function to calculate the square of a number, we can pass the value to be squared as a parameter:

  ```java
  area = square(5);
  y = square(x)
  ```

• These are called actual parameters because these are the actual values (or variables) used by the function being called.
Parameterized recipe

• Recipe for baking 20 cookies:
  – Mix the following ingredients in a bowl:
    • 4 cups flour
    • 1 cup sugar
    • 2 eggs
    • 80 pounds chocolate chips...
  – Place on sheet and Bake for about 10 minutes.

• Recipe for baking N cookies:
  – Mix the following ingredients in a bowl:
    • N/5 cups flour
    • N/20 cups butter
    • N/20 cups sugar
    • N/10 eggs
    • 2N bags chocolate chips...
  – Place on sheet and Bake for about 10 minutes.

Redundant figures

• Consider the task of printing the following lines/boxes:

*************
*******
******
*****************
**********
*****
* * *
* * *
A redundant solution

```java
public class Stars1 {
    public static void main(String[] args) {
        lineOf13();
        lineOf7();
        lineOf35();
        box10x3();
        box5x4();
    }

    public static void lineOf13() {
        for (int i = 1; i <= 13; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void lineOf7() {
        for (int i = 1; i <= 7; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void lineOf35() {
        for (int i = 1; i <= 35; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

- **line** - A method to draw a line of any number of stars.
- **box** - A method to draw a box of any size.
A redundant solution

• This code is redundant.
• Would variables help?
• Would constants help?
• What is a better solution?

Parameterization

• Parameters will allow us to write one method that we can use in many situations.
  – Instead of `lineOf7, lineOf13`, write `line` to draw any length.
    • When *declaring* the method, we will state that it requires a parameter for the number of stars.
    • When *calling* the method, we will specify how many stars to draw.
Formal Parameters

- Functions that use parameters must have them listed in the function header. These parameters are called *formal parameters*.

```java
public static void printSquare(double x) {
    double square;
    square = x*x;
    System.out.println("The square of " + x + " is " + square);
}
```

Declaring a parameter

- Stating that a method requires a parameter in order to run

```java
public static void name (type name) {
    statement(s);
}
```

- Example:

```java
public static void sayPassword(int code) {
    System.out.println("The password is: " + code);
}
```
Passing a parameter

- **Calling a method and specifying values for its parameters**
  name (expression);
- Example:
  ```java
  public static void main(String[] args) {
    sayPassword(42);
    sayPassword(12345);
  }
  ```
  - Output:
    The password is 42
    The password is 12345

Parameter Passing

```java
printSquare(5);
printSquare(x)
public static void printSquare(double x) {
  double square;
  square = x * x;
  System.out.println("The square of "+ x + " is "+ square);
}
```

*In both cases, calling the function requires copying the actual parameter’s value where the function can use it. Initially, x has whatever value the actual parameter has.*
Parameter Passing (continued)

```java
public static void printSquare(double x) {
    double square;
    square = x*x;
    System.out.println("The square of "+ x + " is " + square);
}
```

- `x` initially is set to 5.
- `square` is then set to the value of $x^2$ or $5^2$ or 25.

Parameter Passing (continued)

```java
public static void printSquare(double x) {
    double square;
    square = x*x;
    System.out.println("The square of "+ x + " is " + square);
}
```

- `x` initially is set to whatever value `x` had in the main program. If `x` had the value 12, `square` is then set to the value of $x^2$ or $12^2$ or 144.
Why parameters?

- Parameters are useful because:
  - They allow us to use the same function in different places in the program and to work with different data.
  - They allow the main program to communicate with the function and pass it whatever data it is going to use.
  - The same value can have completely different names in the main program and in the function.

The **Squares** Program

```java
import java.util.Scanner;

public class Squares {
    // main() - A driver for the print_square function
    public static void main(String[] args) {
        Scanner keyb = new Scanner(System.in);
        double value;

        // Get a value and print its square
        System.out.println("Enter a value ");
        value = keyb.nextDouble();
        printSquare(value);
    }
}
```

*the actual parameter in the function call*
// printSquare() - Prints the square of whatever
//                 value that it is given.
public static void print_square(double x) {
    double square;
    square = x*x;
    System.out.println("The square of "+ x
                        + " is "+ square);
}
Passing Parameters - When The User Inputs 6

A Rewrite of `main`

```java
import java.util.Scanner;

public class Squares2 {
    // main() - A driver for the print_square function
    public static void main(String[] args) {
        double value1 = 45, value2 = 25;

        printSquare(value1);
        printSquare(value2);
    }
}
```
Parameters and loops

• A parameter can guide the number of repetitions of a loop.

```java
public static void main(String[] args) {
    chant(3);
}

public static void chant(int times) {
    for (int i = 1; i <= times; i++) {
        System.out.println("Just a salad...");
    }
}
```

What is the output?
Common Errors

• If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.
  ```java
callMethod(); // ERROR: parameter
              // value required
  ```

• The value passed to a method must be of the correct type.
  ```java
callMethod(3.7); // ERROR: must be
                   // of type int
  ```

Stars Solution

```
// Prints several lines of stars.
// Uses a parameterized method to remove
// redundancy.
public class Stars2 {
    public static void main(String[] args) {
        line(13);
        line(7);
        line(35);
    }
}
```
// Prints the given number of stars plus a
// line break.
public static void line(int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print("*");
    }
    System.out.println();
}

Multiple Parameters

- A method can accept multiple parameters. (separate by ,)
- When calling it, you must pass values for each parameter.
- Declaration:
  public static void name(type name, ..., type
  name) {
      statement(s);
  }
- Call:
  methodName (value, value, ..., value);
Multiple Params Example

```java
public static void main(String[] args) {
    printNumber(4, 9);
    printNumber(17, 6);
    printNumber(8, 0);
    printNumber(0, 8);
}

public static void printNumber(int number, int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(number);
    }
    System.out.println();
}
```

Output:

```
444444444
171717171717
00000000
```

Stars Solution

```java
// Prints several lines and boxes made of stars.
// Third version with multiple parameterized methods.

public class Stars3 {
    public static void main(String[] args) {
        // Draw some lines
        line(13);
        line(7);
        line(35);
        System.out.println();

        // Draw some boxes
        box(10, 3);
        box(5, 4);
        box(20, 7);
    }
}
```
// Prints the given number of stars plus
// a line break.
public static void line(int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print("*");
    }
    System.out.println();
}

// Prints a box of stars of the given size.
public static void box(int width,
                        int height) {
    line(width);
    for (int line = 1; line <= height - 2;
         line++) {
        System.out.print("*");
        for (int space = 1;
             space <= width - 2; space++) {
            System.out.print(" ");
        }
        System.out.println("*");
    }
    line(width);
}
Strings

- **String** - A sequence of text characters.

  ```java
  String name = "text";
  String name = expression;
  ```

- Examples:

  ```java
  String name = "Marla Singer";

  int x = 3;
  int y = 5;
  String point = "(" + x + ", " + y + ")";
  ```

Strings As Parameters

```java
public class StringParameters {
    public static void main(String[] args) {
        sayHello("Marty");
        String teacher = "Bictolia";
        sayHello(teacher);
    }

    public static void sayHello(String name) {
        System.out.println("Welcome, " + name);
    }
}
```

**Output:**

Welcome, Marty
Welcome, Bictolia.
Strings As Parameters

- Modify the **Stars** program to use string parameters. Use a method named `repeat` that prints a string many times.

Stars Solution

```java
// Prints several lines and boxes made of stars.
// Fourth version with String parameters.

public class Stars4 {
    public static void main(String[] args) {
        line(13);
        line(7);
        line(35);
        System.out.println();
        box(10, 3);
        box(5, 4);
        box(20, 7);
    }
}
```
// Prints the given number of stars plus a
// line break.
public static void line(int count) {
    repeat("*", count);
    System.out.println();
}

// Prints a box of stars of the given size.
public static void box(int width,
                        int height) {
    line(width);
    for (int line = 1; line <= height - 2;
         line++) {
        System.out.print("*");
        repeat(" ", width - 2);
        System.out.println("*");
    }
    line(width);
}

// Prints the given String the given number
// of times.
public static void repeat(String s,
                          int times) {
    for (int i = 1; i <= times; i++) {
        System.out.print(s);
    }
}
Java's **Math** Class

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.abs(value)</td>
<td>absolute value</td>
</tr>
<tr>
<td>Math.ceil(value)</td>
<td>rounds up</td>
</tr>
<tr>
<td>Math.floor(value)</td>
<td>rounds down</td>
</tr>
<tr>
<td>Math.log10(value)</td>
<td>logarithm, base 10</td>
</tr>
<tr>
<td>Math.max(value1, value2)</td>
<td>larger of two values</td>
</tr>
<tr>
<td>Math.min(value1, value2)</td>
<td>smaller of two values</td>
</tr>
<tr>
<td>Math.pow(base, exp)</td>
<td>base to the exp power</td>
</tr>
</tbody>
</table>

Java's **Math** Class

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.random()</td>
<td>random double between 0 and 1</td>
</tr>
<tr>
<td>Math.round(value)</td>
<td>nearest whole number</td>
</tr>
<tr>
<td>Math.sqrt(value)</td>
<td>square root</td>
</tr>
<tr>
<td>Math.sin(value)</td>
<td>sine/cosine/tangent of an angle in radians</td>
</tr>
<tr>
<td>Math.cos(value)</td>
<td></td>
</tr>
<tr>
<td>Math.tan(value)</td>
<td></td>
</tr>
<tr>
<td>Math.toDegrees(value)</td>
<td>convert degrees to radians and back</td>
</tr>
<tr>
<td>Math.toRadians(value)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.E</td>
<td>2.7182818...</td>
</tr>
<tr>
<td>Math.PI</td>
<td>3.1415926...</td>
</tr>
</tbody>
</table>
Calling **Math** Methods

**Math**.methodName(parameters)

- Examples:

  ```java
  double squareRoot = Math.sqrt(121.0);
  System.out.println(squareRoot);  // 11.0

  int absoluteValue = Math.abs(-50);
  System.out.println(absoluteValue);  // 50

  System.out.println(Math.min(3, 7) + 2);  // 5
  ```

Calling **Math** Methods

- The **Math** methods do not print to the console.
  - Each method produces ("returns") a numeric result.
  - The results are used as expressions (printed, stored, etc.).
Math Questions

• Evaluate the following expressions:
  - `Math.abs(-1.23)`
  - `Math.pow(3, 2)`
  - `Math.pow(10, -2)`
  - `Math.sqrt(121.0) - Math.sqrt(256.0)`
  - `Math.ceil(6.022) + Math.floor(15.9994)`
  - `Math.abs(Math.min(-3, -5))`

Methods and Functions

• Some methods perform specific tasks and do not produce any one data item that seem to be their whole reason for existence.

• Other methods are all about producing some value or data item; in many programming languages they are called `functions`. 
void Functions

- Normally a function is expected to produce some result which is returned to the main program:
  ```java
  average = calcAverage(x, y, z);
  ```
- The data type of the function’s result is also called the function’s type.
  - Functions that produce an integer are called integer functions.
  - Functions that produce float value are called float functions.
  - Functions that do not produce a result are called void functions.
- When we write
  ```java
  public void printSquare(int x);
  ```
  it means that the function is not expected to return a result.

Writing Functions That Return Results

- We can write a function that returns a result by replacing that void with a data type:
  ```java
  public double average3(int a, int b, int c);
  ```
- The rest of the function is a little different from before:
  ```java
  public double average3(int a, int b, int c) {
    float sum, mean;
    sum = a + b + c;
    mean = sum / 3;
    return mean;
  }
  ```
  The result that we are returning is mean.
Writing Functions That Return Results

- The syntax is:
  ```java
  return expression;
  ```
- Return statements have contain expressions, variables, constants or literals:
  ```java
  return true;
  return 35.4;
  return sum;
  return sum/3;
  ```

The Function `average3`

```java
public double average3(int a, int b, int c) {
    float sum, mean;
    sum = a + b + c;
    return sum / 3;
}
```
The Program \textbf{PowerTest}

```java
import java.util.Scanner;

public class PowerTest {
    // A program to calculate 4-cubed using a
    // function called power
    public static void main(String[] args) {
        double x, y;
        int n;

        x = 4.0;
        n = 3;
        y = power(x, n);
        System.out.println("The answer is "+y);
    }

    // power() - Calculates y = x to the nth
    // power
    public static double power(double x, int n) {
        double prod;

        prod = 1.0;
        while (n > 0) {
            prod = prod * x;
            n = n - 1;
        }
        System.out.println("Our result is "+prod);
        return prod;
    }
}
```
An Example – square2

• Let’s write a program where the function calculates the square and passes its value back to the main program, which will print the result:

```java
import java.util.Scanner;

public class Square2 {
    // This illustrates how to use methods to find the square of a value

    // main() - A driver for the findSquare method
    public static void main(String[] args) {
        Scanner keyb = new Scanner(System.in);
        double value, square;

        System.out.println("Enter a value ?");
        value = keyb.nextDouble();
        square = findSquare(value);
        System.out.println("The square of " + value + " is " + square);
    }

    // findSquare() - Calculates the square of whatever value it is given.
    public static double findSquare(double x) {
        double square = x * x;
        return square;
    }
}
```
Quirks Of Real Numbers

• Some Math methods return double or other non-int types.
  
  ```java
  int x = Math.pow(10, 3);
  // ERROR: incompat. types
  ```

• Some double values print poorly (too many digits).
  
  ```java
  double result = 1.0 / 3.0;
  System.out.println(result);
  // 0.33333333333333333
  ```

Quirks Of Real Numbers

• The computer represents doubles in an imprecise way.

  ```java
  System.out.println(0.1 + 0.2);
  ```

  – Instead of 0.3, the output is

  ```java
  0.30000000000000004
  ```
Type Casting

- **type cast** - A conversion from one type to another.
  - To promote an `int` into a `double` to get exact division from `/`
  - To truncate a `double` from a real number to an integer

Type Casting

- Syntax:
  
  (type) expression

- Examples:

  ```java
  double result = (double) 19 / 5;    // 3.8
  int result2 = (int) result;          // 3
  int x = (int) Math.pow(10, 3);      // 1000
  ```
More About Type Casting

• Type casting has high precedence and only casts the item immediately next to it.

```java
double x = (double) 1 + 1 / 2;       // 1
double y = 1 + (double) 1 / 2;       // 1.5
```

More About Type Casting

• You can use parentheses to force evaluation order.

```java
double average = (double) (a + b + c) / 3;
```

• A conversion to `double` can be achieved in other ways.

```java
double average = 1.0 * (a + b + c) / 3;
```
Returning A Value

public static type name(parameters) {
    statements;
    ...
    return expression;
}

Returning A Value – An Example

// Returns the slope of the line between
// the given points.
public static double slope(int x1, int y1,
                           int x2, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    return dy / dx;
}

- slope(1, 3, 5, 11) returns 2.0
Return Examples

// Converts degrees Fahrenheit to Celsius.
public static double fToC(double degreesF) {
    double degreesC = 5.0 / 9.0 * (degreesF - 32);
    return degreesC;
}

// Computes triangle hypotenuse length
// given its side lengths.
public static double hypotenuse(int a, int b) {
    double c = Math.sqrt(a * a + b * b);
    return c;
}

Return Examples

• You can shorten the examples by returning an expression:

    public static double fToC(double degreesF) {
        return 5.0 / 9.0 * (degreesF - 32);
    }
Common Error: Not Storing

• Many students incorrectly think that a return statement sends a variable's name back to the calling method.

Common Error: Not Storing - Example

```java
public static void main(String[] args) {
    slope(0, 0, 6, 3);
    System.out.println("The slope is " + result);
    // ERROR: result not defined
}

public static double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```
Fixing The Common Error

• Instead, returning sends the variable's **value** back.
  – The returned value must be stored into a variable or used in an expression to be useful to the caller.

Fixing The Common Error - Example

```java
public static void main(String[] args) {
    double s = slope(0, 0, 6, 3);
    System.out.println("The slope is "+s);
}

public static double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```