Topic 8
Parameters and Methods

"We're flooding people with information. We need to feed it through a processor. A human must turn information into intelligence or knowledge. We've tended to forget that no computer will ever ask a new question."

— Rear Admiral Grace Murray Hopper "

Based on slides for Building Java Programs by Reges/Stepp, found at http://faculty.washington.edu/stepp/book/

Reminder: global constants

- In the last topic, we used global constants to fix "magic number" redundancy problems:

```
public static final int SIZE = 3;

public static void main(String[] args) {
    drawDiamond();
    System.out.println();
    drawX();
}

public static void drawDiamond(){
    drawCone();
    drawV();
}

public static void drawX(){
    drawV();
    drawCone();
}
```

drawCone() method

```
public static void drawCone(){
    //draws a cone with SIZE lines
    for(int lineNum = 1; lineNum <= SIZE; lineNum++){

        //spaces before the forward slash
        for(int j = 1; j <= SIZE - lineNum; j++)
            System.out.print(" ");

        System.out.print("/");

       //spaces between the forward slash and back slash
        for(int j = 1; j <= (lineNum - 1) * 2; j++)
            System.out.print(" ");

        System.out.println("\\");
    }
}
```

drawV() method

```
public static void drawV(){

    //draws a V with SIZE lines
    for(int lineNum = 1; lineNum <= SIZE; lineNum++){

        // print spaces before back slash
        for(int j = 1; j <= lineNum - 1; j++)
            System.out.print(" ");

        System.out.print("\\");

        // print spaces between back slash and forward slash
        for(int j = 1; j <= (SIZE - lineNum) * 2; j++)
            System.out.print(" ");

        System.out.println("/");
    }
}
```

Another repetitive figure

- Now consider the task of drawing the following figures:

```
**********

*******

**********
*        *
**********

*****
*   *
*   *
*****
```

- We'd like to structure the input using static methods, but each figure is different.
  - What can we do?
- Note on drawing figures: It is the ability to generalize and break the problem into smaller steps that is important!

A poor solution

- Using what we already know, we could write a solution with the following methods:
  - A method to draw a line of 13 stars.
  - A method to draw a line of 7 stars.
  - A method to draw a box of stars, size 10 x 3.
  - A method to draw a box of stars, size 5 x 4.

- These methods would be largely redundant.

- Constants would *not* help us solve this problem, because we do not want to share one value but want to run similar code with many values.

A poor solution

```
public static void oneLine13stars()

public static void oneLine7stars()

public static void draw10By3Box()

public static void draw5By4Box()

public static void draw12By100Box()
```

A better solution

- A better solution would look something like this:
  - A method to draw a line of any number of stars.
  - A method to draw a box of stars of any size.

- It is possible to write *parameterized* methods; methods that are passed information when they are called which affects their behavior.
  - Example: A parameterized method to draw a line of stars might ask us to specify how many stars to draw.

- **parameter**: A value given to a method by its caller, which takes the form of a variable inside the method's code. The method can use the value of this variable to adjust its behavior.

Methods with parameters

- Declaring a method that accepts a parameter:

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

– Example:

```
// Prints the given number of spaces.
public static void printSpaces(int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(" ");
    }
}
```

Does the declaration look like any other kind of statement we have already learned?

Passing parameters

- Calling a method and specifying a value for its parameter is called *passing a parameter.*
- Method call with passing parameter syntax:

```
<name>( <value> );
```

– Example:

```
System.out.print("*");
printSpaces(7);
System.out.print("**");
int x = 3 * 5;
printSpaces(x + 2);
System.out.println("***");
```

– Output:

```
*       **                      ***
```

How parameters are passed

- **formal parameter**: The *variable* declared in the method. Sometimes just called the parameter
- **actual parameter**: The *value* written between the parentheses in the call. Sometimes just called the argument
  - When the program executes, the actual parameter's value is stored into the formal parameter variable, then the method's code executes.

```
printSpaces( 13 );
...

public static void printSpaces(int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(" ");
    }
}
```

`count` will take the value `13` for this method call

Parameterized figure

- This code *parameterizes* the lines of stars:

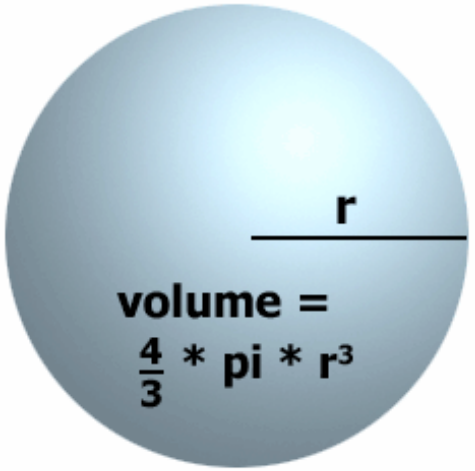
```
public static void main(String[] args) {
    drawLineOfStars(13);
    System.out.println();

    drawLineOfStars(7);
}

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

Output:

```
*************

*******
```

Creating general solutions

- The ability to parameterize problems is a way of generalizing them, and this is a VERY important skill in programming!



Parameter details

- If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.
  - `printSpaces();  // SYNTAX ERROR`
- The actual parameter value passed to a method must be of the correct type, matching the type of the formal parameter variable.
  - `printSpaces(3.7);  // SYNTAX ERROR must int`
- Two methods may have the same name as long as they accept different parameters (this is called **overloading**).

```
public static void printLineOfStars()
{ ... }
public static void printLineOfStars(int length)
{ ... }
```

Value parameter behavior

- **value parameter**: When primitive variables (such as int or double) are passed as parameters in Java, their values are copied and stored into the method's formal parameter.
  - Modifying the formal parameter variable's value will not affect the value of the variable which was passed as the actual parameter.

```
int x = 23;
strange(x);
System.out.println(x);   // this x is unaffected
...

public static void strange(int x) {
    x = x + 1;           // modifies the x in strange
    System.out.println(x);
}
```

Draw the boxes to understand the behavior!

Output:

```
24
23
```

Multiple parameters

- A method can accept more than one parameter, separated by commas:

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

  - Example:

```
public static void printPluses(int lines, int count)
  {
    for (int line = 1; line <= lines; line++) {
        for (int i = 1; i <= count; i++) {
            System.out.print("+");
        }
        System.out.println();
    }
}
```

Parameterized box figure

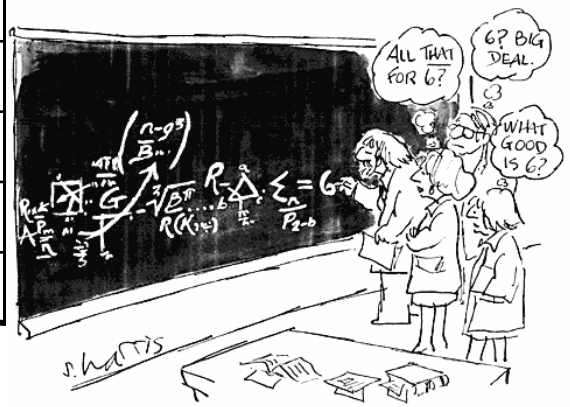
- This code parameterizes the boxes of stars:

```
public static void main(String[] args) {
    drawBoxOfStars(10, 3);
    System.out.println();
    drawBoxOfStars(4, 5);
}

public static void drawBoxOfStars(int width, int height)
    drawLineOfStars(width);

    for (int line = 1; line <= height - 2; line++) {
        System.out.print("*");
        printSpaces(width - 2);
        System.out.println("*");
    }
    drawLineOfStars(width)
}
```

Java's `Math` class

- Java has a class called `Math` that has several useful methods that perform mathematical calculations.

Method name	Description
abs(*value*)	absolute value
cos(*value*)	cosine, in radians
log(*value*)	logarithm base *e*
max(*value1*, *value2*)	larger of two values
min(*value1*, *value2*)	smaller of two values
pow(value1, value2)	*value1* to the *value2* power
random()	random `double` between 0 and 1
round(*value*)	returns nearest whole number
sin(*value*)	sine, in radians
sqrt(*value*)	square root



  - The Math methods are called by writing:
    `Math.` ***<name>*** `(` ***<values>*** `);`

Methods that "return" values

- The methods of the `Math` class do not print their results to the console.
  - Instead, a call to one of these methods can be used as an expression or part of an expression.
  - The method evaluates to produce (or *return*) a numeric result.
  - The result can be printed or used in a larger expression.

  - Example:

```
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
```

Methods that return values

- Declaring a method that returns a value:

```
public static <type> <name> ( <type> <name> ) {
    <statement(s)> ;
}
```

- Returning a value from a method:

```
return <value> ;
```

- Example:

```
// Returns the given number cubed (to the third power).
public static int cube(int number) {
    return number * number * number;
}
```

Example returning methods

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

// Rounds the given real number to the
  nearest whole number.
// Examples: round(3.1415) returns 3
  and round(4.75) returns 5.
public static int round(double value) {
    return (int) (value + 0.5);
}
```

How to comment: methods

- If your method accepts parameters and/or returns a value, write a brief description of what the parameters are used for and what kind of value will be returned.
  - In your comments, you can also write your assumptions about the values of the parameters.
  - You may wish to give examples of what values your method returns for various input parameter values.
  - Example:

```
// This method returns the factorial of the given integer n.
// The factorial is the product of all integers up to that number.
// I assume that the parameter value is non-negative.
// Example: factorial(5) returns 1 * 2 * 3 * 4 * 5 = 120.
public static int factorial(int n) {
    //implementation of factorial
}
```