Parameters
Repetitive figures

- Consider the task of drawing the following figures:

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

*********

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

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

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

*****
  *
  *
```

- The lines and figures are similar, but not exactly the same.
A solution?

- **Observation**: Methods are redundant.
- Would constants help us solve this problem?
- Other ideas?
Generalizing methods

What if we had the following?

- `drawLine` - A method to draw a line of any number of stars.
- `drawBox` - A method to draw a box of any size.
Parameterization

- **parameterized method**: A method that is given extra information (e.g. number of stars to draw) when it is called.

- **parameter**: A value passed to a method by its caller.

**Examples:**
- `System.out.println(“*”)` or `System.out.println(“**”)`
- `random.nextInt(10)`
Declaring parameterized methods

- Parameterized method declaration syntax:
  ```java
  public static void <name> (<type> <name>) {
      <statement(s)>;
  }
  ```

- The scope of the parameter is the entire method.

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

- Whenever `printSpaces` is called, the caller must specify how many spaces to print.
Calling parameterized methods

- **passing a parameter**: Calling a parameterized method and specifying a value for its parameter(s).

  Parameterized method call syntax:
  `<name> (<expression>) ;`

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

  Output:
  ```
  *     **
  ***
  ```
Passing parameters

- When the parameterized method call executes:
  - the value passed to the method is *copied* into the parameter variable
  - the method's code executes using that value

```java
public static void main(String[] args) {
    printSpaces(7);
    printSpaces(13);
}

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

- **value semantics**: When primitive variables (such as `int` or `double`) are passed as parameters, their values are *copied* into the method's parameter variable.
  - Modifying the method’s parameter variable will **NOT** affect the variable which was passed to the method.

```java
...  
int x = 23;
strange(x);
System.out.println("2. x = " + x); // this x unchanged
...

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

**Output:**

1. x = 24  
2. x = 23
Errors in coding

- **ERROR**: Not passing a parameter to a method that accepts parameters.

  ```java
  printSpaces();  // ERROR: parameter value required
  ```

- **ERROR**: Passing a parameter of the wrong type.

  ```java
  printSpaces(3.7);  // ERROR: must be of type int
  ```
  - The parameter must satisfy the *domain* of the method.
Parameters: Exercise

- Change the Stars program to use parameterized methods.

```java
public class Stars {
    public static void main(String[] args) {
        drawLineOf13Stars();
        drawLineOf7Stars();
        drawLineOf35Stars();
        draw10x3Box();
        draw5x4Box();
    }

    public static void drawLineOf13Stars() {
        for (int i = 1; i <= 13; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void drawLineOf7Stars() {
        for (int i = 1; i <= 7; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void drawLineOf35Stars() {
        for (int i = 1; i <= 7; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
...
// Prints several lines of stars.
// Uses a parameterized method to remove redundancy.

class Stars2 {
    public static void main(String[] args) {
        drawLine(13);
        drawLine(7);
        drawLine(35);
    }

    // Prints the given number of stars plus a line break.
    public static void drawLine(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
Multiple parameters

- Methods can accept as many parameters as you like.
  - When the method is called, it must be passed values for each of its parameters.

- Multiple parameters declaration syntax:
  ```java
  public static void <name> (<type> <name>,
      <type> <name>, ..., <type> <name>) {
      <statement(s)>;
  }
  ```

- Multiple parameters call syntax:
  ```java
  <name>(<expression>, <expression>, ..., <expression>);
  ```
Multiple parameters: Solution

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

public class Stars3 {
    public static void main(String[] args) {
        drawLine(13);
        drawLine(7);
        drawLine(35);
        System.out.println();
        drawBox(10, 3);
        drawBox(5, 4);
        drawBox(20, 7);
    }

    // Prints the given number of stars plus a line break.
    public static void drawLine(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 drawBox(int width, int height) {
    drawLine(width);

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

    drawLine(width);
}

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

Multiple parameters: Exercise

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

**Output:**

```
444444444
171717171717
00000000
```
public static void main(String[] args) {
    printNumber(4, 9);
    printNumber(17, 6);
    printNumber(8, 0);
    printNumber(0, 8);
}

Output:
444444444
171717171717
00000000

public static void printNumber(int number, int count) {
    for(int i=0; i<count; i++) {
        System.out.print(number);
    }
    System.out.println();
}
What is the output of the following program?

```java
public class Mystery {
    public static void main(String[] args) {
        int x = 5, y = 9, z = 2;
        mystery(z, y, x);
        System.out.println(x + " " + y + " " + z);
        mystery(y, x, z);
        System.out.println(x + " " + y + " " + z);
    }

    public static void mystery(int x, int z, int y) {
        x++;
        y = x - z * 2;
        x = z + 1;
        System.out.println(x + " " + y + " " + z);
    }
}
```
Exercises

- Write a method named `printDiamond` that accepts a height as a parameter and prints a diamond figure.
  
  ```
  *
  ***
  *****
  ***
  *
  ```

- Write a method named `multiplicationTable` that accepts a maximum integer as a parameter and prints a table of multiplication from 1 x 1 up to that integer times itself.

- Write a method named `bottlesOfBeer` that accepts an integer as a parameter and prints the "Bottles of Beer" song with that many verses.
Exercise

- Rewrite the following program to use parameterized methods:

```java
// Draws triangular figures of stars.
public class Loops {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i - 1; j++) {
                System.out.print(" ");
            }
            for (int j = 1; j <= 10 - 2 * i + 1; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
        for (int i = 1; i <= 12; i++) {
            for (int j = 1; j <= i - 1; j++) {
                System.out.print(" ");
            }
            for (int j = 1; j <= 25 - 2 * i; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```
Solution

// Draws triangular figures using parameterized methods.
public class Loops {
    public static void main(String[] args) {
        triangle(5);
        triangle(12);
    }

    // Draws a triangle figure of the given size.
    public static void triangle(int height) {
        for (int i = 1; i <= height; i++) {
            printSpaces(i - 1);
            drawLine(2 * height + 1 - 2 * i);
        }
    }
}

Methods that return values
Return values

- **return**: To send a value out as the result of a method, which can be used in an expression.

- A return value is like the opposite of a parameter.
  - Parameters pass information *in* from the caller to the method.
  - Return values pass information *out* from a method to its caller.

- How would this be useful?
Java has a class called `Math` that has several useful static methods to perform mathematical calculations.

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(value)</td>
<td>absolute value</td>
</tr>
<tr>
<td>cos(value)</td>
<td>cosine, in radians</td>
</tr>
<tr>
<td>log(value)</td>
<td>logarithm base e</td>
</tr>
<tr>
<td>log10(value)</td>
<td>logarithm base 10</td>
</tr>
<tr>
<td>max(value1, value2)</td>
<td>larger of two values</td>
</tr>
<tr>
<td>min(value1, value2)</td>
<td>smaller of two values</td>
</tr>
<tr>
<td>pow(base, exponent)</td>
<td>base to the exponent power</td>
</tr>
<tr>
<td>random()</td>
<td>random double between 0 and 1</td>
</tr>
<tr>
<td>round(value)</td>
<td>nearest whole number</td>
</tr>
<tr>
<td>sqrt(value)</td>
<td>square root</td>
</tr>
</tbody>
</table>
Using the Math class methods

- **Math method call syntax:**
  Math. `<method name>`( `<parameter(s)>`)

- **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
  ```

- Notice that the preceding calls are used in expressions; they can be printed, stored into a variable, etc…
Return values

- The Math methods do not print results to the console.
- Instead, each method evaluates to produce (or return) a numeric result, which can be used in an expression.
Exercises

- 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))`

- `Math.max` and `Math.min` can be used to bound numbers. Consider an `int` variable named `age`.
  - What statement would replace negative ages with `0`?
  - What statement would cap the maximum age to `40`?
Writing methods that return values

- Declaring a method that returns a value:
  
  ```java
  public static <type> <name>(<parameters>) {
    <statement(s)>;
  }
  ```

- Returning a value from a method:
  
  ```java
  return <expression>;
  ```

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

- Question: What return type have we used up until now?
// Converts Fahrenheit to Celsius.
public static double fToC(double degreesF) {
    return 5.0 / 9.0 * (degreesF - 32);
}

// Rounds the given number to the given number of decimal places.
// Example: round(3.14159265, 4) returns 3.1416.
public static double round(double value, int places) {
    double pow = Math.pow(10, places);
    value = value * pow;        // upscale the number
    value = Math.round(value);  // round to nearest whole number
    value = value / pow;        // downscale the number
    return value;
}
Errors in coding

- **ERROR**: Writing statements after a `return` statement.

```java
public static int increment(int x) {
    return (x + 1);
    x = x + 1;  // ERROR: statement unreachable!
}
```

- **ERROR**: Confusing the return variable with a variable in the calling method, AKA ignoring the return value.

```java
public class ReturnExample {
    public static int addOne(int x) {
        x = x + 1;
        return x;
    }

    public static void main(String[] args) {
        int x = 1;
        addOne(x);
        System.out.println("x = " + x);
    }
}
```
Exercises

- Write a method called `power` that accepts a `base` and an `exponent` as parameters, and returns the value of $base^{exponent}$. (Use a loop, not `Math.pow()`.)

- Write a method called `isFactor` that accepts two integers as parameters, and returns `true` if the first is a factor of the second.

- Write a method named `distanceFromOrigin` that accepts x and y coordinates as parameters and returns the distance between that (x, y) point and the origin.
Methods can return different values under different conditions:

```java
public static int min(int a, int b) {
    if (a > b) {
        return b;
    } else {
        return a;
    }
}

public static String message(int place) {
    if (place == 1) {
        return "You won!";
    } else {
        return "If you're not first, you're last!";
    }
}
```
Errors in coding

```java
public static int min(int a, int b) {
    if (a > b) {
        return b;
    }
}
```

- The compiler will complain about a "missing return statement". Why?

- ERROR: Not returning a value in every path. In the above example, what if \( a \leq b \)?
public static int min(int a, int b) {
    if (a > b) {
        return b;
    } else if (a <= b) {
        return a;
    }
}

- It still produces the "missing return statement" error. Why?
  - To our eyes, it is clear that all paths (greater, equal, less) do return a value.
  - But the compiler thinks that if/else if code might choose not to execute any branch, so it refuses to accept this code.

- How can we fix it?
Exercise: Counting primes

- Write a program that prompts the user for a maximum integer and prints out a list of all prime numbers up to that maximum. Here is an example log of execution:

  Maximum number? 50
  2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47
  15 total primes
import java.util.*;

public class PrintPrimes {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        printPrimes(getNumber(console));
    }

    public static int countFactors(int num) {
        int count = 0;
        for (int i = 1; i <= num; i++) {
            if (num % i == 0) {
                count++;
            }
        }
        return count;
    }

    ...
Solution: Counting primes

```java
public static int getNumber(Scanner console) {
    System.out.print("Maximum number? ");
    return console.nextInt();
}

public static void printPrimes(int max) {
    int numPrimes = 0;
    if (max >= 2) {
        System.out.print(2);
        numPrimes++;
        for (int i = 3; i <= max; i++) {
            if (countFactors(i) == 2) {
                numPrimes++;
                System.out.print(" , "+i);
            }
        }
        System.out.println();
    }
    System.out.println(numPrimes + " total primes");
}
```