جامعة بيروت العربية
BEIRUT ARAB UNIVERSITY

CMPS 241
Introduction to Programming

Static Methods

Parameters and Return Values

Redundant figures

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

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

*******

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

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

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

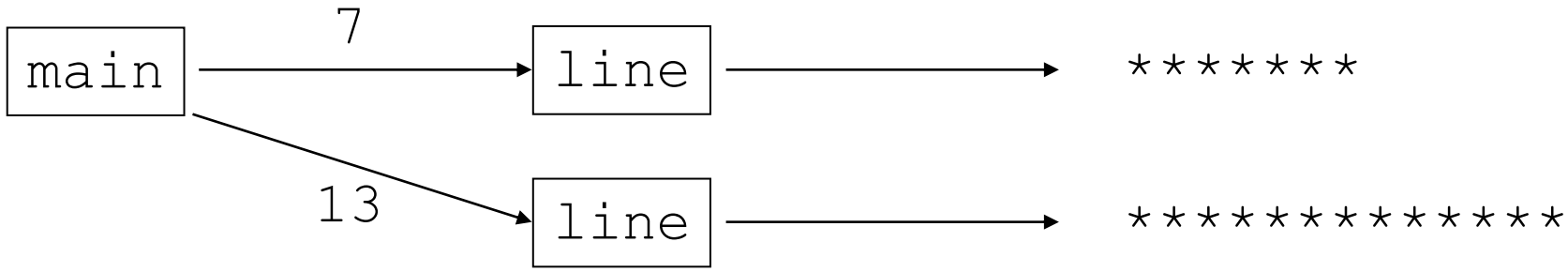
A redundant solution

```
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();
    }
    ...
```

- This code is redundant.
- Would variables help?
  Would constants help?
- What is a better solution?
  - `line` - A method to draw a line of any number of stars.
  - `box` - A method to draw a box of any size.

Parameterization

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



Declaring a parameter

Stating that a method requires a parameter (argument) in order to run

```
public static void methodName ( type paramName ) {
    statement(s);
}
```

- Example:

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

  – When `sayPassword` is called, the caller must specify the integer code to print.

Passing a parameter

Calling a method and specifying values for its parameters

```
methodName (expression);
```

- Example:

```
public static void main(String[] args) {
    sayPassword(42);
    sayPassword(12345);
}
```

Output:

```
The password is 42
The password is 12345
```

Parameters and loops

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

```
public static void main(String[] args) {
    chant(3);
    chant(7);
}

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

Common errors

- If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.

```
chant();        // ERROR: parameter
value required
```

- The value passed to a method must be of the correct type.

```
chant(3.7);     // ERROR: must be of
type int
```

Value semantics

- **value semantics**: When primitive variables (`int`, `double`) are passed as parameters, their values are ***copied***.
  - Modifying the parameter will not affect the variable passed in.

```
public static void main(String[] args) {
    int x = 23;
    strange(x);
    System.out.println("2. x = " + x);
    ...
}
public static void strange(int x) {
    x = x + 1;
    System.out.println("1. x = " + x);
}
```

Output:

```
1. x = 24
2. x = 23
```

Value semantics Example

```
public class ParameterExample{
    public static void main(String[] args) {
        int x = 17;
        doubleNumber(x);
        System.out.println(x); ⟶   17
        int number = 42;
        doubleNumber(number);
        System.out.println(number); ⟶   42
    }

    public static void doubleNumber(int number) {
        System.out.println(number); ⟶   17   42
        number = number * 2;
        System.out.println(number); ⟶   34   84
    }
}
```

Factorial – Solution 1

```
public class Fact{
     public static void main(String[] args) {

          // print factorial of 4
          int n = 4;
          int fact = 1;
          for(int i = 2; i <= n ; i++) {
               fact *= i;
          }
          System.out.println(fact);

          // print factorial of 7
          n = 7;
          fact = 1;
          for(int i = 2; i <= n ; i++) {
               fact *= i;
          }
          System.out.println(fact);

          // print factorial of 9
          n = 9;
          fact = 1;
          for(int i = 2; i <= n ; i++) {
               fact *= i;
          }
          System.out.println(fact);
     }
}
```

Factorial – Solution 2

```java
public static void main(String[] args) {
    fact4();
    fact7();
    fact9();
}
```

```java
public static void fact4() {
    int n = 4;
    int fact = 1;
    for(int i = 2; i <= n ; i++) {
        fact *= i;
    }
    System.out.println(fact);
}

public static void fact7() {
    int n = 7;
    int fact = 1;
    for(int i = 2; i <= n ; i++) {
        fact *= i;
    }
    System.out.println(fact);
}

public static void fact9() {
    int n = 9;
    int fact = 1;
    for(int i = 2; i <= n ; i++) {
        fact *= i;
    }
    System.out.println(fact);
}
```

Factorial – Solution 3

```
public class Factorial{

    public static void main(String[] args) {

        fact(4);
        fact(7);
        fact(9);
    }

    public static void fact(int n) {

        int fact = 1;
        for(int i = 2; i <= n ; i++) {
            fact *= i;
        }
        System.out.println(fact);
    }
}
```

Multiple parameters

- A method can accept multiple parameters. (separate by ,)
  - When calling it, you must pass values for each parameter.

- Declaration:

```
public static void methodName (type name, …, type name)
   {
      statement(s);
 }
```

- Call:

```
methodName (value, value, …, value);
```

Multiple params example

```
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
```

"Parameter Mystery" problem

```
public class ParameterMystery {
    public static void main(String[] args) {
        int x = 9;
        int y = 2;
        int z = 5;

        mystery(z, y, x);

        mystery(y, x, z);
    }

    public static void mystery(int x, int z, int y) {
        System.out.println(z + " and " + (y - x));
    }
}
```

9 2 5

2 9 5

Output:

2 and 4

"Parameter Mystery" problem

```
public class ParameterMystery {
    public static void main(String[] args) {
        int x = 9;
        int y = 2;
        int z = 5;

        mystery(z, y, x);

        mystery(y, x, z);
    }

    public static void mystery(int x, int z, int y) {
        System.out.println(z + " and " + (y - x));
    }
}
```

5 9 2

9 5 2

Output:

9 and 3

Strings as parameters

```
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);
    }
}
```

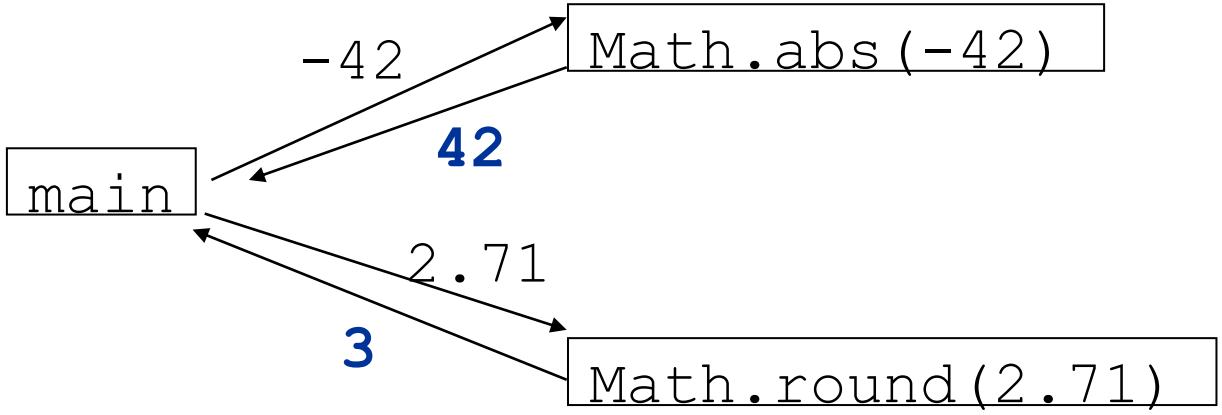
Output:

```
Welcome, Marty
Welcome, Bictolia
```

Return values

Return

- **return**: To send out a value as the result of a method.
  - The opposite of a parameter:
    - Parameters send information *in* from the caller to the method.
    - Return values send information *out* from a method to its caller.
      - A call to the method can be used as part of an expression.



Returning a value

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

- 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`

Factorial (Sum?)

```
public class Factorial{

    public static void main(String[] args) {
        int sum = fact(3) + fact(4);
        System.out.println(sum);
    }

    public static int fact(int n) {
        int factorial = 1;
        for(int i = 2; i <= n ; i++) {
            factorial *= i;
        }
        return factorial;
    }
}
```

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;
}
```

- 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.

```
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.

```
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;
}
```

Fixing the common error
(use the returned value in an expression)

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

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;
}
```

Example (CountFactors)

- Write a method **`countFactors`** that returns the number of factors of an integer.
  - **`countFactors(24)`** returns 8 because 1, 2, 3, 4, 6, 8, 12, and 24 are factors of 24.

- **Solution**:

```
// Returns how many factors the given number has.
public static int countFactors(int number) {
    int count = 0;
    for (int i = 1; i <= number; i++) {
        if (number % i == 0) {
            count++;  // i is a factor of number
        }
    }
    return count;
}
```

Returning *boolean* Example (isPrime)

```
public static boolean isPrime(int n) {
    int factors = 0;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            factors++;
        }
    }
    if (factors == 2) {
        return true;
    } else {
        return false;
    }
}
```

Can be replaced with

```
return factors == 2;
```

- Calls to methods returning `boolean` can be used as tests:

```
if (isPrime(57)) {
    ...
}
```