Redundancy Solutions

Parameters & Return Values

Redundant figures

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

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

*******

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

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

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

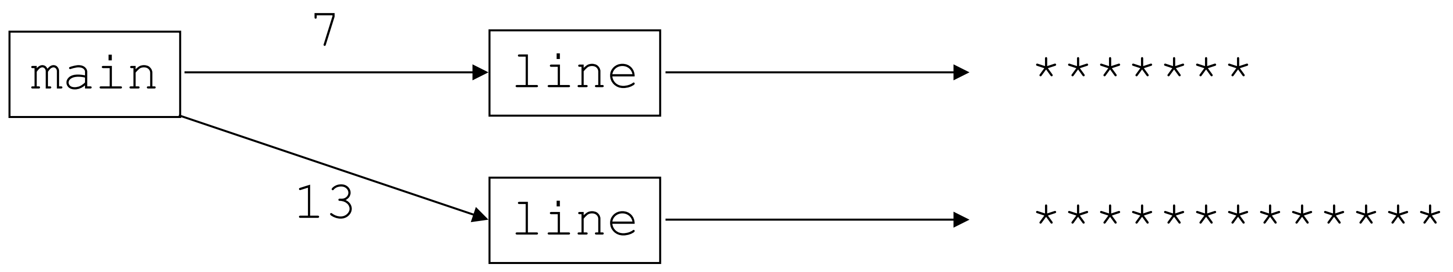
A redundant solution

```
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 in order to run

```
public static void name ( type name ) {
    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

name (**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);
}

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

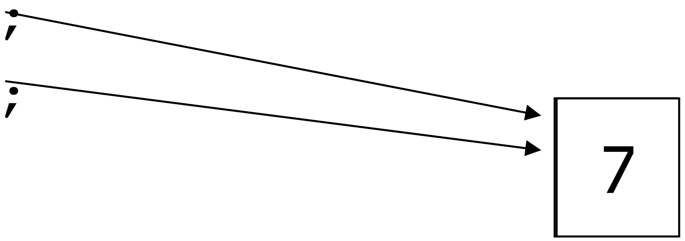
Output:

```
Just a salad...
Just a salad...
Just a salad...
```

How parameters are passed

- When the method is called:
  - The value is stored into the parameter variable.
  - The method's code executes using that value.

```
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
```

- Exercise: Change the `Stars` program to use a parameterized method for drawing lines of stars.

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

```
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
```

- Modify the `Stars` program to draw boxes with parameters.

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 strange(int x) {
    x = x + 1;
    System.out.println("1. x = " + x);
}

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

Output:

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

Return values

Java's `Math` class

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.ceil(`*value*`)`	rounds up
`Math.floor(`*value*`)`	rounds down
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1*`, `*value2*`)`	larger of two values
`Math.min(`*value1*`, `*value2*`)`	smaller of two values
`Math.pow(`*base*`, `*exp*`)`	*base* to the *exp* power
`Math.random()`	random `double` between 0 and 1
`Math.round(`*value*`)`	nearest whole number
`Math.sqrt(`*value*`)`	square root
`Math.sin(`*value*`)` `Math.cos(`*value*`)` `Math.tan(`*value*`)`	sine/cosine/tangent of an angle in radians
`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

Calling `Math` methods

`Math.`**methodName**`(`**parameters**`)`

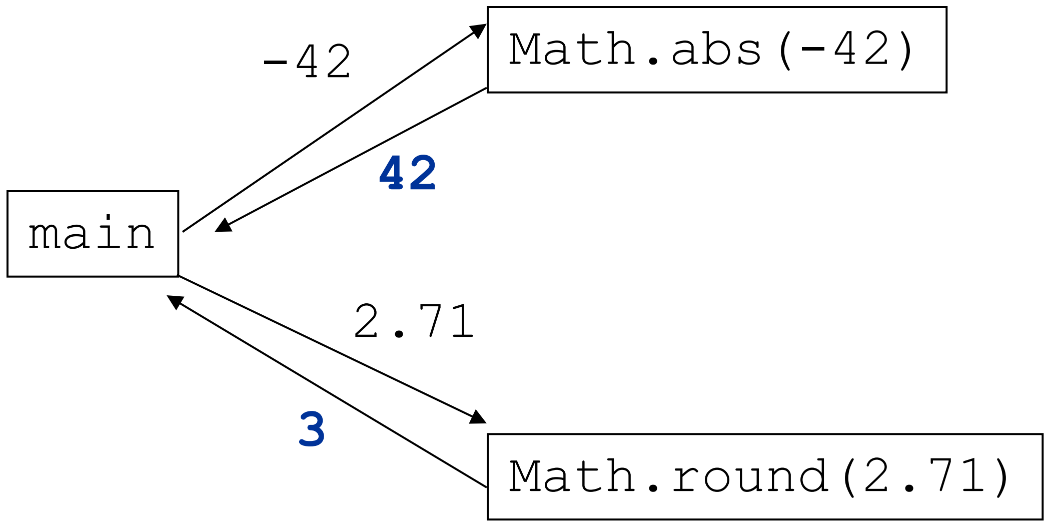
- Examples:

```
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
```

- 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.).

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.



`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.round(Math.PI) + Math.round(Math.E)`
  - `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?

Returning a value

```
public static type name(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`

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;
}
```