

Understanding Methods



Jim Wilson

MOBILE SOLUTIONS DEVELOPER & ARCHITECT

@hedgehogjim jwhh.com



Overview



Declaring and calling methods

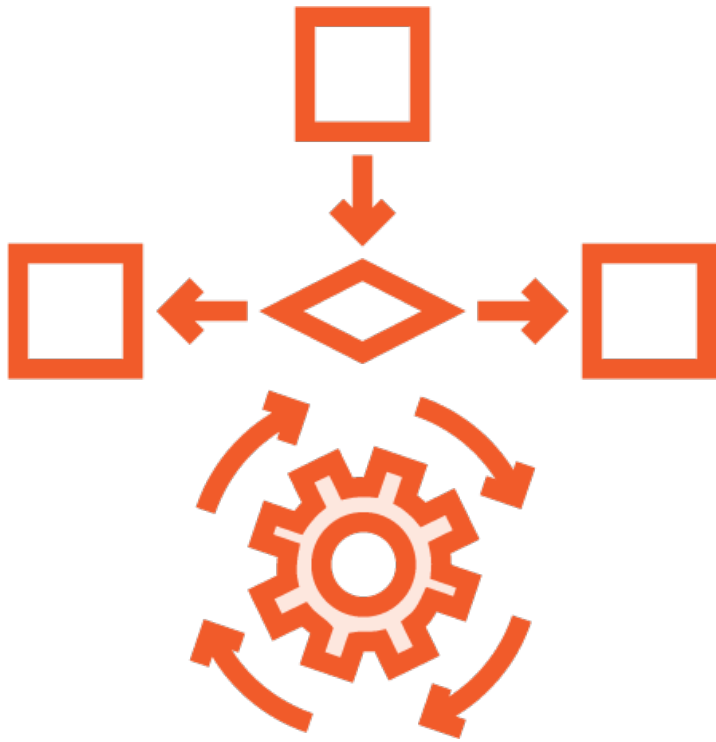
Parameters

Exiting a method

Returning a value

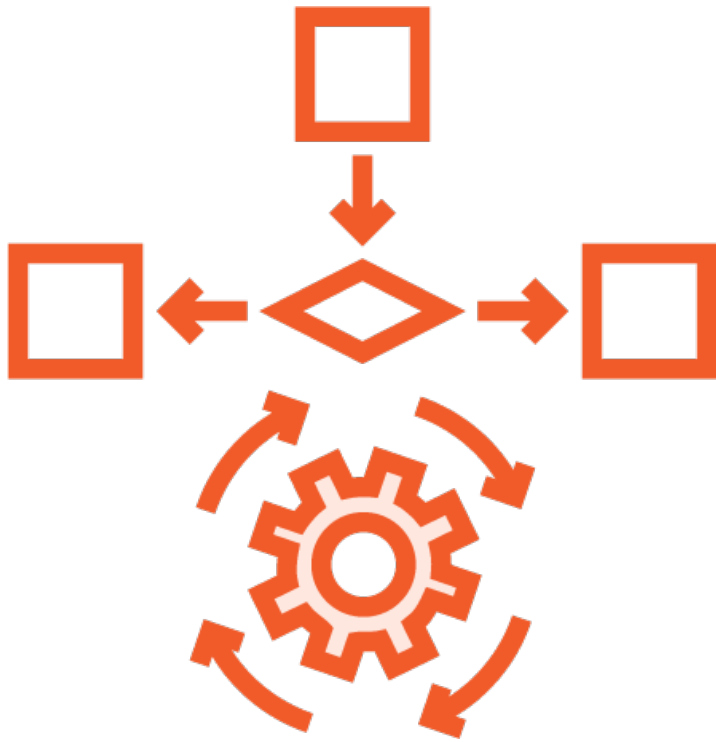
Program command-line arguments





Method

- Mechanism for organizing code
- Enables creation of reusable code blocks
- Can receive data
- Can return data



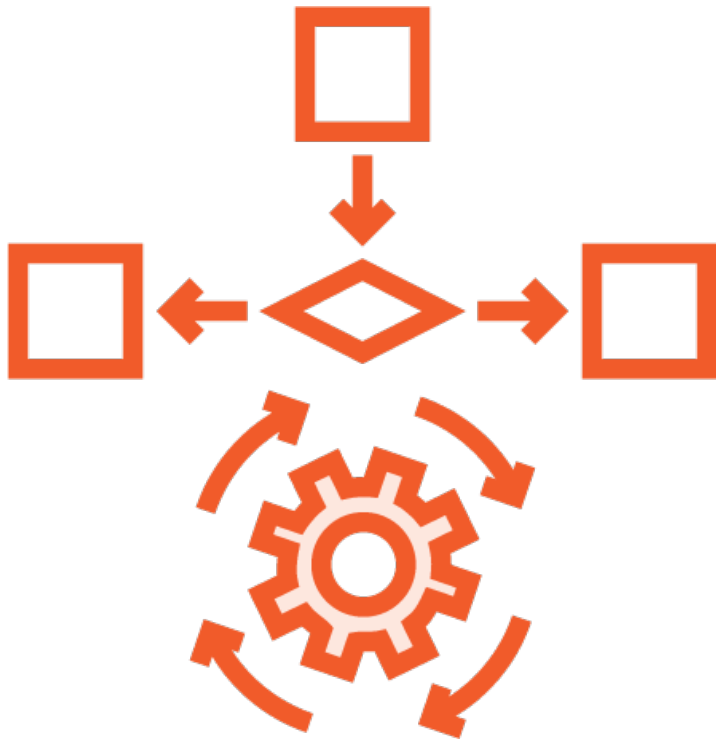
Name

- Same rules and conventions as variables

Typed parameter list

- Allow data values to be passed in
- Can be empty

name



Body

- Consists of zero or more statements
- Enclosed in brackets

Return type

- Indicates the type of data returned
- Use void when no value returned

```
name (typed-parameter-list)  
statements  
}
```



Using a Simple Method

Main.java

```
System.out.println("Before method call");
```

```
doSomething();
```

```
System.out.println("After method call");
```

doSomething

```
System.out.println("Inside method");
```

```
System.out.println("Still inside");
```

```
}
```

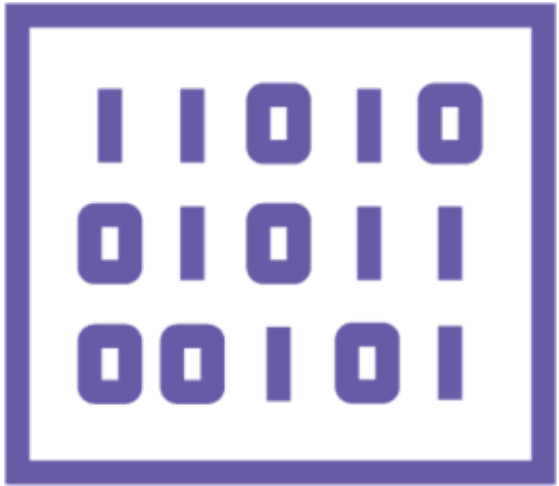
Before method call

Inside method

Still inside

After method call

Method Data



Variables

Scope limited to method where declared



Parameters

Enable passing data values to methods
Passed values matched to parameters
by position



Using Parameters

```
showSum(7.5, 1.4, 3);
```

```
static void showSum(float x, float y, int count) {
```

```
    float sum = x + y;
```

```
    for(int i = 0, i < count; i++)
```

```
        System.out.println(sum);
```

Prints
3 times

```
}
```

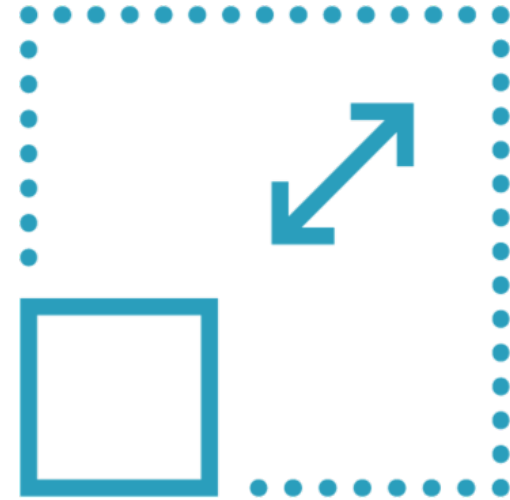


Parameter Passing



Parameters passed “by value”

Parameter receives a copy
of the original value



Method changes to parameter values

Visible within method
Not visible outside of method

Method Changes to Parameter Values

Main.java

```
int val1 = 10;
int val2 = 20;
swap(val1, val2);
System.out.println(val1) // 10
System.out.println(val2) // 20
```

val1

10

val2

20

Main.java

```
static void swap(int i, int j) {
    int k = i;
    i = j;
    j = k;
    // i:20 j:10
}
```

20

i

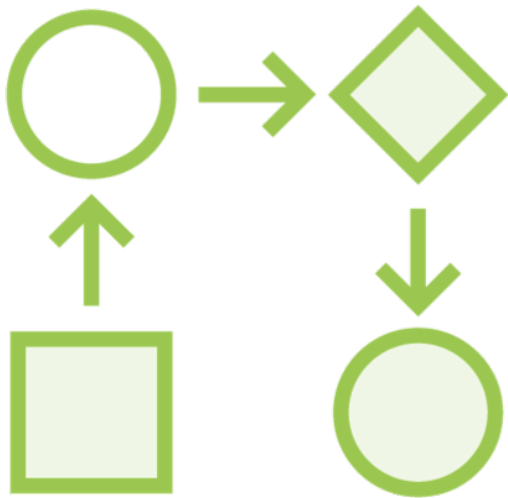
20

j

10

k

Reasons a Method Exits



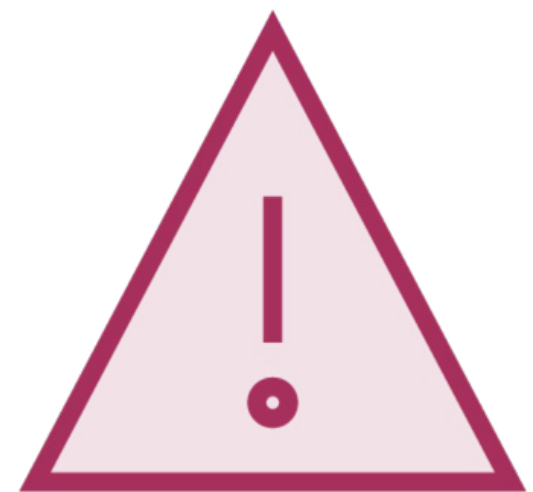
End of method

No more code in method



Return statement

Explicitly exit method



Error occurs

Method abruptly exits
Throws an exception

Exiting a Method

```
showSum(7.5, 1.4, 3);  
System.out.println("Back from showSum");
```



```
static void showSum(float x, float y, int count) {
```

```
    float sum = x + y;
```

```
    for(int i = 0, i < count; i++)
```

```
        System.out.println(sum);
```

```
    return;
```

```
}
```



Exiting a Method

```
showSum(7.5, 1.4, 0);  
System.out.println("Back from showSum");
```



```
static void showSum(float x, float y, int count) {
```

```
    float sum = x + y;
```

```
    for(int i = 0, i < count; i++)
```

```
        System.out.println(sum);
```

```
    return;
```

```
}
```



Exiting a Method

```
showSum(7.5, 1.4, 0);
```

```
System.out.println("Back from showSum");
```

```
static void showSum(float x, float y, int count) {
```

```
    if (count < 1)
```

```
        return;
```

```
    float sum = x + y;
```

```
    for(int i = 0, i < count; i++)
```

```
        System.out.println(sum);
```

```
    return;
```

```
}
```





A method can return a single value

- Value returned with return statement

Method return type

- Can be a primitive type
- Can be a more complex type such as an array



Returning a Value

```
calculateInterest(100d, 0.05d, 10);
```

```
System.out.println(result); // 50.0
```

```
static double calculateInterest(double amt, double rate, int years) {  
    double interest = amt * rate * years;  
    return  
}
```



Returning a Value

```
double result = calculateInterest(100d, 0.05d, 10);
```

```
System.out.println(result); // 50.0
```

```
static double calculateInterest(double amt, double rate, int years) {
```

```
    return amt * rate * years;
```

```
}
```



Returning an Array

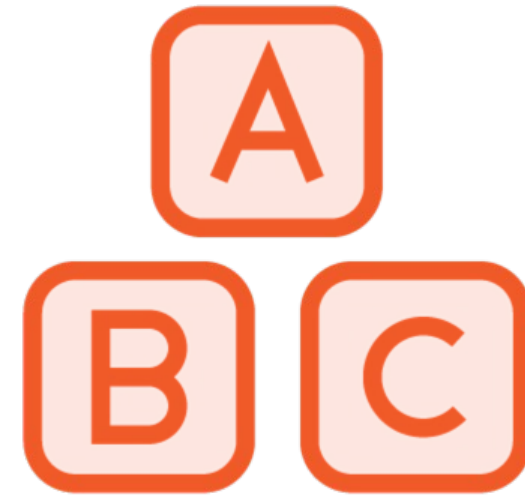
```
static double[] produceInterestHistory(double amt, double rate, int years)
{
    double[] accumulatedInterest = new double[years];
    for(int yearIndex = 0; yearIndex < years; yearIndex++) {
        int year = yearIndex + 1;
        accumulatedInterest[yearIndex] = calculateInterest(amt, rate, year);
    }
    return accumulatedInterest;
}
```



Command-line Arguments



Application main method
Serves as program entry point



Receives an array as parameter
Contains command-line arguments



Command-line Arguments

```
java com.pluralsight.example.Main
```

Main.java

```
public static void main(String[] args) {  
    if(args.length < 1)  
        System.out.println("No args provided");  
    else  
        for(String arg : args)  
            System.out.println(arg);  
}
```

Command-line Arguments

```
java com.pluralsight.example.Main Hello "Mary Ann"
```

Main.java

```
public static void main(String[] args) {  
    if(args.length < 1)  
        System.out.println("No args provided");  
    else  
        for(String arg : args)  
            System.out.println(arg);  
}
```

Hello

Mary Ann

Summary



Method

- Mechanism for organizing code
- Enables creation of reusable code

Variables

- Scope limited to method



Summary



Parameters

- Enable passing values to methods
- Parameters passed positionally

Parameters passed “by value”

- A copy of the original value is passed
- Changes made to parameter values not visible outside of method



Summary



Method return value

- Value specified with return statement
- Can be primitive type
- Can be complex type such as an array

Command-line arguments

- Received by app's main method
- Received as String array

