CHAPTER 5

# METHODS



Slides by Donald W. Smith
TechNeTrain.com

Final Draft
10/29/2011

# 5.1 Methods as Black Boxes



- A method is a sequence of instructions with a name
  - You declare a method by defining a named block of code

```
public static void main(String[] args)
{
  double result = Math.pow(2, 3);
  . . .
}
```

  - You call a method in order to execute its instructions

A method packages a computation consisting of multiple steps into a form that can be easily understood and reused.

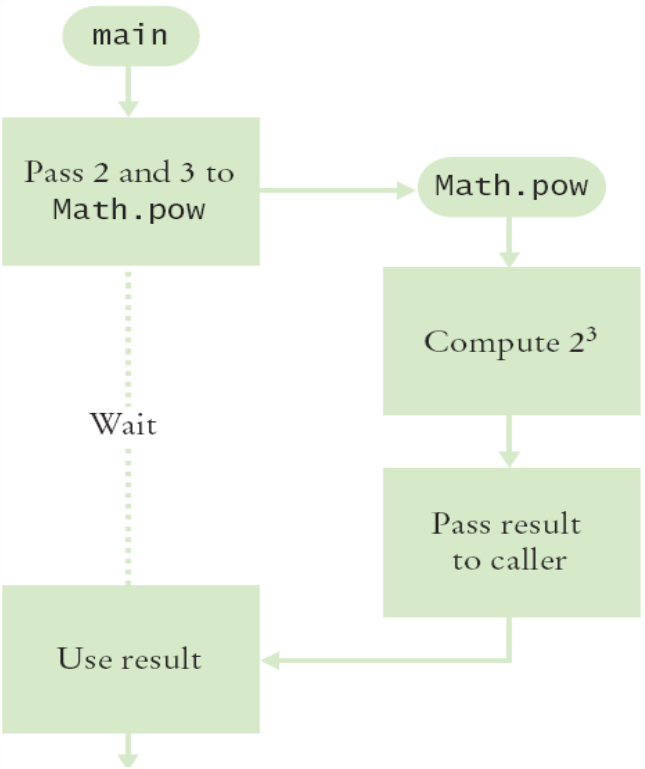
# Example Static Methods



- Some example static methods:
  - `Math.pow()`
  - `String.length()`
  - `Character.isDigit()`
  - `Scanner.nextInt()`
  - `main()`

# Flowchart of Calling a Method





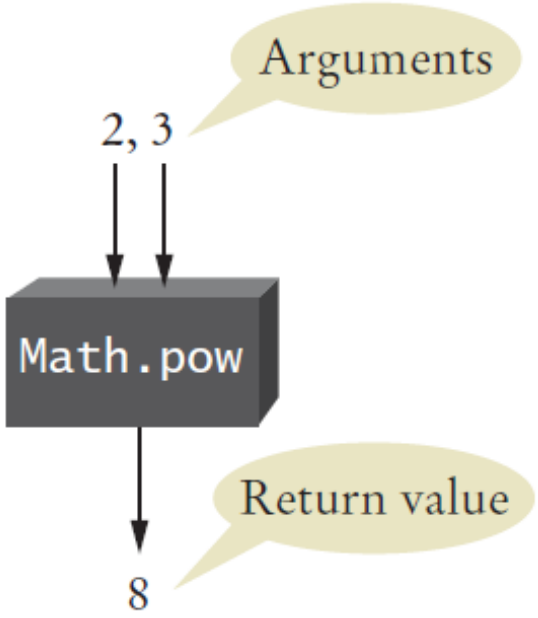

```
public static void main(String[] args)
{
  double result = Math.pow(2, 3);
  . . .
}
```

- One method 'calls' another
  - main calls `Math.pow()`
  - Passes two arguments
    - 2 and 3
  - `Math.pow` starts
    - Uses variables (2, 3)
    - Does its job
    - Returns the answer
  - main uses result

# Arguments and Return Values



```
public static void main(String[] args)
{
  double result = Math.pow(2,3);
  . . .
}
```




- `main` 'passes' two arguments (2 and 3) to `Math.pow`
- `Math.pow` calculates and returns a value of 8 to `main`
- `main` stores the return value to variable 'result'

# Example Method



- The method body is surrounded by curly braces { }

```
public static double cubeVolume(double sideLength)
{
  double volume = sideLength * sideLength * sideLength;
  return volume;
}
```

# Calling Methods



```
public static void main(String[] args)
{
   double result1 = cubeVolume(2);
   double result2 = cubeVolume(10);
   System.out.println("A cube of side length 2 has volume
         "  + result1);
   System.out.println("A cube of side length 10 has volume
         " + result2);
}
```

# Method Declaration



Type of return value

Type of parameter variable

Name of method

Name of parameter variable

```
public static double cubeVolume(double sideLength)
{
   double volume = sideLength * sideLength * sideLength;
   return volume;
}
```

Method body, executed when method is called.

`return` statement exits method and returns result.

# Method Comments



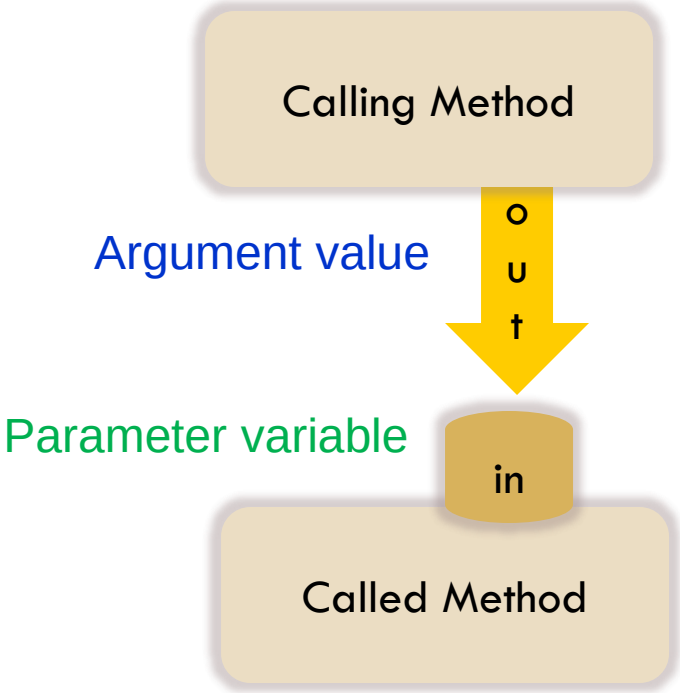
- Write a Javadoc comment above each method
- Start with `/**`
  - Note the purpose of the method
  - `@param` Describe each parameter variable
  - `@return` Describe the return value
- End with `*/`

```
/**
  Computes the volume of a cube.
  @param sideLength the side length of the cube
  @return the volume
*/
public static double cubeVolume(double sideLength)
```

# 5.3 Parameter Passing



- Parameter variables receive the argument values supplied in the method call
  - They both must be the same type
- The argument value may be:
  - The contents of a variable
  - A 'literal' value (2)
  - aka. 'actual parameter' or argument
- The parameter variable is:
  - **Declared** in the called method
  - Initialized with the value of the argument value
  - Used as a variable inside the called method
  - aka. 'formal parameter'



# Parameter Passing Steps



```
public static void main(String[] args)
{
  double result1 = cubeVolume(2);
  . . .
}
```

result1 = 8

```
public static double cubeVolume(double result1)
{
  double volume = result1*result1*result1;Length *
   sideLength;
  return volume;
}
```

sideLength = 2

volume = 8

# Common Error 5.1



- Trying to Modify Arguments
  - A copy of the argument values is passed
  - Called method (addTax) can modify local copy (price)
    - But not original

    in calling method
    - total

```
public static void main(String[] args)
{
  double price = 10;
  addTax(price, 7.5);
}
```

total

10.0

copy

```
public static int addTax(double price, double rate)
{
  double tax = price * rate / 100;
  price = price + tax; // Has no effect outside the method
  return tax;
}
```

price

10.75

# 5.4 Return Values



- Methods can (optionally) return one value
  - Declare a return type in the method declaration
  - Add a return statement that returns a value
    - A return statement does two things:
      1) Immediately terminates the method
      2) Passes the return value back to the calling method

return type

```
public static double cubeVolume (double sideLength)
{
  double volume = sideLength * sideLength * sideLength;

}
```

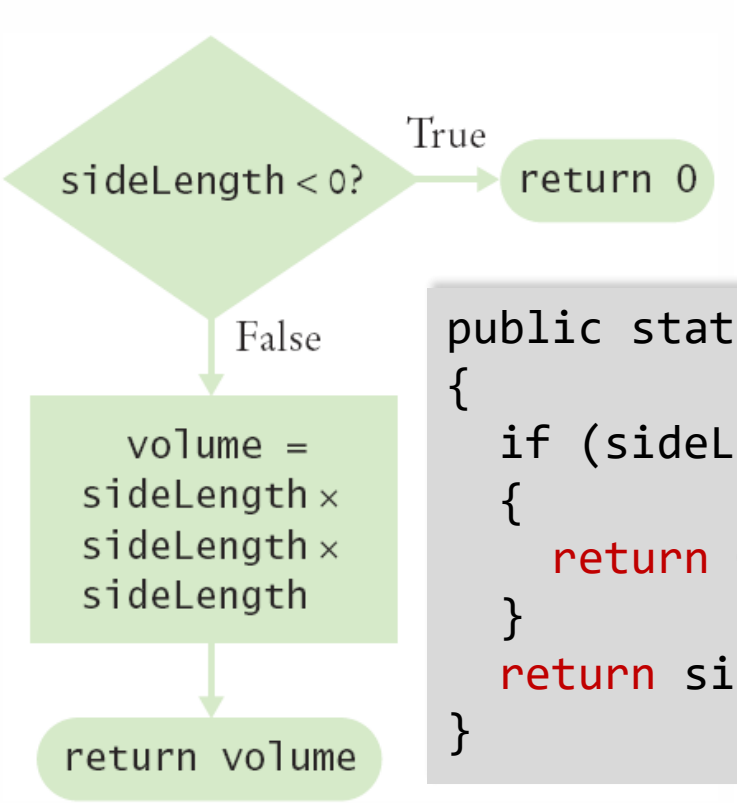
return statement

- The return value may be a value, a variable or a calculation
  - Type must match return type

# Multiple `return` Statements



- A method can use multiple `return` statements
  - But every branch must have a `return` statement




```
public static double cubeVolume(double sideLength)
{
  if (sideLength < 0)
  {
    return 0;
  }
  return sideLength * sideLength * sideLength;
}
```

# Common Error 5.2



- Missing `return` Statement
  - Make sure all conditions are handled
  - In this case, `x` could be equal to 0
    - No `return` statement for this condition
    - The compiler will complain if any branch has no `return` statement

```
public static int sign(double x)
{
  if (x < 0) { return -1; }
  if (x > 0) { return 1; }
  return 0;// Error: missing return value if x equals 0
}
```

# 5.5 Methods without Return Values



- Methods are not required to return a value
  - The return type of void means nothing is returned
  - No `return` statement is required
  - The method can generate output though!

```
...
boxString("Hello");
...
```

```
-------
!Hello!
-------
```

```
public static void boxString(String str)
{
  int n = str.length();
  for (int i = 0; i < n + 2; i++)
    { System.out.print("-"); }
  System.out.println();
  System.out.println("!" + str + "!");
  for (int i = 0; i < n + 2; i++)
    { System.out.print("-"); }
  System.out.println();
}
```

# Using return Without a Value



- You can use the return statement without a value
  - In methods with void return type
  - The method will terminate immediately!

```
public static void boxString(String str)
{
  int n = str.length();
  if (n == 0)
  {
    return; // Return immediately
  }
  for (int i = 0; i < n + 2; i++) { System.out.print("-"); }
  System.out.println();
  System.out.println("!" + str + "!");
  for (int i = 0; i < n + 2; i++) { System.out.print("-"); }
  System.out.println();
}
```

# Write a ‘Parameterized’ Method



```
/**
  Prompts a user to enter a value in a given range until the user
  provides a valid input.
  @param low the low end of the range
  @param high the high end of the range
  @return the value provided by the user
*/
public static int readValueBetween(int low, int high)
{
  int input;
  do
  {
    System.out.print("Enter between " + low + " and " + high + ": ");
    Scanner in = new Scanner(System.in);
    input = in.nextInt();
  }
  while (input < low || input > high);
  return input;
}
```

# Variable Scope

# 5.8 Variable Scope



The scope of a variable is the part of the program in which it is visible.

- Variables can be declared:
  - 1- Inside a method
    - Known as 'local variables'
    - Only available inside this method
    - Parameter variables are like local variables
  - 2 - Inside a block of code { }
    - Sometimes called 'block scope'
    - If declared inside block { ends at end of block }
  - 3 - Outside of a method
    - Sometimes called 'global scope'
    - Can be used (and changed) by code in any method
- How do you choose?

# Examples of Scope



- `sum` is a local variable in `main`
- `square` is only visible inside the for loop block
- `i` is only visible inside the for loop

```
public static void main(String[] args)
{
  int sum = 0;
  for (int i = 1; i <= 10; i++)
  {
    int square = i * i;
    sum = sum + square;
  }
  System.out.println(sum);
}
```

sum

i

square

# Local Variables of Methods



- Variables declared inside one method are not visible to other methods
  - `sideLength` is local to `main`
  - Using it outside `main` will cause a compiler error

```
public static void main(String[] args)
{
  double sideLength = 10;
  int result = cubeVolume();
  System.out.println(result);
}

public static double cubeVolume()
{
  return sideLength * sideLength * sideLength; // ERROR!!
}
```

# Re-using names for local variables



- Variables declared inside one method are not visible to other methods
  - `result` is local to square and `result` is local to `main`
  - They are two different variables and do not overlap

```
public static int square(int n)
{
  int result = n * n;
  return result;
}

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

result

result

# Re-using names for block variables



- Variables declared inside one block are not visible to other methods
  - i is inside the first **for** block and i is inside the second
  - They are two different variables and do not overlap

```
public static void main(String[] args)
{
  int sum = 0;
  for (int i = 1; i <= 10; i++)
  {
    sum = sum + i;
  }
  for (int j = 1; j<= 10; j++)
  {
    sum = sum + i * i;
  }
  System.out.println(sum);
}
```

i

i

# Overlapping Scope



- Variables (including parameter variables) must have unique names within their scope
  - n has local scope and n is in a block inside that scope
  - The compiler will complain when the block scope n is declared

```
public static int sumOfSquares(int n)
{
  int sum = 0;
  for (int i = 1; i <= n; i++)
  {
    int n = i * i; // ERROR
    sum = sum + n;
  }
  return sum;
}
```

block scope n

n

# Global and Local Overlapping



- Global and Local (method) variables can overlap
  - The local `same` will be used when it is in scope
  - No access to global `same` when local `same` is in scope

```
public class Scoper
{
  public static int same;   // 'global'
  public static void main(String[] args)
  {
    int same = 0;     // local
    for (int i = 1; i <= 10; i++)
    {
      int square = i * i;
      same = same + square;
    }
    System.out.println(same);
  }
}
```

same

same

Variables in different scopes with the same name will compile, but it is not a good idea