APROG – Algoritmia e Programação

Emanuel Cunha Silva
ecs@isep.ipp.pt

Bibliography:
Big Java Late Objects
Book by Cay S. Horstmann

Chapter Goals

- To be able to implement methods
- To become familiar with the concept of parameter passing
- To develop strategies for decomposing complex tasks into simpler ones
- To be able to determine the scope of a variable

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 square(int x)
{
  int result = x * x;
  return result;
}
```

  - You call a method in order to execute its instructions

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

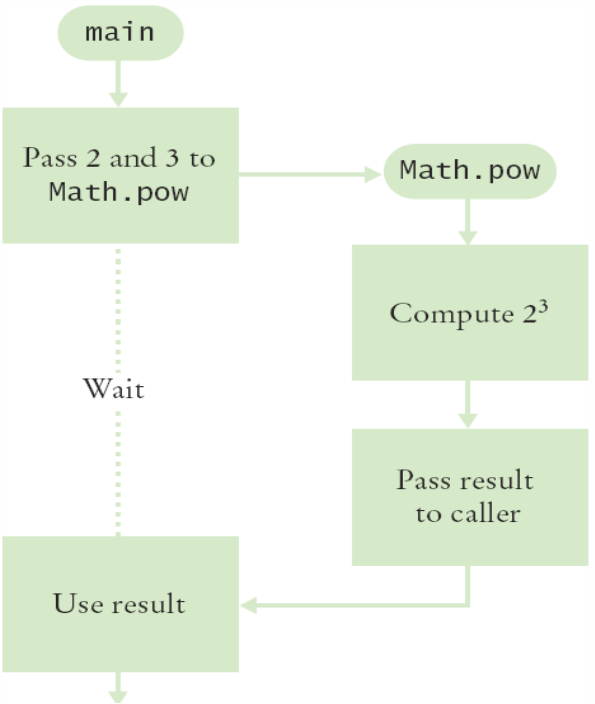
What Is a Method?

- Some methods you have already used are:
  - `Math.pow()`
  - `String.length()`
  - `Character.isDigit()`
  - `Scanner.nextInt()`
  - `main()`

- Methods may have:
  - A capitalized name and a dot (.) before them
  - A method name
    - Follow the same rules as variable names, camelHump style
  - ( ) - a set of parenthesis at the end
    - A place to provide the method input information

Flowchart of Calling a Method

- 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

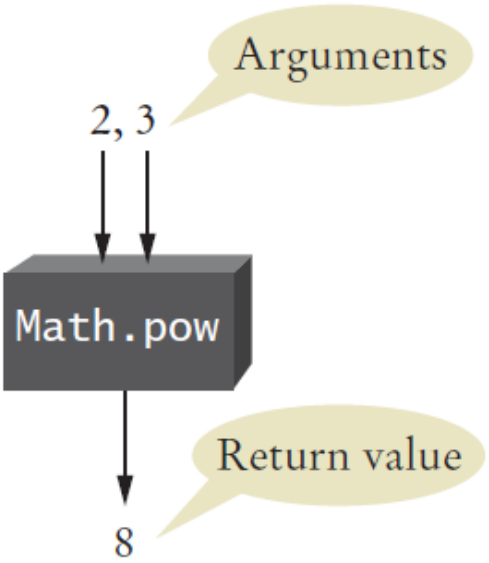



```
public static void main(String[] args)
{
  double result = Math.pow(2, 3);
  . . .
}
```

Arguments and Return Values

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

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



Black Box Analogy

- A thermostat is a ‘black box’
  - Set a desired temperature
  - Turns on heater/AC as required
  - You don’t have to know how it really works!
    - How does it know the current temp?
    - What signals/commands does it send to the heater or A/C?

- Use methods like ‘black boxes’
  - Pass the method what it needs to do its job
  - Receive the answer

Self Check

Consider the method call `Math.pow(3, 2)`. What are the arguments and return values?

Answer: The arguments are 3 and 2. The return value is 9.

What is the return value of the method call `Math.pow(Math.pow(2, 2), 2)`?

Answer: The inner call to `Math.pow` returns $2^2 = 4$. Therefore, the outer call returns $4^2 = 16$.

The `Math.ceil` method in the Java standard library is described as follows: The method receives a single argument *a* of type `double` and returns the smallest `double` value ≥ *a* that is an integer. What is the return value of `Math.ceil(2.3)`?

Answer: 3.0

Implementing Methods

- A method to calculate the volume of a cube
  - What does it need to do its job?
  - What does it answer with?

- When declaring a method, you provide a name for the method, a variable for each argument, and a type for the result
  - Pick a name for the method (`cubeVolume`).
  - Declare a variable for each incoming argument
  - (`double sideLength`) (called parameter variables)
  - Specify the type of the return value (`double`)
  - Add modifiers such as `public static`

```
public static double cubeVolume(double sideLength)
```

Inside the Box

- Then write the body of the method
  - The body is surrounded by curly braces `{ }`
  - The body contains the variable declarations and statements that are executed when the method is called
  - It will also return the calculated answer

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

Back from the Box

- The values returned from `cubeVolume` are stored in local variables inside `main`
- The results are then printed out

```
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

Syntax

```
public static returnType methodName(parameterType parameterName, . . . )
{
   method body
}
```

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.

Cubes.java

```java
/**
   This program computes the volumes of two cubes.
*/
public class Cubes
{
   public static void main(String[] args)
   {
      double result1 = cubeVolume(2);
      double result2 = cubeVolume(10);
      System.out.println("A cube with side length 2 has volume " + result1);
      System.out.println("A cube with side length 10 has volume " + result2);
   }

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

Program Run

```
A cube with side length 2 has volume 8
A cube with side length 10 has volume 1000
```

Self Check

What is the value of `cubeVolume(3)`?

Answer: 27

What is the value of `cubeVolume(cubeVolume(2))`?

Answer: 8 × 8 × 8 = 512

Provide an alternate implementation of the body of the `cubeVolume` method by calling the `Math.pow` method.

Answer:

```
double volume = Math.pow(sideLength, 3);
return volume;
```

Self Check

Declare a method `squareArea` that computes the area of a square of a given side length.

Answer:

```
public static double squareArea(double sideLength){
   double area = sideLength * sideLength;
   return area;
}
```

Consider this method:

```
public static int mystery(int x, int y){
   double result = (x + y) / (y - x);
   return result;
}
```

What is the result of the call mystery(2, 3)?

Answer: `(2 + 3) / (3 - 2) = 5`

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 `*/`

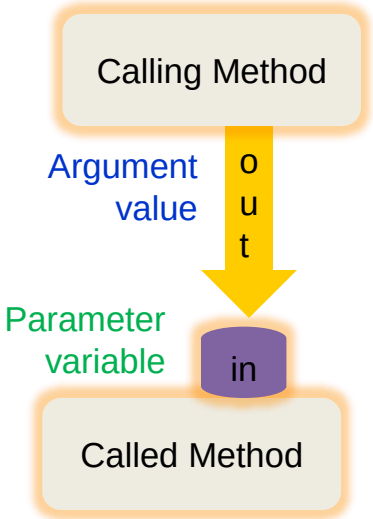
```
/**
  Computes the volume of a cube.
  @param sideLength the side length of the cube
  @return the volume
*/
public static double cubeVolume(double sideLength)
```

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 sideLength)
{
  double volume = sideLength * sideLength * sideLength;
  return volume;
}
```

sideLength = 2

volume = 8

Self Check

What does this program print?

```
public static double mystery(int x, int y)
{
   double z = x + y;
   z = z / 2.0;
   return z;
}
public static void main(String[] args)
{
   int a = 5;
   int b = 7;
   System.out.println(mystery(a, b));
}
```

Answer: When the `mystery` method is called, `x` is set to 5, `y` is set to 7, and `z` becomes 12.0. Then `z` is changed to 6.0, and that value is returned and printed.

Self Check

What does this program print?

```
public static int mystery(int x)
{
   int y = x * x;
   return y;
}
public static void main(String[] args)
{
   int a = 4;
   System.out.println(mystery(a + 1));
}
```

Answer: When the method is called, `x` is set to 5. Then `y` is set to 25, and that value is returned and printed.

Self Check

What does this program print?

```
public static int mystery(int n)
{
   n++;
   n++;
   return n;
}
public static void main(String[] args)
{
   int a = 5;
   System.out.println(mystery(a));
}
```

Answer: When the method is called, `n` is set to 5. Then `n` is incremented twice, setting it to 7. That value is returned and printed.

Common Error

- 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 total = 10;
  addTax(total,        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

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:
      - Immediately terminates the method
      - Passes the return value back to the calling method

return type

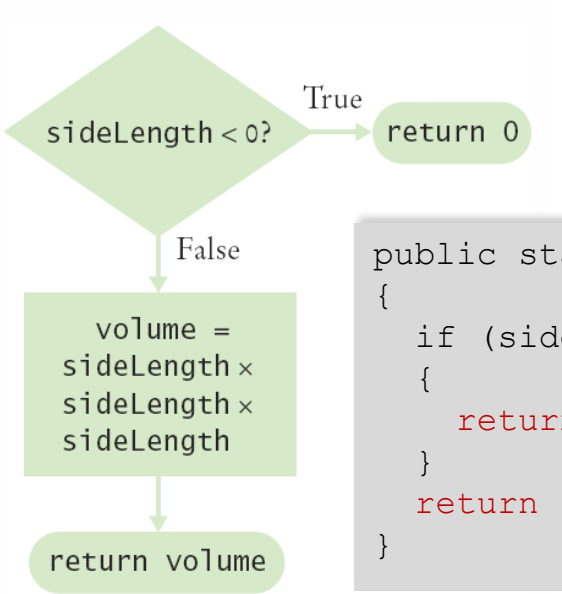
```
public static double cubeVolume (double sideLength)
{
  double volume = sideLength * sideLength * sideLength;
  return volume;
}
```

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;
}
```

Self Check

Suppose we change the body of the `cubeVolume` method from:

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

to:

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

How does this method differ from the one described in this section?

Answer: It acts the same way: If `sideLength` is 0, it returns 0 directly instead of computing 0 × 0 × 0.

Self Check

What does this method do?

```
public static boolean mystery(int n){
   if (n % 2 == 0) {
      return true;
   }
   else {
      return false;
   }
}
```

Answer: It returns `true` if `n` is even; `false` if `n` is odd.

Implement the above `mystery` method with a single `return` statement.

Answer:

```
public static boolean mystery(int n){
   return n % 2 == 0;
}
```

Common Error

- 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; }
  // Error: missing return value if x equals 0
}
```

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, but a `return` without a value can be coded
  - 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();
}
```

Self Check

How do you generate the following printout, using the `boxString` method?

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

Answer:

```
boxString("Hello");
boxString("World");
```

What is wrong with the following statement?

```
System.out.print(boxString("Hello"));
```

Answer: The `boxString` method does not return a value. Therefore, you cannot use it in a call to the `print` method.

Implement a method `shout` that prints a line consisting of a string followed by three exclamation marks. For example, `shout("Hello")` should print `Hello!!!`. The method should not return a value.

Answer:

```
public static void shout(String message){
   System.out.println(message + "!!!");
}
```

Self Check

How would you modify the `boxString` method to leave a space around the string that is being boxed, like this:

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

Answer:

```
public static void boxString(String contents){
   int n = contents.length();
   for (int i = 0; i < n + 4; i++){
      System.out.print("-");
   }
   System.out.println();
   System.out.println("! " + contents + " !");
   for (int i = 0; i < n + 4; i++){
      System.out.print("-");
   }
   System.out.println()
}
```

Self Check

The `boxString` method contains the code for printing a line of - characters twice. Place that code into a separate method `printLine`, and use that method to simplify `boxString`. What is the code of both methods?

Answer:

```
public static void printLine(int count){
   for (int i = 0; i < count; i++){
      System.out.print("-");
   }
   System.out.println();
}

public static void boxString(String contents){
   int n = contents.length();
   printLine(n + 2);
   System.out.println("!" + contents + "!");
   printLine(n + 2);
}
```

Problem Solving: Reusable Methods

- Find Repetitive Code
  - May have different values but same logic

1 - 12

```
int hours;
do{
  System.out.print("Enter a value between 1 and 12: ");
  hours = in.nextInt();
}while (hours < 1 || hours > 12);
```

0 - 59

```
int minutes;
do{
  System.out.print("Enter a value between 0 and 59: ");
  minutes = in.nextInt();
}while (minutes < 0 || minutes > 59);
```

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)
{
  Scanner in = new Scanner(System.in);  int input;
  do{
    System.out.print("Enter between " + low + " and " + high + ": ");
    input = in.nextInt();
  }while (input < low || input > high);
  return input;
}
```

1, 12

0, 59

8

Self Check

Consider this method that prints a page number on the left or right side of a page:

```
if (page % 2 == 0) {
   System.out.println(page);
}
else {
   System.out.println("                                " + page);
}
```

Introduce a method with return type `boolean` to make the condition in the `if` statement easier to understand.

Answer:

```
if (isEven(page)) . . .
```

where the method is defined as follows:

```
public static boolean isEven(int n){
   return n % 2 == 0;
}
```

Self Check

Consider the following method that computes compound interest for an account with an initial balance of $10,000 and an interest rate of 5 percent:

```
public static double balance(int years) {
   return 10000 * Math.pow(1.05, years);
}
```

How can you make this method more reusable?

Answer: Add parameter variables so you can pass the initial balance and interest rate to the method:

```
public static double balance(double initialBalance, double rate, int years){
   return initialBalance * pow(1 + rate / 100, years);
}
```

Self Check

The comment explains what the following loop does. Use a method instead.

```
// Counts the number of spaces
int spaces = 0;
for (int i = 0; i < input.length(); i++){
   if (input.charAt(i) == ' ') {
      spaces++;
   }
}
```

Answer:

```
int spaces = countSpaces(input);
```

where the method is defined as follows:

```
/**
   Gets the number of spaces in a string.
   @param str any string
   @return the number of spaces in str
*/
public static int countSpaces(String str){
   int count = 0;
   for (int i = 0; i < str.length(); i++){
      if (str.charAt(i) == ' ){
         count++;
      }
   }
   return count;
}
```

Self Check

How can you generalize a method so that it can count any character? Why would you want to do this?

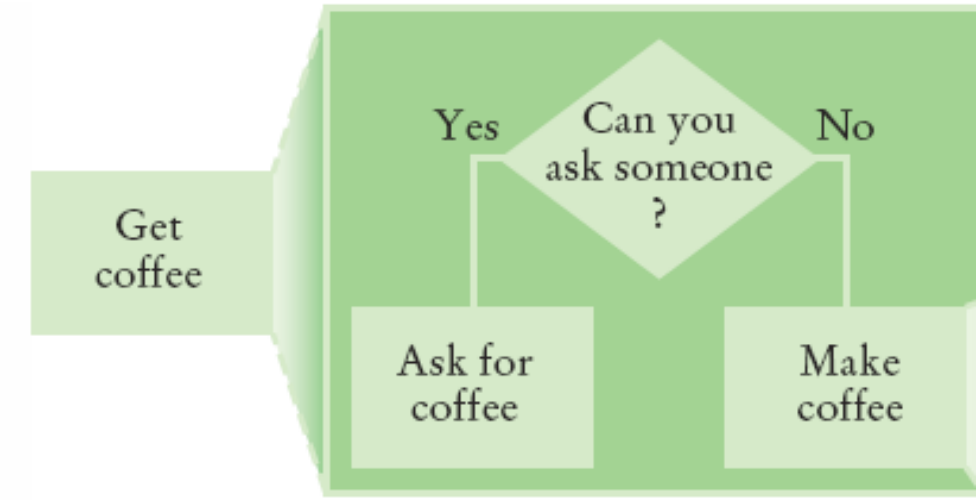
Answer: It is very easy to replace the space with any character.

```
/**
   Counts the instances of a given character in a string.
   @param str any string
   @param ch a character whose occurrences should be counted
   @return the number of times that ch occurs in str
*/
public static int count(String str, char ch){
   int count = 0;
   for (int i = 0; i < str.length(); i++){
      if (str.charAt(i) == ch) {
         count++;
      }
   }
   return count;
}
```

This is useful if you want to count other characters. For example, `count(input, ",")` counts the commas in the input.

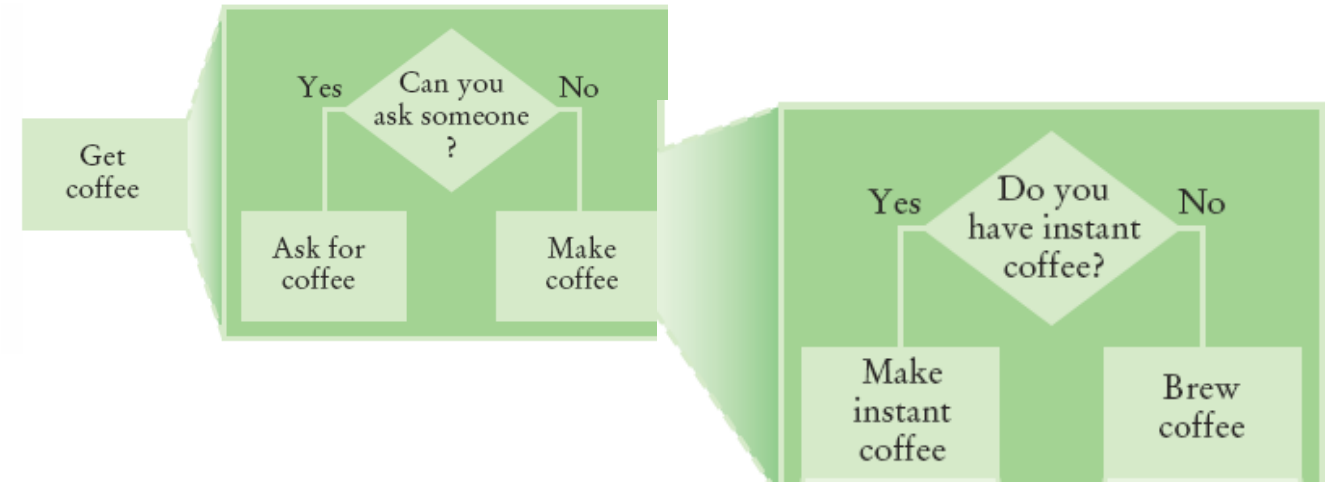
Problem Solving

- Stepwise Refinement
  - To solve a difficult task, break it down into simpler tasks
  - Then keep breaking down the simpler tasks into even simpler ones, until you are left with tasks that you know how to solve



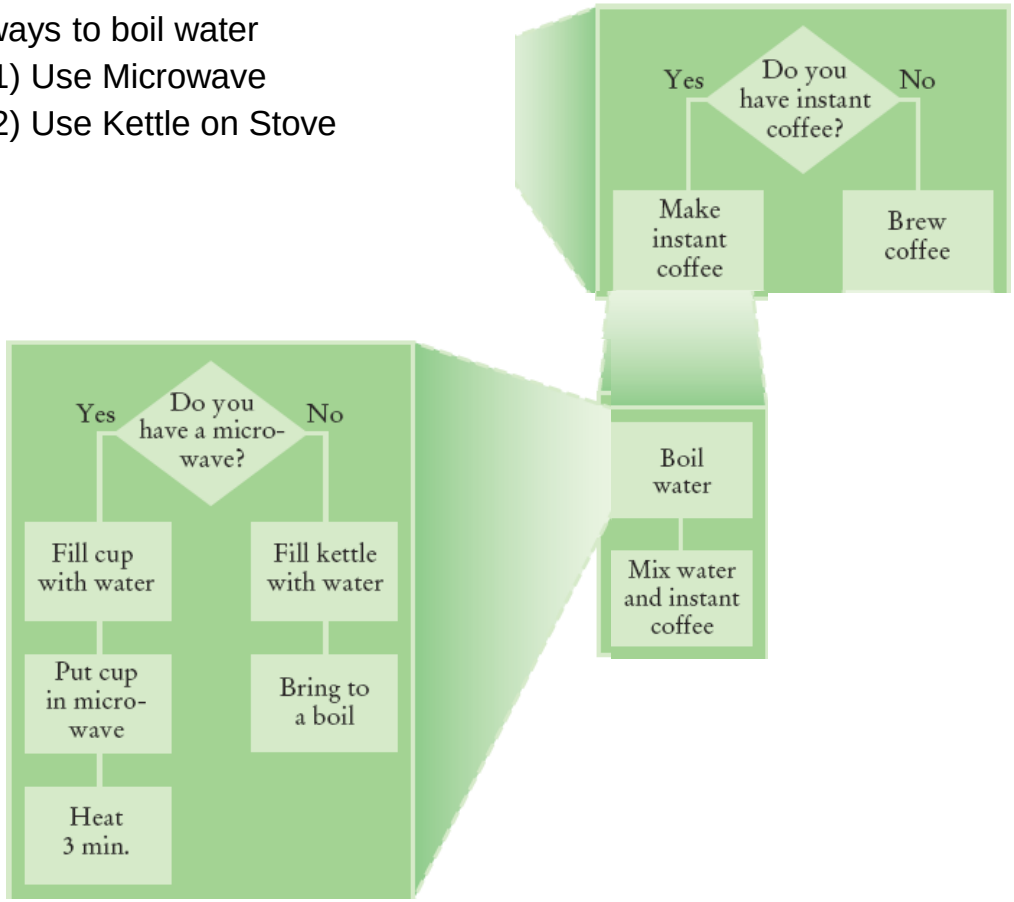
Get Coffee

- If you must make coffee, there are two ways:
  - Make Instant Coffee
  - Brew Coffee



Instant Coffee

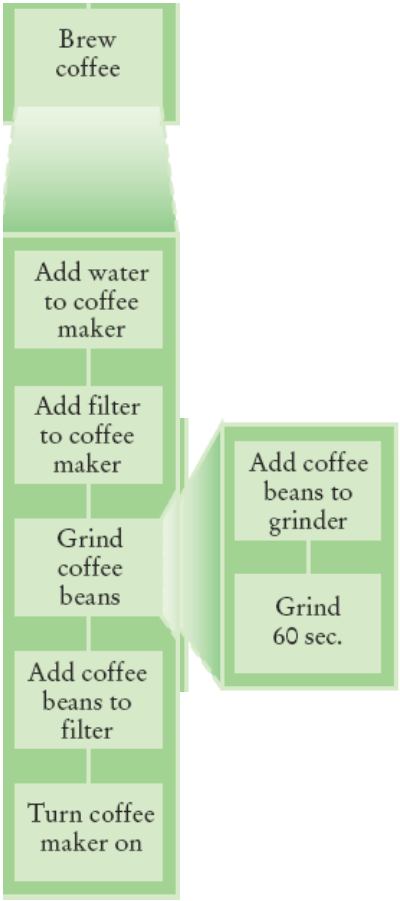
- Two ways to boil water
  1) Use Microwave
  2) Use Kettle on Stove



Brew Coffee

- Assumes coffee maker
  - Add water
  - Add filter
  - Grind Coffee
    - Add beans to grinder
    - Grind 60 seconds
  - Fill filter with ground coffee
  - Turn coffee maker on

- Steps are easily done



Variable Scope

- Variables can be declared:
  - Inside a method
    - Known as ‘local variables’
    - Only available inside this method
    - Parameter variables are like local variables
  - Inside a block of code { }
    - Sometimes called ‘block scope’
    - If declared inside block { ends at end of block }
  - 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

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

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 i = 1; i <= 10; i++)
  {
    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;
}
```

Local n

block scope 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

Self Check

Consider this sample program, then answer the questions below.

```
public class Sample
{
   public static void main(String[] args)
   {
      int x = 4;
      x = mystery(x + 1);
      System.out.println(s);
   }

   public static int mystery(int x)
   {
      int s = 0;
      for (int i = 0; i < x; x++)
      {
         int x = i + 1;
         s = s + x;
      }
      return s;
   }
}
```

Self Check

Which lines are in the scope of the variable `i` declared in line 13?

Answer: Lines 14-17.

Which lines are in the scope of the parameter variable `x` declared in line 10?

Answer: Lines 11-19.

The program declares two local variables with the same name whose scopes don't overlap. What are they?

Answer: The variables x defined in lines 5 and 15.

There is a scope error in the mystery method. How do you fix it?

Answer: Rename the local variable `x` that is declared in line 15, or rename the parameter variable `x` that is declared in line 10.

There is a scope error in the `main` method. What is it, and how do you fix it?

Answer: The `main` method accesses the local variable `s` of the `mystery` method. Assuming that the `main` method intended to print the last value of `s` before the method returned, it should simply print the return value that is stored in its local variable `x`.