

METHODS



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
- To learn how to think recursively (optional)

In this chapter, you will learn how to design and implement your own methods. Using the process of stepwise refinement, you will be able to break up complex tasks into sets of cooperating methods.

Contents

- Methods as Black Boxes
- Implementing Methods
- Parameter Passing
- Return Values
- Methods without Return Values
- Problem Solving:
 - Reusable Methods
 - Stepwise Refinement
- Variable Scope
- Recursive Methods (optional)



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

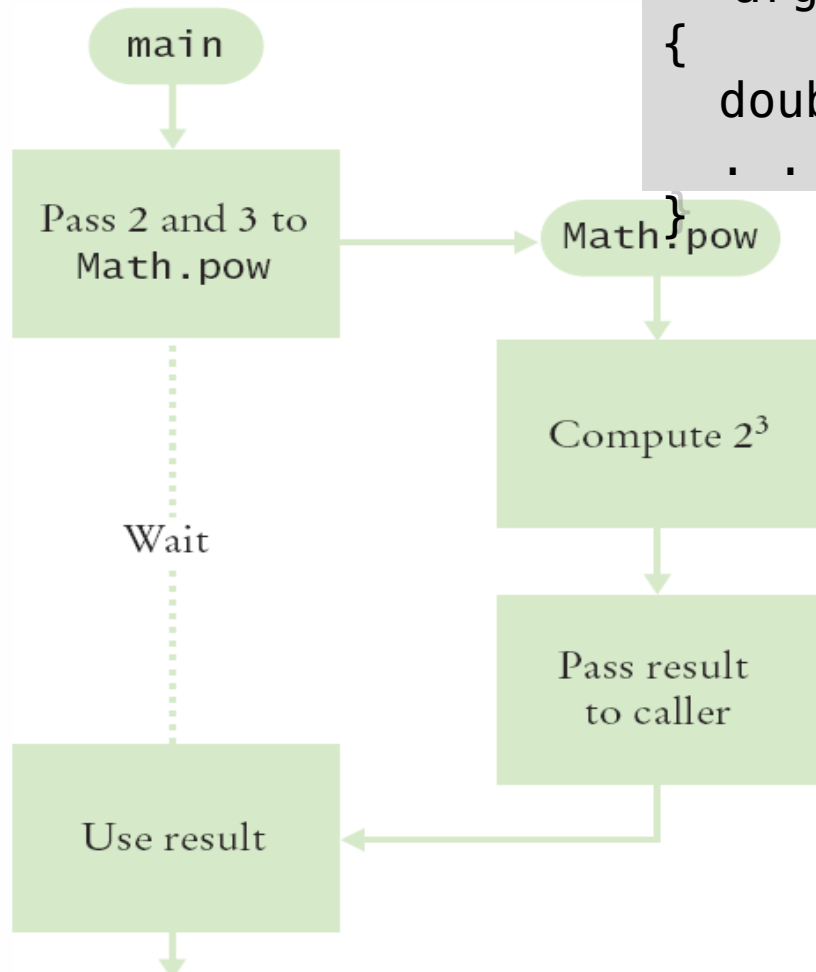
A method packages a computation consisting of multiple steps into a form that can be easily understood and reused.

What is a method?

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

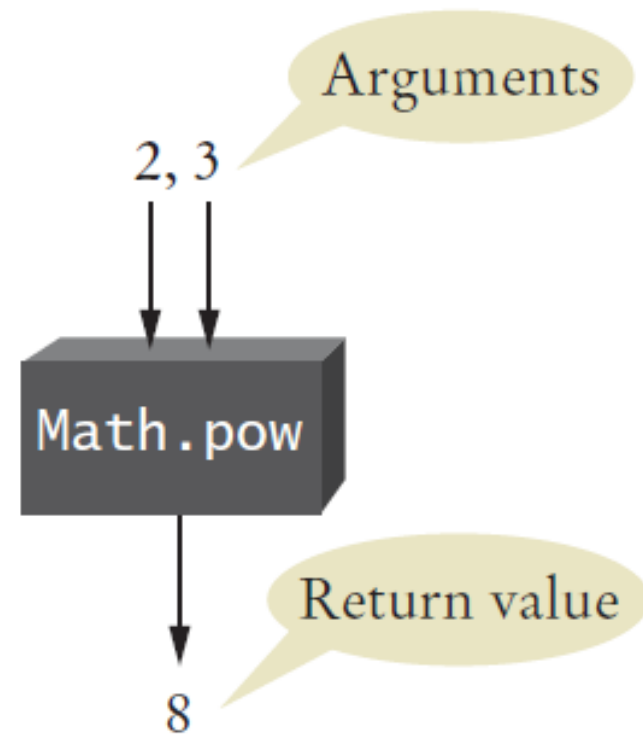
```
public static void main(String[]  
    args)  
{  
    double result = Math.pow(2, 3);  
    . . .  
}
```



- 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'

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



5.2 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 writing this method:
 - 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`
 - (see Chapter 8)



When declaring a method, you provide a name for the method, a variable for each argument, and a type for the result

```
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

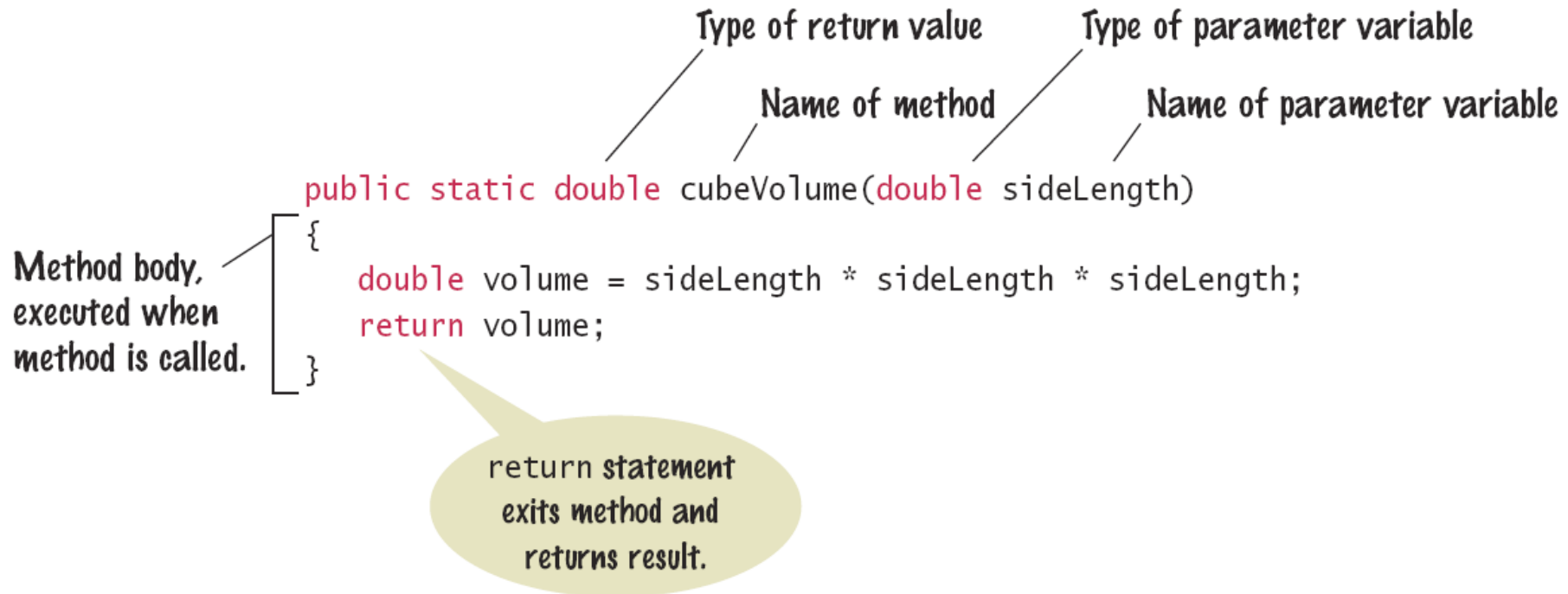
```
public static double cubeVolume(double sideLength)
{
    double volume = sideLength * sideLength *
    sideLength;
    return volume;
}
```

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);
}
```

Syntax 5.1: Method Declaration



Cubes.java

```
1  /**
2   * This program computes the volumes of two cubes.
3   */
4  public class Cubes
5  {
6      public static void main(String[] args)
7      {
8          double result1 = cubeVolume(2);
9          double result2 = cubeVolume(10);
10         System.out.println("A cube with side length 2 has volume " + result1);
11         System.out.println("A cube with side length 10 has volume " + result2);
12     }
13
14     /**
15      * Computes the volume of a cube.
16      * @param sideLength the side length of the cube
17      * @return the volume
18      */
19     public static double cubeVolume(double sideLength)
20     {
21         double volume = sideLength * sideLength * sideLength;
22         return volume;
23     }
24 }
```

Program Run

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

Method Comments

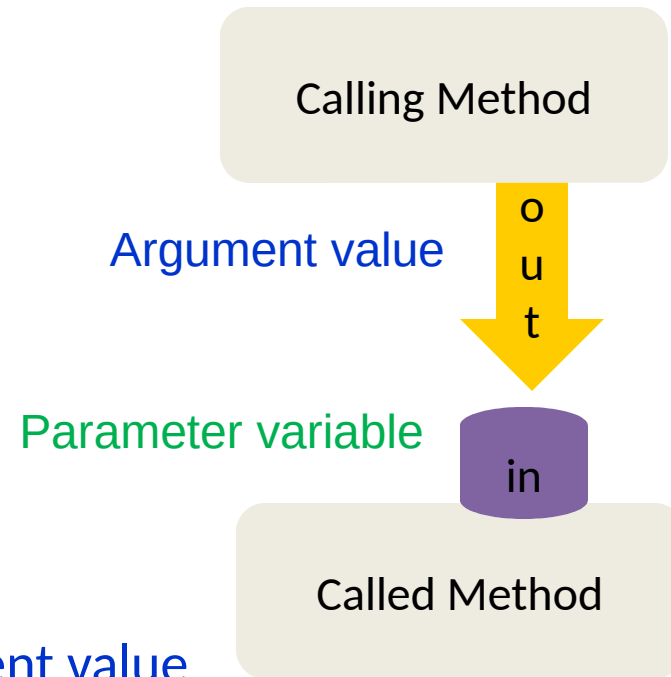


- Write a Javadoc comment above each method
- Start with `/**`
 - Note the purpose of the method
 - `@param` Describe each parameter variable
 - `@return` Describe the return value
- End with `*/`

```
/**  
    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 sideLength)  
{  
    double volume = sideLength * sideLength *  
        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 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

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

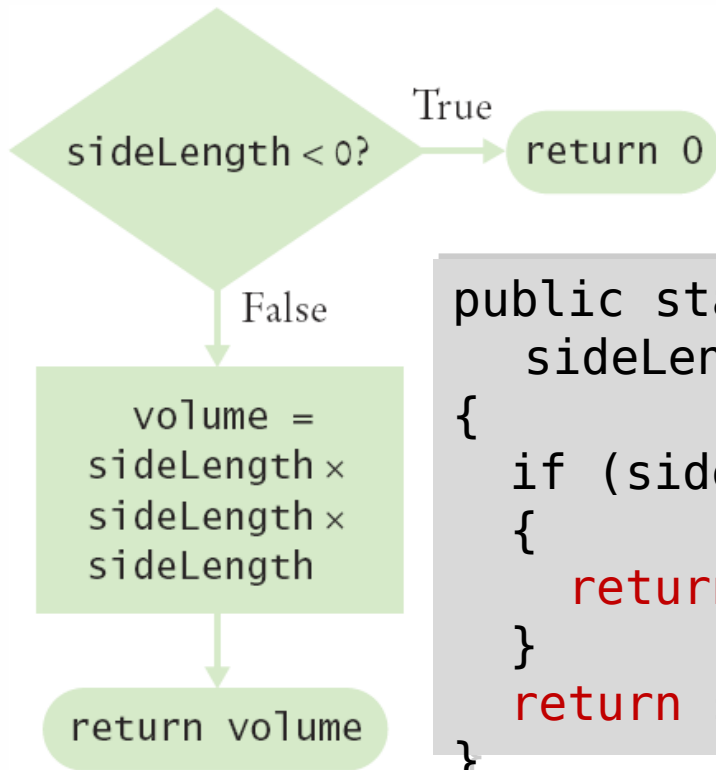
```
public static double cubeVolume (double sideLength)
{
    double volume = sideLength * sideLength *
    sideLength;
    return volume;
}
```

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;
}
```

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

Implementing a Method: Steps

- 1) Describe what the method should do.
- 2) Determine the method's "inputs".
- 3) Determine the types of parameter values and the return value.
- 4) Write pseudocode for obtaining the desired result.
- 5) Implement the method body.

```
public static double pyramidVolume(double  
    height, double baseLength)  
{  
    double baseArea = baseLength * baseLength;  
    return height * baseArea / 3;  
}
```

- 6) Test your method.
 - Design test cases and code

```
Volume: 300  
Expected: 300  
Volume: 0  
Expected: 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

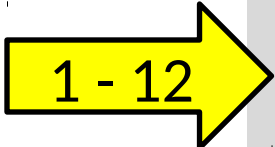
- You can use the `return` statement without a value
 - In methods with `void` return type
 - The method will terminate immediately!

```
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();
}
```

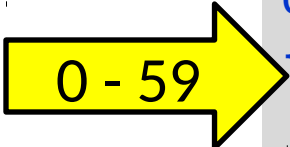
5.6 Problem Solving: Reusable Methods

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

```
int hours;  
do  
{  
    System.out.print("Enter a value between 1 and 12:  
    ");  
    hours = in.nextInt();  
}  
while (hours < 1 || hours > 12);  
  
int minutes;  
do  
{  
    System.out.print("Enter a value between 0 and 59:  
    ");  
    minutes = in.nextInt();  
}  
while (minutes < 0 || minutes > 59);
```



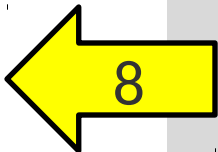
1 - 12



0 - 59

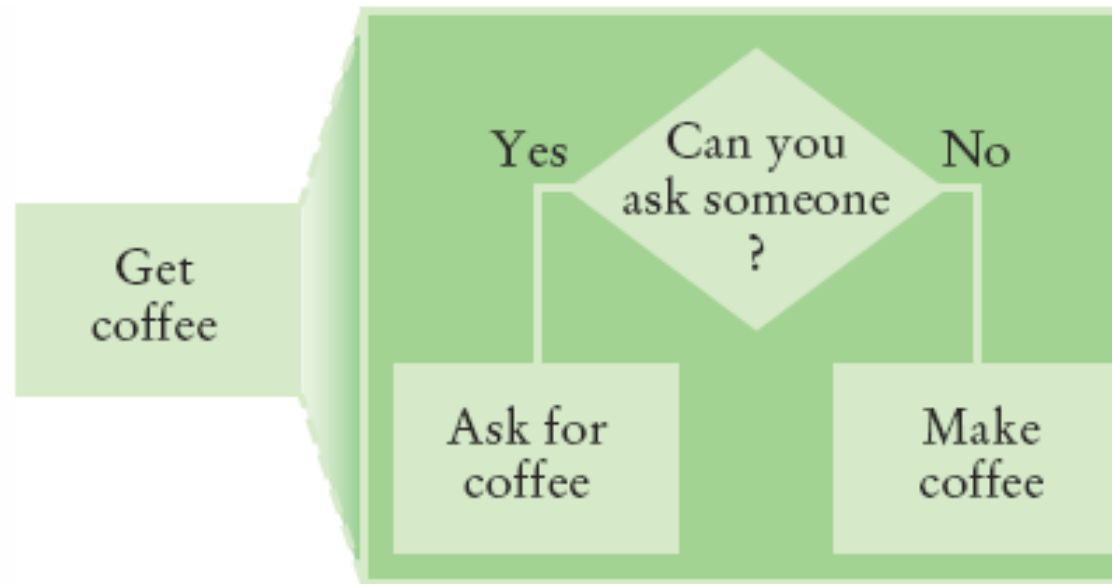
Write a 'Parameterized' Method

```
/**
    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)
{
    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;
}
```

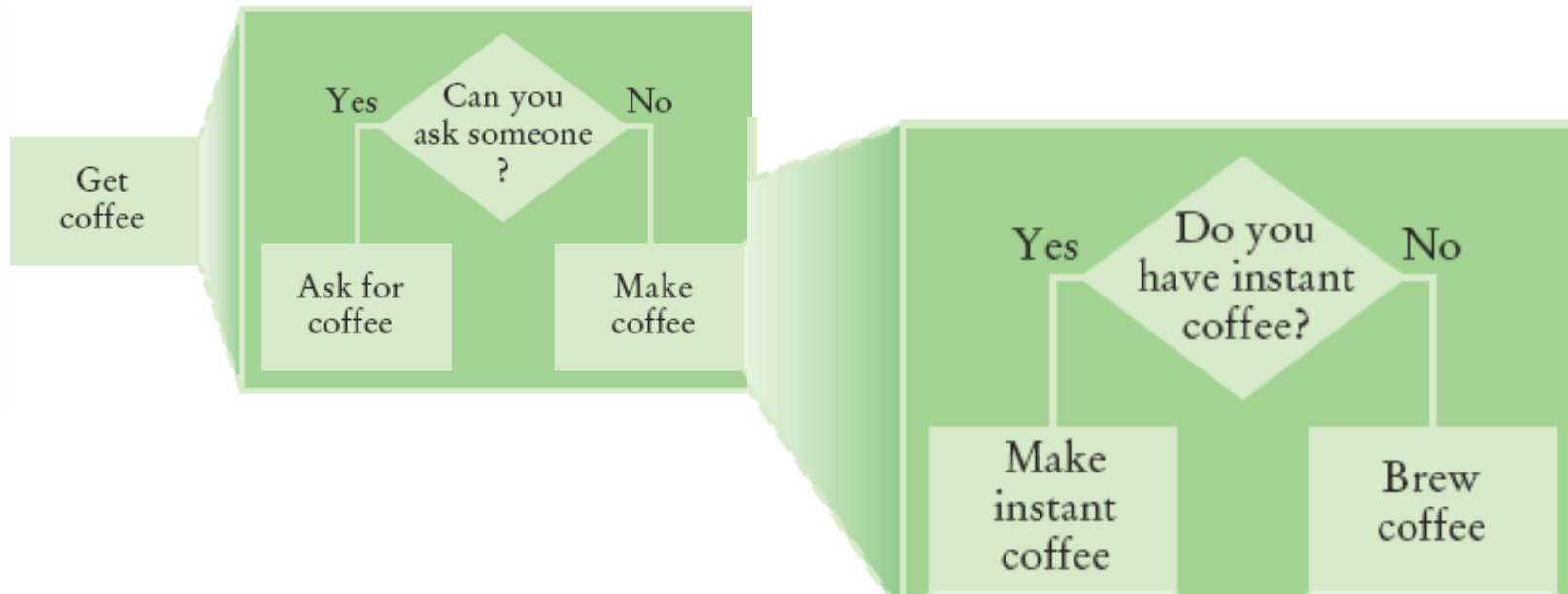


5.7 Problem Solving

- 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

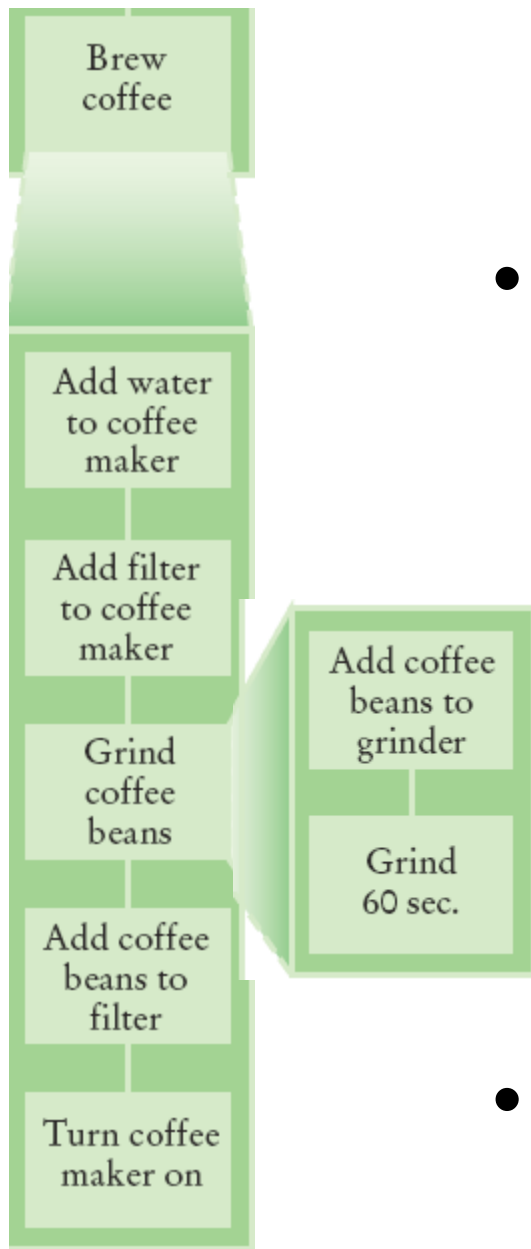


- If you must make coffee, there are two ways:
 - Make Instant Coffee
 - Brew Coffee

Instant Coffee

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



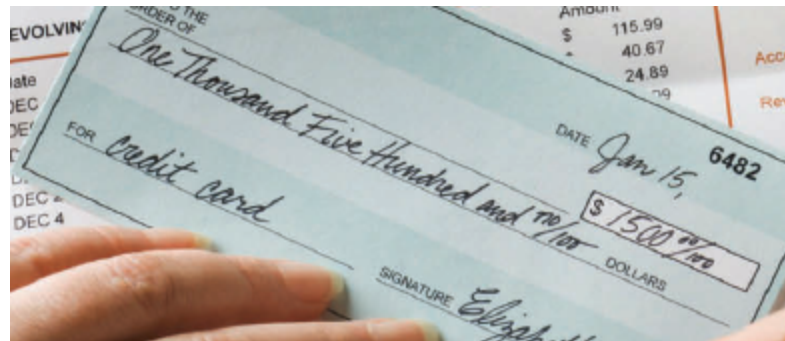


Brew Coffee

- 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

Stepwise Refinement Example

- When printing a check, it is customary to write the check amount both as a number (“\$274.15”) and as a text string (“two hundred seventy four dollars and 15 cents”). Write a program to turn a number into a text string.
- Wow, sounds difficult!
- Break it down
 - Let’s take the dollar part (274) and come up with a plan
 - Take an Integer from 0 – 999
 - Return a String
 - Still pretty hard...



Stepwise Refinement Example

- Take it digit by digit (2, 7, 4) – left to right
- Handle the first digit (hundreds)
 - If empty, we're done with hundreds
 - Get first digit (Integer from 1 – 9)
 - Get digit name (“one”, “two ”, “three”...)
 - Add the word “ hundred”
 - Sounds easy!
- Second digit (tens)
 - Get second digit (Integer from 0 – 9)
 - If 0, we are done with tens... handle third digit
 - If 1, ... may be eleven, twelve.. Teens... Not easy!
 - Let's look at each possibility left (1x-9x)...

Stepwise Refinement Example

- If second digit is a 0
 - Get third digit (Integer from 0 – 9)
 - Get digit name (“”, “one”, “two”...) ... Same as before?
 - Sounds easy!
- If second digit is a 1
 - Get third digit (Integer from 0 – 9)
 - Return a String (“ten”, “eleven”, “twelve”...)
- If second digit is a 2-9
 - Start with string “twenty”, “thirty”, “forty”...
 - Get third digit (Integer from 0 – 9)
 - Get digit name (“”, “one”, “two”...) ... Same as before
 - Sounds easy!

Name the Sub-Tasks

- `digitName`
 - Takes an Integer from 0 – 9
 - Return a String (“”, “one”, “two”...)
- `tensName` (second digit ≥ 20)
 - Takes an Integer from 0 – 9
 - Return a String (“twenty”, “thirty”...) plus
 - `digitName`(third digit)
- `teenName`
 - Takes an Integer from 0 – 9
 - Return a String (“ten”, “eleven”...)

Write Pseudocode

part = number (The part that still needs to be converted)

name = "" (The name of the number)

If part \geq 100

name = name of hundreds in part + "hundred"

Remove hundreds from part

Identify methods that we can use (or re-use!) to do the work.

If part \geq 20

Append tensName(part) to name.

Remove tens from part.

Else if part \geq 10

Append teenName(part) to name.

part = 0

If (part > 0)

Append digitName(part) to name.

Plan The Methods

- Decide on name, parameter(s) and types and return type
- `String digitName (int number)`
 - Return a String (“”, “one”, “two”...)
- `String tensName (int number)`
 - Return a String (“twenty”, “thirty”...) plus
 - Return from `digitName(thirdDigit)`
- `String teenName (int number)`
 - Return a String (“ten”, “eleven”...)

Convert to Java: `intName` method

```
21 public static String intName(int number)
22 {
23     int part = number; // The part that still needs to be converted
24     String name = ""; // The name of the number
25
26     if (part >= 100)
27     {
28         name = digitName(part / 100) + " hundred";
29         part = part % 100;
30
31         if (part >= 20)
32         {
33             name = name + " " + tensName(part);
34             part = part % 10;
35         }
36         else if (part >= 10)
37         {
38             name = name + " " + teenName(part);
39             part = 0;
40         }
41
42         if (part > 0)
43         {
44             name = name + " " + digitName(part);
45         }
46
47         return name;
48     }
49 }
```

- `main` calls `intName`
 - Does all the work
 - Returns a `String`
- Uses methods:
 - `tensName`
 - `teenName`
 - `digitName`

digitName, teenName, tensName

```
56 public static String digitName(int digit)
57 {
58     if (digit == 1) { return "one"; }
59     if (digit == 2) { return "two"; }
60     if (digit == 3) { return "three"; }
61     if (digit == 4) { return "four"; }
62     if (digit == 5) { return "five"; }
63     if (digit == 6) { return "six"; }
64     if (digit == 7) { return "seven"; }
65     if (digit == 8) { return "eight"; }
66     if (digit == 9) { return "nine"; }
67     return "";
68 }

81 public static String teenName(int number)
82 {
83     if (number == 10) { return "ten"; }
84     if (number == 11) { return "eleven"; }
85     if (number == 12) { return "twelve"; }
86     if (number == 13) { return "thirteen"; }
87     if (number == 14) { return "fourteen"; }
88     if (number == 15) { return "fifteen"; }
89     if (number == 16) { return "sixteen"; }
90     if (number == 17) { return "seventeen"; }
91     if (number == 18) { return "eighteen"; }
92     if (number == 19) { return "nineteen"; }
93     return "";
94 }

95 public static String tensName(int number)
96 {
97     if (number >= 90) { return "ninety"; }
98     if (number >= 80) { return "eighty"; }
99     if (number >= 70) { return "seventy"; }
100    if (number >= 60) { return "sixty"; }
101    if (number >= 50) { return "fifty"; }
102    if (number >= 40) { return "forty"; }
103    if (number >= 30) { return "thirty"; }
104    if (number >= 20) { return "twenty"; }
105    return "";
106 }
```

Program Run

Please enter a positive integer: 729
seven hundred twenty nine

Programming Tips

- Keep methods short
 - If more than one screen, break into 'sub' methods

intName(number = 416)	
part	name
416	iii
16	"four hundred"
0	"four hundred sixteen"

- Trace your methods
 - One line for each step
 - Columns for key variables
- Use Stubs as you write larger programs
 - Unfinished methods that return a 'dummy' value



```
public static String digitName(int
    digit)
{
    return "mumble";
}
```