



CSE 101 - COMPUTER PROGRAMMING I METHODS

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OPENING PROBLEM

Find the sum of integers from 1 to 10, from 20 to 30, and from 35 to 45, respectively.



PROBLEM

```
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
    sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
    sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

3



PROBLEM

```
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
    sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
    sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

4



SOLUTION

```
public static int sum(int i1, int i2) {
    int sum = 0;
    for (int i = i1; i <= i2; i++)
        sum += i;
    return sum;
}
```

```
public static void main(String[] args) {
    System.out.println("Sum from 1 to 10 is " + sum(1, 10));
    System.out.println("Sum from 20 to 30 is " + sum(20, 30));
    System.out.println("Sum from 35 to 45 is " + sum(35, 45));
}
```

5



OBJECTIVES

- To define methods with formal parameters (§6.2).
- To invoke methods with actual parameters (i.e., arguments) (§6.2).
- To define methods with a return value (§6.3).
- To define methods without a return value (§6.4).
- To pass arguments by value (§6.5).
- To develop reusable code that is modular, easy to read, easy to debug, and easy to maintain (§6.6).
- To write a method that converts hexadecimal to decimals (§6.7).
- To use method overloading and understand ambiguous overloading (§6.8).
- To determine the scope of variables (§6.9).
- To apply the concept of method abstraction in software development (§6.10).
- To design and implement methods using stepwise refinement (§6.10).

6



DEFINING METHODS

A method is a collection of statements that are grouped together to perform an operation.

Define a method

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Invoke a method

```
int z = max(x, y);
        ↑  ↑
    actual parameters
    (arguments)
```

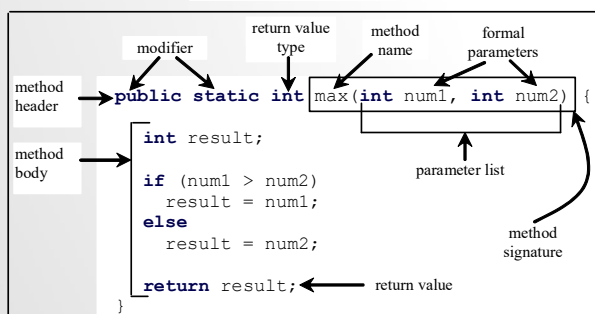
7



DEFINING METHODS

A method is a collection of statements that are grouped together to perform an operation.

Define a method



Invoke a method

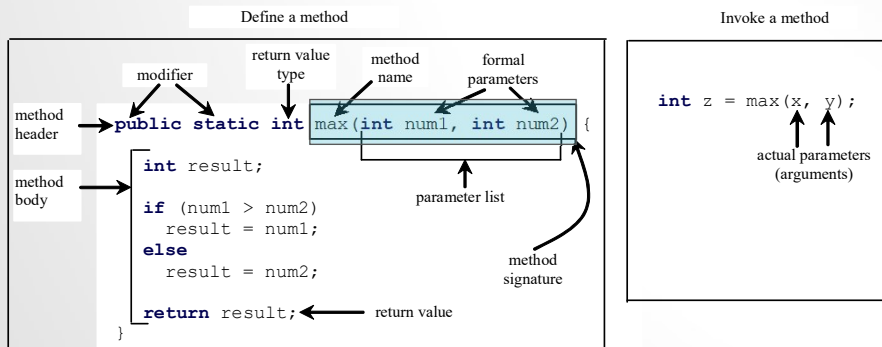
```
int z = max(x, y);
        ↑  ↑
    actual parameters
    (arguments)
```

8



METHOD SIGNATURE

Method signature is the combination of the method name and the parameter list.

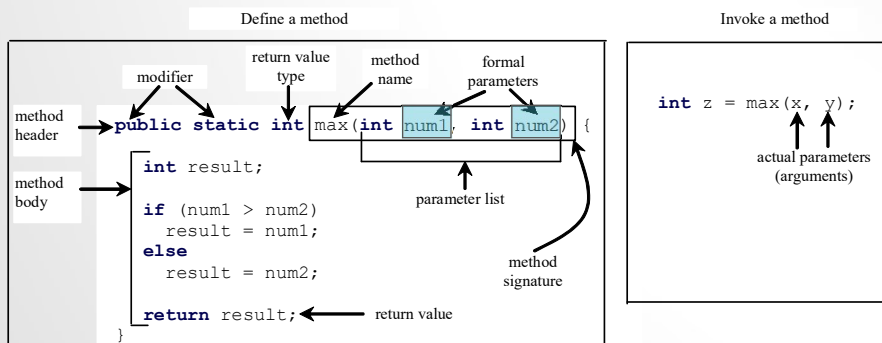


9



FORMAL PARAMETERS

The variables defined in the method header are known as *formal parameters*.

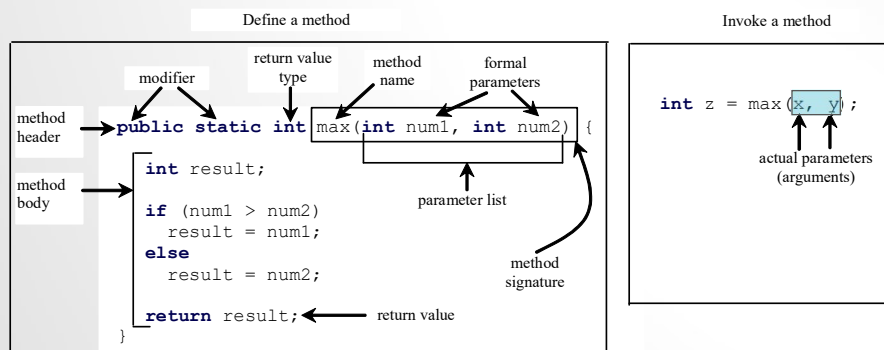


10



ACTUAL PARAMETERS

When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.

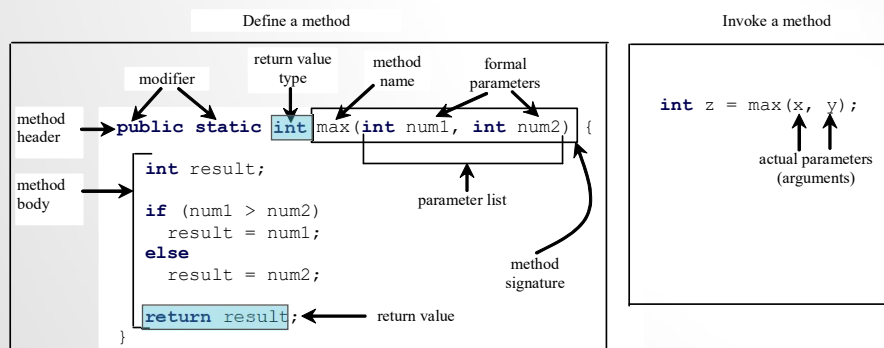


11



RETURN VALUE TYPE

A method may return a value. The returnValueType is the data type of the value the method returns. If the method does not return a value, the returnValueType is the keyword `void`. For example, the returnValueType in the main method is `void`.



12



CALLING METHODS

Testing the `max` method

This program demonstrates calling a method `max` to return the largest of the `int` values

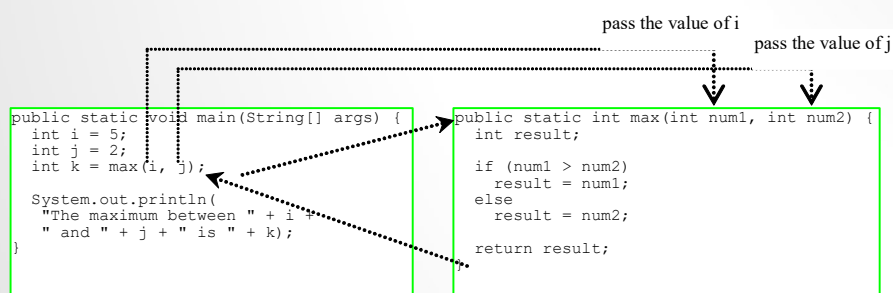


13



animation

CALLING METHODS, CONT.



14



animation

TRACE METHOD INVOCATION

i is now 5

```

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}

```

```

public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

```

15



animation

TRACE METHOD INVOCATION

j is now 2

```

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}

```

```

public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

```

16



animation

TRACE METHOD INVOCATION

invoke max(i, j)

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

17



animation

TRACE METHOD INVOCATION

invoke max(i, j)

Pass the value of i to num1
Pass the value of j to num2

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

18



animation

TRACE METHOD INVOCATION

declare variable result

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

19



animation

TRACE METHOD INVOCATION

(num1 > num2) is true since num1
is 5 and num2 is 2

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

20



animation

TRACE METHOD INVOCATION

result is now 5

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

21



animation

TRACE METHOD INVOCATION

return result, which is 5

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

22



animation

TRACE METHOD INVOCATION

return max(i, j) and assign the
return value to k

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

23



animation

TRACE METHOD INVOCATION

Execute the print statement

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

24



CAUTION

A return statement is required for a value-returning method. The method shown below in (a) is logically correct, but it has a compilation error because the Java compiler thinks it possible that this method does not return any value.

```
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else if (n < 0)
        return -1;
}
```

(a)

Should be

```
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else
        return -1;
}
```

(b)

To fix this problem, delete if ($n < 0$) in (a), so that the compiler will see a return statement to be reached regardless of how the if statement is evaluated.

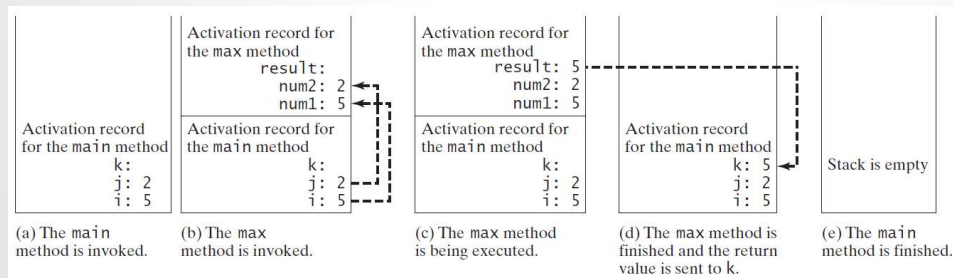


REUSE METHODS FROM OTHER CLASSES

NOTE: One of the benefits of methods is for reuse. The max method can be invoked from any class besides TestMax. If you create a new class Test, you can invoke the max method using ClassName.methodName (e.g., TestMax.max).



CALL STACKS



27



animation

TRACE CALL STACK

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

`i` is declared and initialized



The main method is invoked.

28



animation

TRACE CALL STACK

j is declared and initialized

```

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}

```

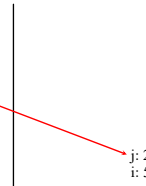
```

public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

```



The main method
is invoked.

29



animation

TRACE CALL STACK

Declare k

```

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}

```

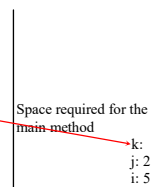
```

public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

```



The main method
is invoked.

30



animation

TRACE CALL STACK

Invoke max(i, j)

```

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}

```

```

public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

```

Space required for the main method

k:
j: 2
i: 5

The main method is invoked.

31



animation

TRACE CALL STACK

pass the values of i and j to num1 and num2

```

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}

```

```

public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

```

Space required for the main method

num2: 2
num1: 5

k:
j: 2
i: 5

The max method is invoked.

32



animation

TRACE CALL STACK

Declare result

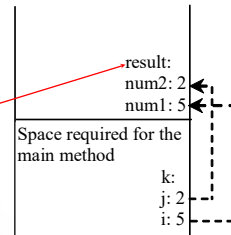
```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```



The max method is
invoked.

33



animation

TRACE CALL STACK

(num1 > num2) is true

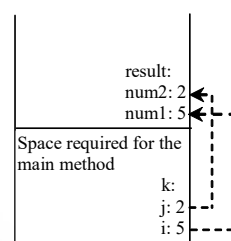
```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```



The max method is
invoked.

34



animation

TRACE CALL STACK

Assign num1 to result

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2)
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Space required for the
max method

result: 5
num2: 2
num1: 5

Space required for the
main method

k: 2
j: 2
i: 5

The max method is
invoked.

35



animation

TRACE CALL STACK

Return result and assign it to k

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2)
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Space required for the
max method

result: 5
num2: 2
num1: 5

Space required for the
main method

k: 5
j: 2
i: 5

The max method is
invoked.

36



animation

TRACE CALL STACK

Execute print statement

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
```

```
    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Space required for the
main method

k:5
j:2
i:5

The main method
is invoked.

37



VOID METHOD EXAMPLE

This type of method does not return a value. The method performs some actions.

TestVoidMethod

Run

TestReturnGradeMethod

Run

38



PASSING PARAMETERS

```
public static void nPrintln(String message, int n) {
    for (int i = 0; i < n; i++)
        System.out.println(message);
}
```

Suppose you invoke the method using
nPrintln("Welcome to Java", 5);

What is the output?

Suppose you invoke the method using
nPrintln("Computer Science", 15);

What is the output?

Can you invoke the method using
nPrintln(15, "Computer Science");

39



PASS BY VALUE

This program demonstrates passing values
to the methods.



Increment

Run

40



PASS BY VALUE

Testing Pass by value

This program demonstrates passing values to the methods.



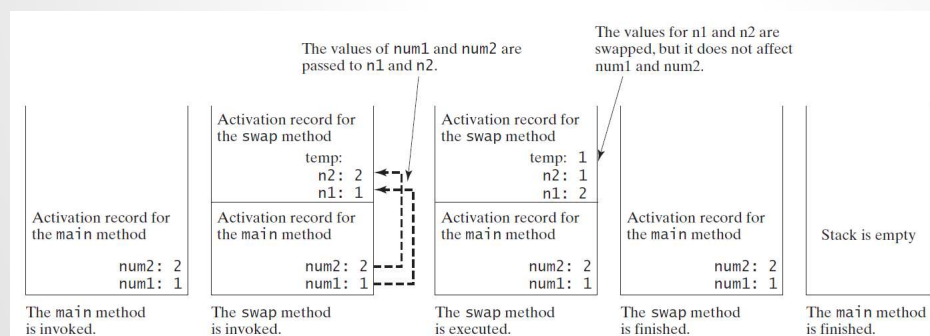
[TestPassByValue](#)

Run



41

PASS BY VALUE, CONT.



42

MODULARIZING CODE

Methods can be used to reduce redundant coding and enable code reuse. Methods can also be used to modularize code and improve the quality of the program.



GreatestCommonDivisorMethod

Run



PrimeNumberMethod

Run

43



CASE STUDY: CONVERTING HEXADECIMALS TO DECIMALS

Write a method that converts a hexadecimal number into a decimal number.

ABCD =>

$$A*16^3 + B*16^2 + C*16^1 + D*16^0$$

$$= ((A*16 + B)*16 + C)*16 + D$$

$$= ((10*16 + 11)*16 + 12)*16 + 13 = ?$$



Hex2Dec

Run

44



OVERLOADING METHODS

Overloading the `max` Method

```
public static double max(double num1, double  
num2) {  
    if (num1 > num2)  
        return num1;  
    else  
        return num2;  
}
```



[TestMethodOverloading](#)

Run



45

AMBIGUOUS INVOCATION

Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match. This is referred to as *ambiguous invocation*. Ambiguous invocation is a compile error.



46

AMBIGUOUS INVOCATION

```
public class AmbiguousOverloading {
    public static void main(String[] args) {
        System.out.println(max(1, 2));
    }

    public static double max(int num1, double num2) {
        if (num1 > num2)
            return num1;
        else
            return num2;
    }

    public static double max(double num1, int num2) {
        if (num1 > num2)
            return num1;
        else
            return num2;
    }
}
```

47



SCOPE OF LOCAL VARIABLES

A local variable: a variable defined inside a method.

Scope: the part of the program where the variable can be referenced.

The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared before it can be used.

48



SCOPE OF LOCAL VARIABLES, CONT.

You can declare a local variable with the same name multiple times in different non-nesting blocks in a method, but you cannot declare a local variable twice in nested blocks.

49



SCOPE OF LOCAL VARIABLES, CONT.

A variable declared in the initial action part of a for loop header has its scope in the entire loop. But a variable declared inside a for loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable.

```

public static void method1() {
    .
    .
    for (int i = 1; i < 10; i++) {
        .
        .
        int j;
        .
        .
    }
}

```

The scope of i →

The scope of j →

 A diagram showing the scope of variables 'i' and 'j'. A bracket on the left side of the 'for' loop header indicates that the scope of 'i' extends from the start of the loop to the end of the method. Another bracket on the left side of the loop body indicates that the scope of 'j' is limited to the body of the loop.

50



SCOPE OF LOCAL VARIABLES, CONT.

It is fine to declare `i` in two non-nesting blocks

```
public static void method1() {
    int x = 1;
    int y = 1;
    for (int i = 1; i < 10; i++) {
        x += i;
    }
    for (int i = 1; i < 10; i++) {
        y += i;
    }
}
```

It is wrong to declare `i` in two nesting blocks

```
public static void method2() {
    int i = 1;
    int sum = 0;
    for (int i = 1; i < 10; i++) {
        sum += i;
    }
}
```

51



SCOPE OF LOCAL VARIABLES, CONT.

```
// Fine with no errors
public static void correctMethod() {
    int x = 1;
    int y = 1;
    // i is declared
    for (int i = 1; i < 10; i++) {
        x += i;
    }
    // i is declared again
    for (int i = 1; i < 10; i++) {
        y += i;
    }
}
```

52



SCOPE OF LOCAL VARIABLES, CONT.

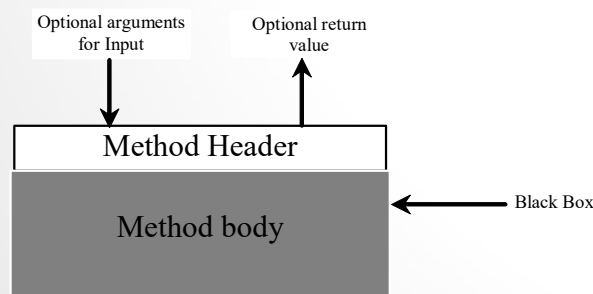
```
// With errors
public static void incorrectMethod() {
    int x = 1;
    int y = 1;
    for (int i = 1; i < 10; i++) {
        int x = 0;
        x += i;
    }
}
```



53

METHOD ABSTRACTION

You can think of the method body as a black box that contains the detailed implementation for the method.



54

BENEFITS OF METHODS

- Write a method once and reuse it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity.

55



CASE STUDY: GENERATING RANDOM CHARACTERS

Computer programs process numerical data and characters. You have seen many examples that involve numerical data. It is also important to understand characters and how to process them.

As introduced in Section 2.9, each character has a unique Unicode between 0 and FFFF in hexadecimal (65535 in decimal). To generate a random character is to generate a random integer between 0 and 65535 using the following expression: (note that since $0 \leq \text{Math.random}() < 1.0$, you have to add 1 to 65535.)

```
(int)(Math.random() * (65535 + 1))
```

56



CASE STUDY: GENERATING RANDOM CHARACTERS, CONT.

Now let us consider how to generate a random lowercase letter. The Unicode for lowercase letters are consecutive integers starting from the Unicode for 'a', then for 'b', 'c', ..., and 'z'. The Unicode for 'a' is

`(int)'a'`

So, a random integer between `(int)'a'` and `(int)'z'` is

`(int)((int)'a' + Math.random() * ((int)'z' - (int)'a' + 1))`

57



CASE STUDY: GENERATING RANDOM CHARACTERS, CONT.

As discussed in Chapter 2., all numeric operators can be applied to the char operands. The char operand is cast into a number if the other operand is a number or a character. So, the preceding expression can be simplified as follows:

`'a' + Math.random() * ('z' - 'a' + 1)`

So a random lowercase letter is

`(char)('a' + Math.random() * ('z' - 'a' + 1))`

58



CASE STUDY: GENERATING RANDOM CHARACTERS, CONT.

To generalize the foregoing discussion, a random character between any two characters $ch1$ and $ch2$ with $ch1 < ch2$ can be generated as follows:

$$(\text{char})(ch1 + \text{Math.random()} * (ch2 - ch1 + 1))$$

59



THE RANDOMCHARACTER CLASS

```
// RandomCharacter.java: Generate random characters
public class RandomCharacter {
    /** Generate a random character between ch1 and ch2 */
    public static char getRandomCharacter(char ch1, char ch2) {
        return (char)(ch1 + Math.random() * (ch2 - ch1 + 1));
    }

    /** Generate a random lowercase letter */
    public static char getRandomLowerCaseLetter() {
        return getRandomCharacter('a', 'z');
    }

    /** Generate a random uppercase letter */
    public static char getRandomUpperCaseLetter() {
        return getRandomCharacter('A', 'Z');
    }

    /** Generate a random digit character */
    public static char getRandomDigitCharacter() {
        return getRandomCharacter('0', '9');
    }

    /** Generate a random character */
    public static char getRandomCharacter() {
        return getRandomCharacter('\u0000', '\uFFFF');
    }
}
```



[RandomCharacter](#)



[TestRandomCharacter](#)

Run

60



STEPWISE REFINEMENT (OPTIONAL)

The concept of method abstraction can be applied to the process of developing programs. When writing a large program, you can use the “divide and conquer” strategy, also known as *stepwise refinement*, to decompose it into subproblems. The subproblems can be further decomposed into smaller, more manageable problems.

61



PRINTCALENDER CASE STUDY

Let us use the PrintCalendar example to demonstrate the stepwise refinement approach.

```

C:\book>java PrintCalendar
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 4
      April 2009
-----
Sun Mon Tue Wed Thu Fri Sat
      1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
C:\book>
  
```


[PrintCalendar](#)
[Run](#)

62



DESIGN DIAGRAM

printCalendar
(main)

63



DESIGN DIAGRAM

printCalendar
(main)

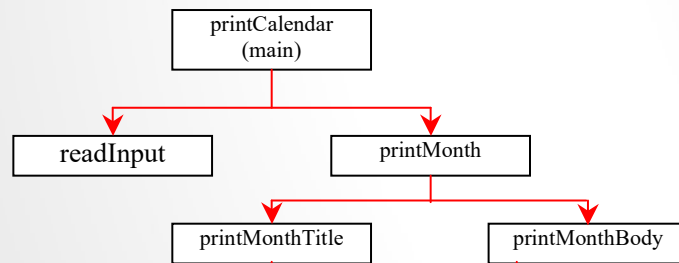
readInput

printMonth

64



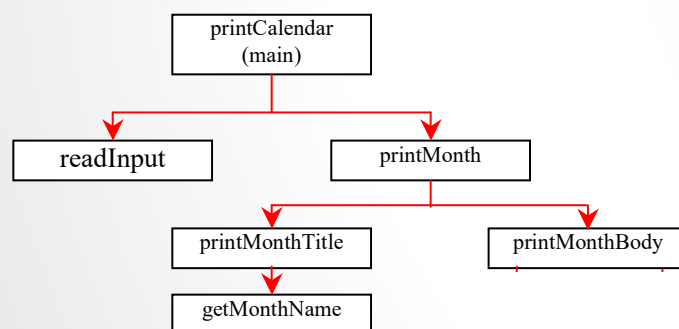
DESIGN DIAGRAM



65



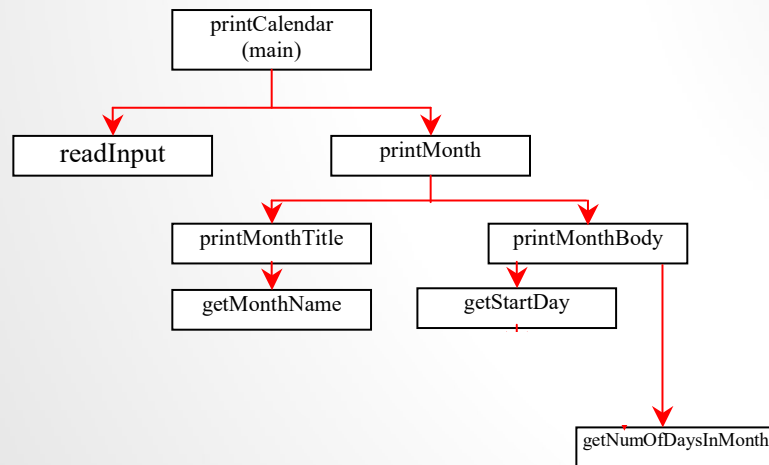
DESIGN DIAGRAM



66



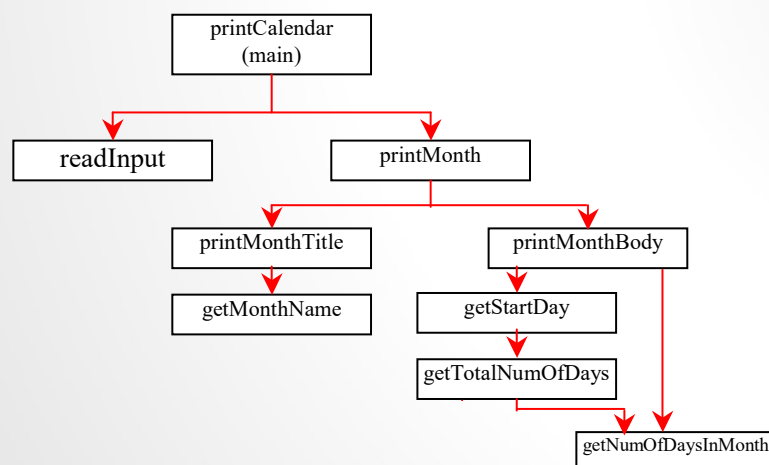
DESIGN DIAGRAM



67



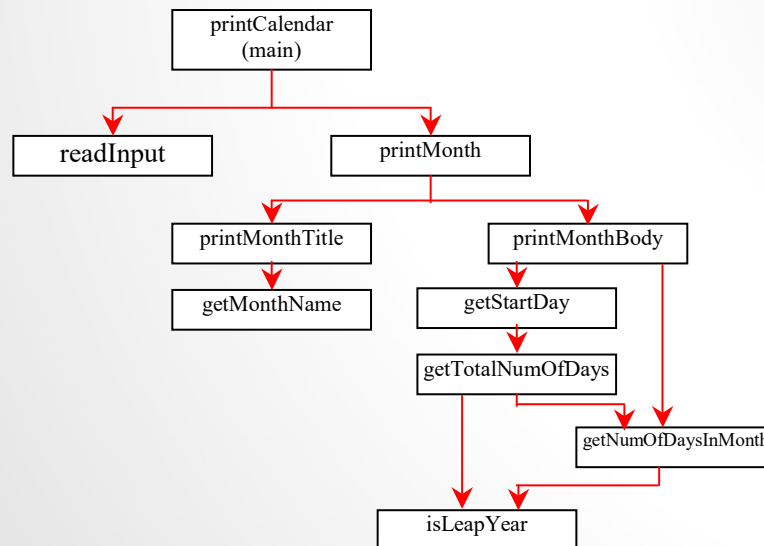
DESIGN DIAGRAM



68



DESIGN DIAGRAM



69



IMPLEMENTATION: TOP-DOWN

Top-down approach is to implement one method in the structure chart at a time from the top to the bottom. Stubs can be used for the methods waiting to be implemented. A stub is a simple but incomplete version of a method. The use of stubs enables you to test invoking the method from a caller. Implement the main method first and then use a stub for the printMonth method. For example, let printMonth display the year and the month in the stub. Thus, your program may begin like this:

[A Skeleton for printCalendar](#)

70



IMPLEMENTATION: BOTTOM-UP

Bottom-up approach is to implement one method in the structure chart at a time from the bottom to the top. For each method implemented, write a test program to test it. Both top-down and bottom-up methods are fine. Both approaches implement the methods incrementally and help to isolate programming errors and makes debugging easy. Sometimes, they can be used together.

71



BENEFITS OF STEPWISE REFINEMENT

Simpler Program

Reusing Methods

Easier Developing, Debugging, and Testing

Better Facilitating Teamwork

72



SOME EXAMPLES

- Palindrome
 - String that is the same forward and backward
- Number Rectangle
 - Given two integers (`row` and `col`), print `row` rows of the numbers 1 through `col`
- Substring to a character
 - Given a String and a character, get the String up to the first time that character appears
- Count characters
 - Get the number of times that a character appears in a String
- Menu
 - Display a set of menu options



73

PALINDROME

- A text that is the same forward and backward

- racecar



- mom



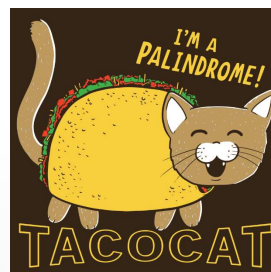
- kayak



- noon



- tacocat



74

NUMBER RECTANGLE

- N rows of the numbers 1 to M
- 5 and 5 2 and 3

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

1	2	3
1	2	3



75

STRING METHODS

- Substring to a character
 - Get the characters in a string up to the first time a certain character appears
 - "CSE 101 is awesome!" and 'm' gives "CSE 101 is aweso"
- Count the number of times a character appears in a string
 - "bir berber bir berbere" and 'b' gives 6



76

MENU

- A method that only displays a set of menu options
- Consider how to return a value from the menu

