Chapter 6 Methods

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 \le 30; i++)
  sum += i;
System.out.println("Sum from 20 to 30 is " + sum);
sum = 0;
for (int i = 35; i \le 45; i++)
 sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

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 \le 30; i++)
  sum += i;
System.out.println("Sum from 20 to 30 is " + sum);
sum = 0;
for (int i = 35; i \le 45; i++)
sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

Solution

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

```
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));
}
```

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 ($\S 6.6$).
- To write a method that converts hexadecimals 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).

Defining Methods

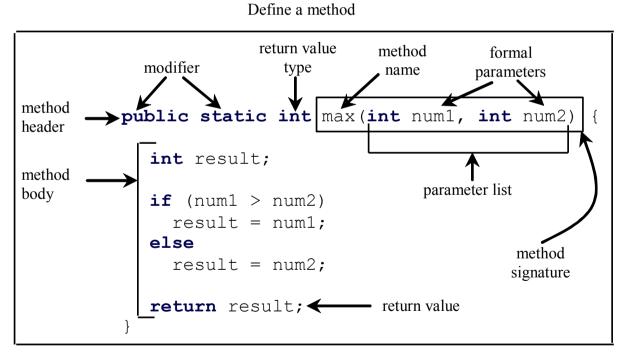
A method is a collection of statements that are grouped together to perform an operation.

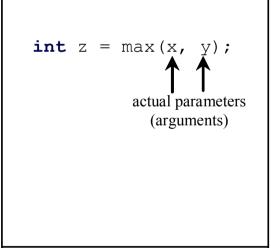
Define a method

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

Defining Methods

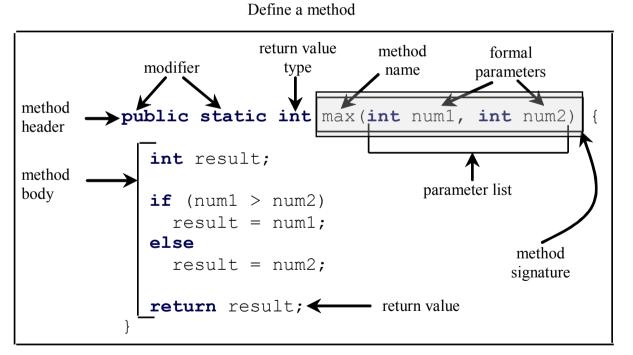
A method is a collection of statements that are grouped together to perform an operation.

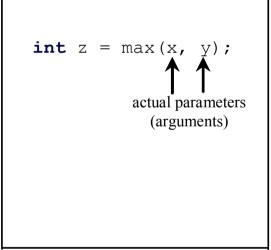




Method Signature

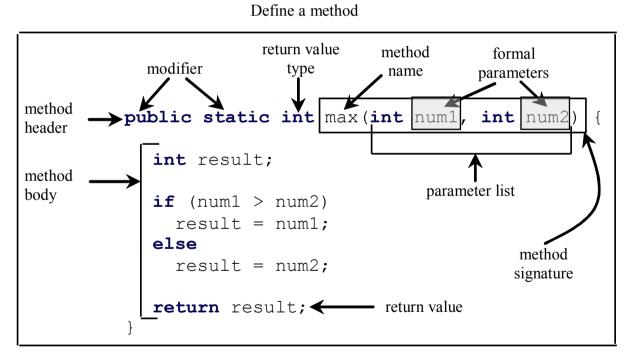
Method signature is the combination of the method name and the parameter list.

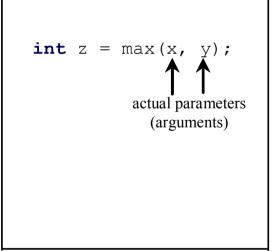




Formal Parameters

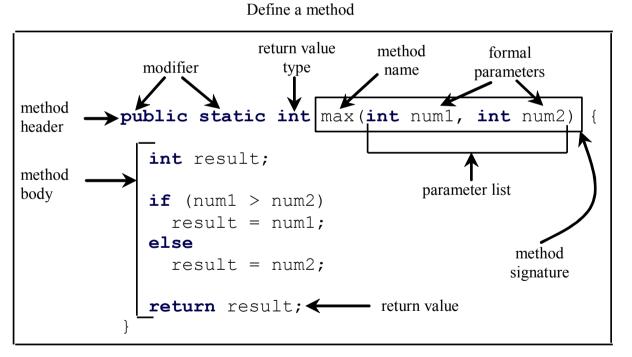
The variables defined in the method header are known as *formal parameters*.

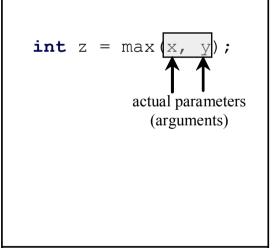




Actual Parameters

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

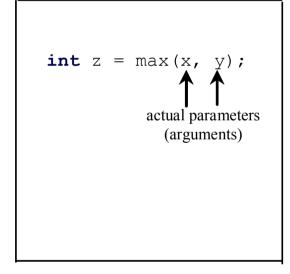




Return Value Type

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

Define a method return value method formal modifier type name parameters method public static int max (int num1, int num2) header int result; method parameter list body if (num1 > num2) result = num1; else method result = num2; signature return result; return value

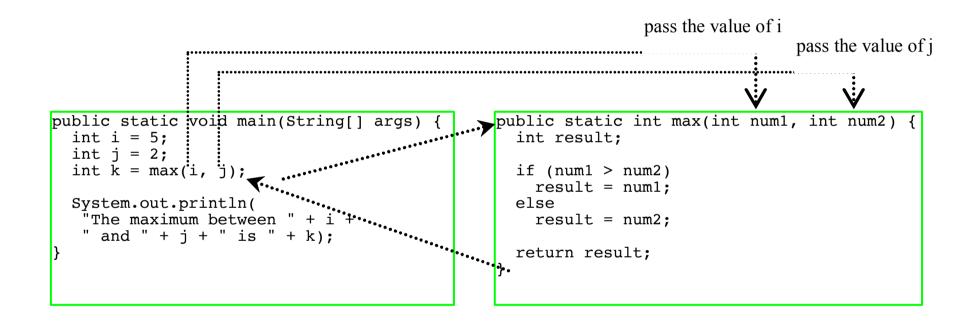


Calling Methods

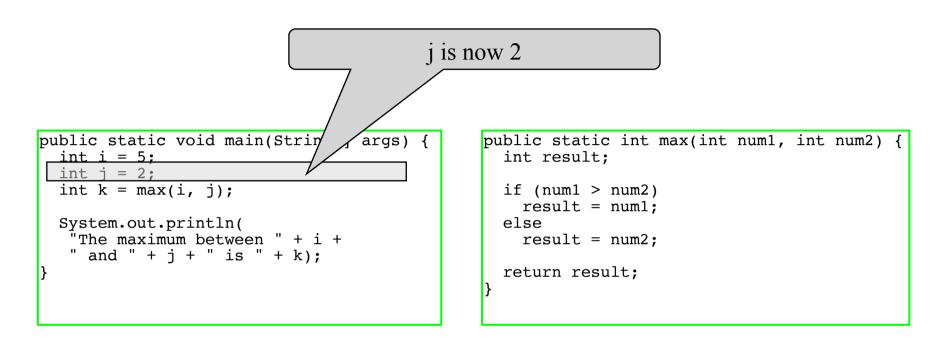
Testing the max method

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

Calling Methods, cont.



i is now 5 public static void main(Stri public static int max(int num1, int num2) { 1 args) int i = 5; int result; int j = 2; int k = max(i, j);if (num1 > num2)result = num1; System.out.println(else "The maximum between " + i + result = num2; " and " + j + " is " + k); return result;



```
public static void main(Strin 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;
}

return result;
}
```

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;
}
```

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 at max(int num1, int num2) {
   int result;

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

   return result;
}
```

(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 result;

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

return result;
}
```

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 stati
    max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;

    else
        result = num2;

    return 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);
}

return result, which is 5

pu tatic int max(int num1, int num2) {
  esult;
  inum1 > num2)
  esult = num1;
  se
  result = num2;
  return result;
}
```

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

```
public static void main(Strin 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;
}
```

Execute the print statement

```
public static void main(String
  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;
}
```

CAUTION

A <u>return</u> 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.

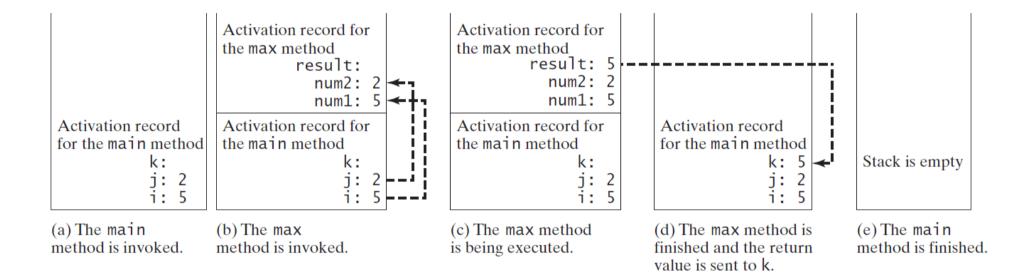
```
public static int sign(int n) {
                                              public static int sign(int n) {
  if (n > 0)
                                                 if (n > 0)
                                     Should be
    return 1;
                                                   return 1;
  else if (n == 0)
                                                 else if (n == 0)
    return 0;
                                                   return 0;
  else if (n < 0)</pre>
    return −1;
                                                   return −1;
                 (a)
                                                                 (b)
```

To fix this problem, delete \underline{if} (n < 0) in (a), so that the compiler will see a <u>return</u> statement to be reached regardless of how the \underline{if} statement is evaluated.

Reuse Methods from Other Classes

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

Call Stacks

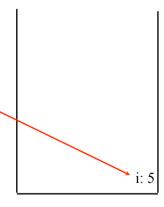


i is declared and initialized

```
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.

j is declared and initialized public static void main(String[] argo int i = 5; int i = 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; The main method else is invoked. result = num2; return result;

Declare k

```
public static void main(Strings) {
  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.

Invoke max(i, j) public static void main(String[] args)/ int i = 5; int j = 2; int $k = \max(i,$ System.out.println("The maximum between " + i + " and " + j + " is " + k); Space required for the main method k: public static int max(int num1, int num2) { int result; if (num1 > num2) result = num1; The main method else is invoked. result = num2; return result;

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); num2: 2 num1: 5 Space required for the public static int max(int num1, int num2 main method int result; k: if (num1 > num2)result = num1; else result = num2; The max method is return result; invoked.

```
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 +
                                                                        result:
   " and " + j + " is " + k);
                                                                        num2: 2
                                                                        num1: 5
                                                             Space required for the
public static int max(int num1, int num2
                                                             main method
 int result;
                                                                            k:
  if (num1 > num2)
    result = num1;
  else
    result = num2;
                                                               The max method is
  return result;
                                                              invoked.
```

```
(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 +
                                                                        result:
   " and " + j + " is " + k);
                                                                        num2: 2
                                                                        num1: 5
                                                             Space required for the
public static int max(int num1, int num2
                                                             main method
  int result;
                                                                           k:
 if (num1 > num2)
    result = num1;
  else
    result = num2;
                                                              The max method is
  return result;
                                                              invoked.
```

```
Assign num1 to result
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);
                                                               Space required for the
                                                               max method
  System.out.println(
   "The maximum between " + i +
                                                                          result: 5
   " and " + j + " is " + k);
                                                                          num2: 2
                                                                          num1: 5
                                                               Space required for the
public static int max(int num1, int num2)/
                                                               main method
  int result;
                                                                             k:
  if (num1 > num2)
    result = num1;
  else
    result = num2;
                                                                The max method is
  return result;
                                                                invoked.
```

Return result and assign it to k

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);
                                                              Space required for the
                                                              max method
  System.out.println(
   "The maximum between " + i +
                                                                         result: 5
   " and " + j + " is " + k);
                                                                         num2: 2
                                                                         num1: 5
                                                              Space required for the
public static int max(int num1, int num/
                                                              main method
  int result;
  if (num1 > num2)
    result \= num1;
  else
    result = num2;
                                                               The max method is
  return result:
                                                               invoked.
```

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.

void Method Example

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

<u>TestVoidMethod</u>

<u>TestReturnGradeMethod</u>

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");
```

Pass by Value

This program demonstrates passing values to the methods.

Increment

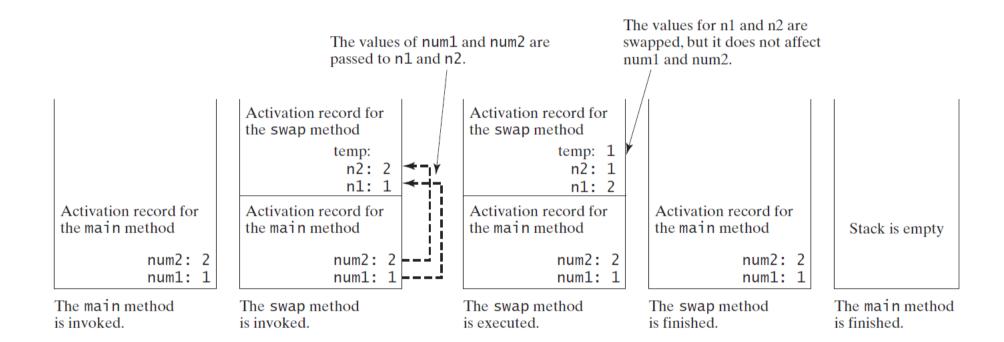
Pass by Value

Testing Pass by value

This program demonstrates passing values to the methods.

TestPassByValue

Pass by Value, cont.



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.

<u>GreatestCommonDivisorMethod</u>

PrimeNumberMethod

Overloading Methods

Overloading the max Method

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

<u>TestMethodOverloading</u>

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.

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;
```

Scope of Local Variables

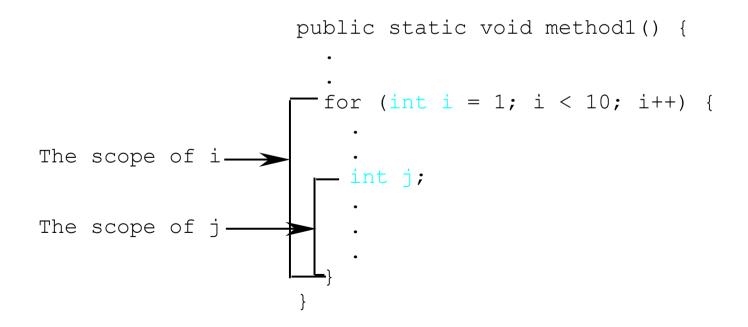
A local variable: a variable defined inside a method.

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.

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.

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



```
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;
   }
}</pre>
```

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

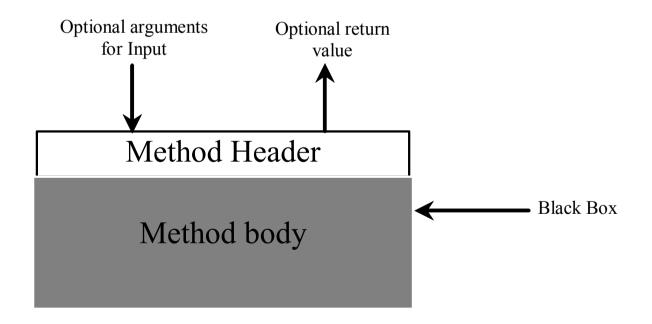
public static void method2() {
   int i = 1;
   int sum = 0;
   for (int i = 1; i < 10; i++)
      sum += i;
   }
}</pre>
```

```
// 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;
```

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

Method Abstraction

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



Benefits of Methods

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

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 <= Math.random() < 1.0, you have to add 1 to 65535.)

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

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)

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)

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))

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

```
(char)(ch1 + Math.random() * (ch2 - ch1 + 1))
```

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

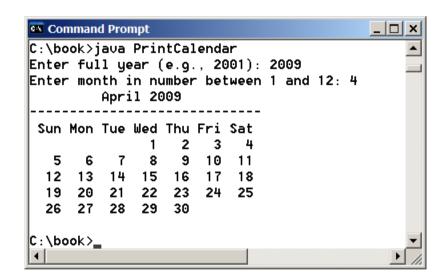
TestRandomCharacter

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.

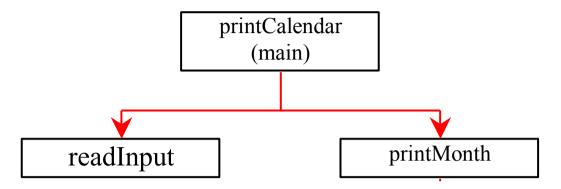
PrintCalender Case Study

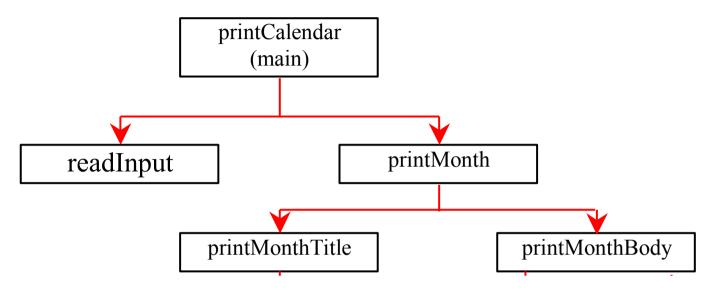
Let us use the PrintCalendar example to demonstrate the stepwise refinement approach.

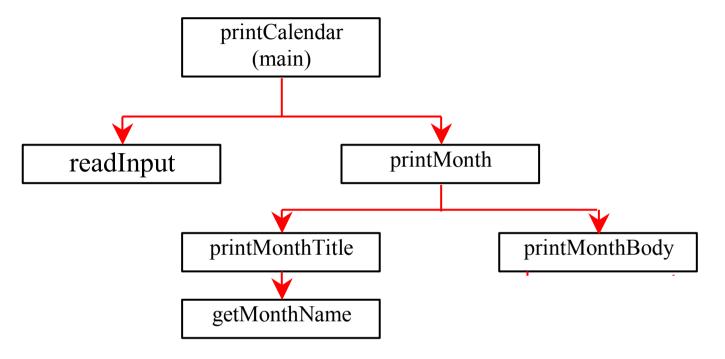


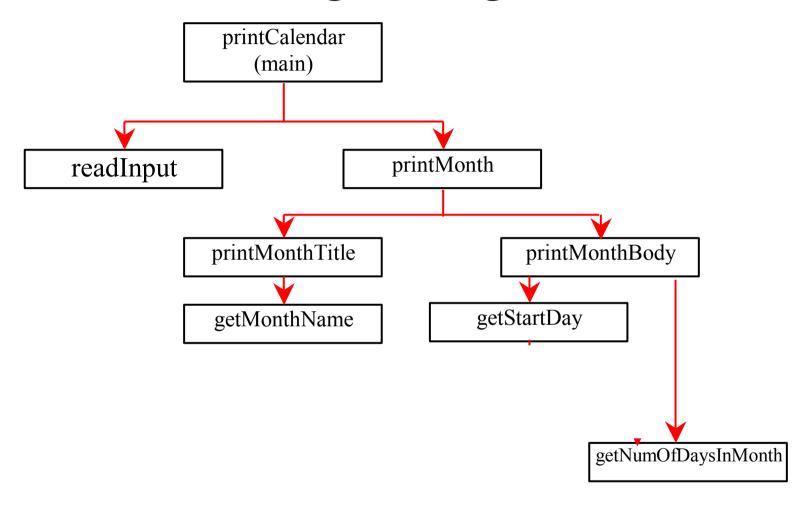
PrintCalendar

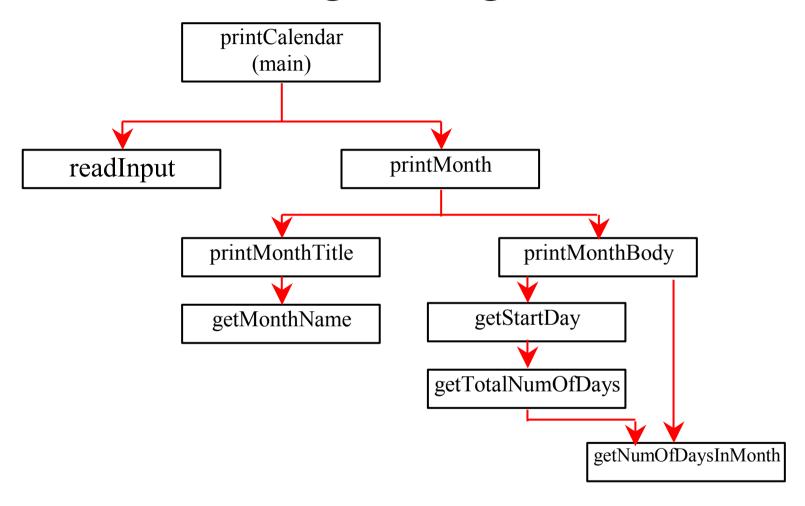
printCalendar (main)

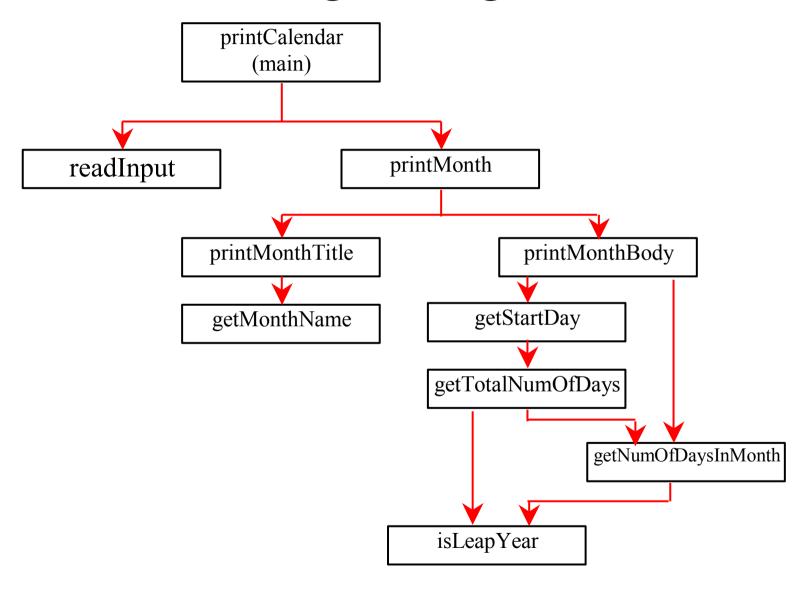












Implementation: Top-Down

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

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.

Benefits of Stepwise Refinement

Simpler Program

Reusing Methods

Easier Developing, Debugging, and Testing

Better Facilitating Teamwork

编程练习题(10.24周一交)

6.1 Converting Hexadecimals to Decimals

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

$$ABCD =>$$

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

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

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