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 <= 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);
```

Problem

```java
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);
```

Solution

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

MethodDemo   Run

```
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));
}
```

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;
}
```
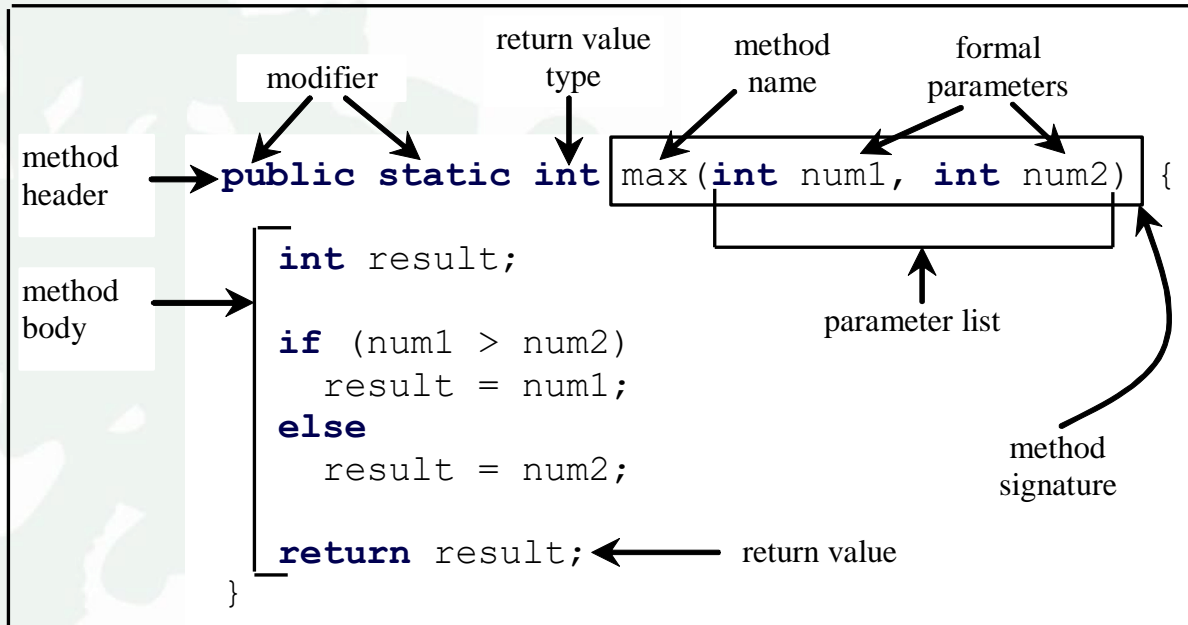
Invoke a method

```
int z = max(x, y);
```

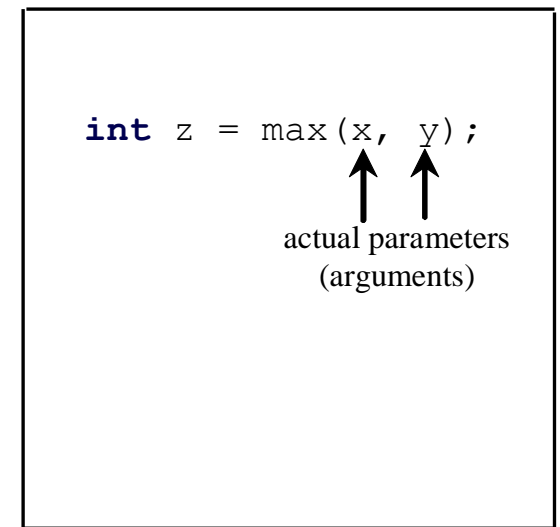actual parameters
(arguments)

Defining Methods

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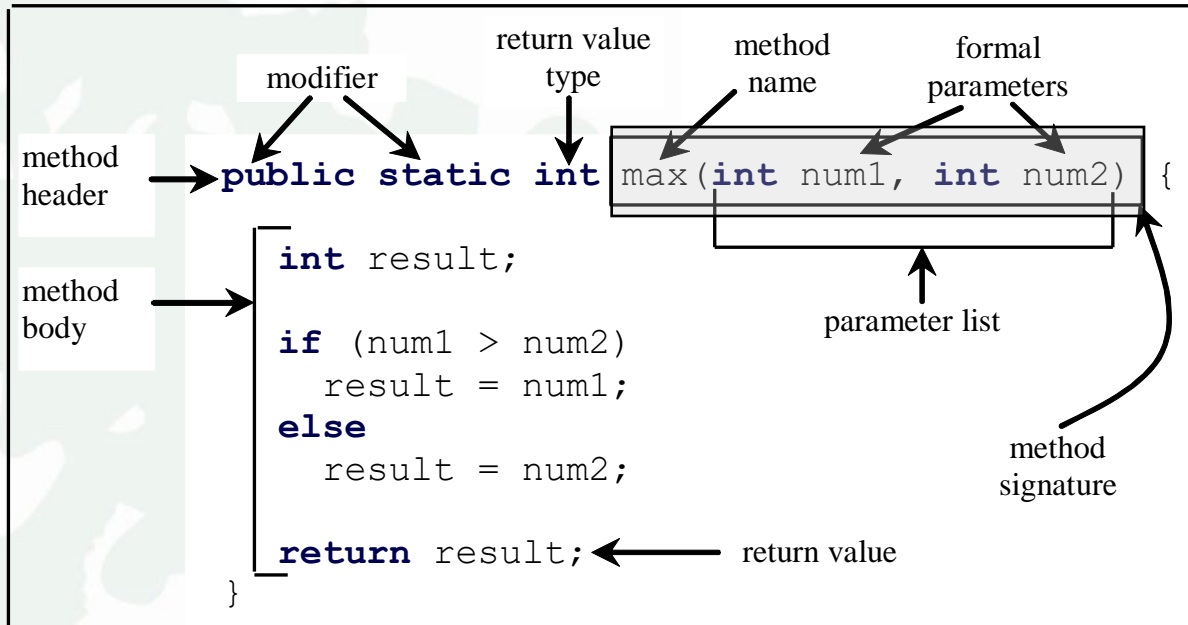
Define a method

Invoke a method

```
                    return value    method      formal
         modifier      type         name       parameters

method
header  →  public static int  max(int num1, int num2)  {

method
body       int result;

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

           return result;  ←  return value
         }
```

parameter list

```
int z = max(x, y);

            actual parameters
              (arguments)
```

7

Method Signature

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

Define a method

method header	**public static int** max(**int** num1, **int** num2) {
	modifier — return value type — method name — formal parameters — parameter list — method signature

method
body
	int result;
	
	  if (num1 > num2)
	    result = num1;
	  else
	    result = num2;
	
	  return result; ← return value
	}

Invoke a method

int z = max(x, y);

actual parameters
(arguments)

BIRZEIT UNIVERSITY

Calling Methods, cont.

pass the value of i

pass the value of j

```java
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);
}
```

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

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

  return result;
}
```

9

Trace Method Invocation

i is now 5

```java
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);
}
```

```java
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.

```java
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 →

```java
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 <u>return</u> statement to be reached regardless of how the <u>if</u> statement is evaluated.

Trace Call Stack

animation

> i 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;
}
```

i: 5

The main method
is invoked.

Trace Call Stack

BIRZEIT UNIVERSITY

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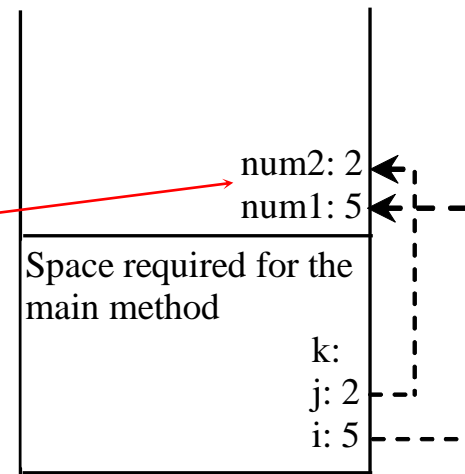
```
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

num2: 2
num1: 5

Space required for the
main method

k:
j: 2
i: 5

The max method is
invoked.

Trace Call Stack

BIRZEIT UNIVERSITY

Return result and assign it to k

```java
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.

14

Trace Call Stack

Execute print statement

```java
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);
}
```

```java
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.

Passing Parameters

```java
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");
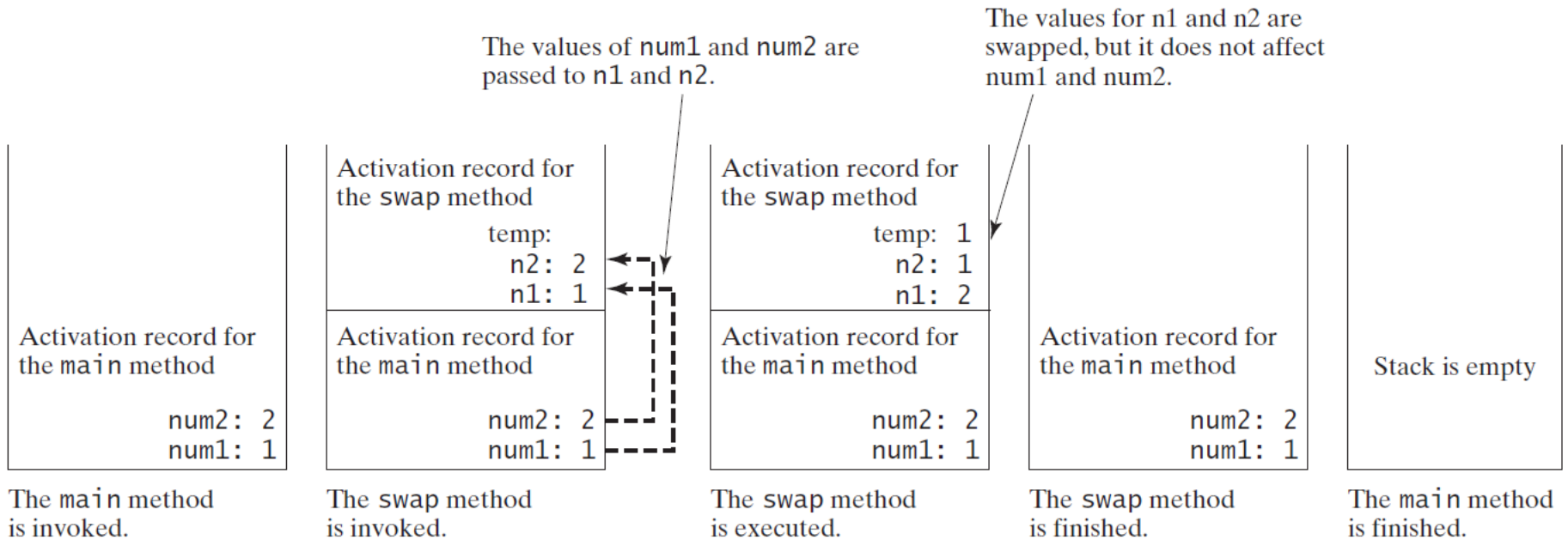
Pass by Value

This program demonstrates passing values to the methods.

```java
public static void swap(int n1, int n2) {
    System.out.println("\tInside the swap method");
    System.out.println("\t\tBefore swapping, n1 is " + n1 +
    " and n2 is " + n2);
    // Swap n1 with n2
    int temp = n1; n1 = n2;
    n2 = temp;
    System.out.println("\t\tAfter swapping, n1 is " + n1 + " and n2 is " + n2);
}
```

Run

TestPassByValue

Pass by Value, cont.

The values of num1 and num2 are passed to n1 and n2.

The values for n1 and n2 are swapped, but it does not affect num1 and num2.

Activation record for the swap method

temp:
n2: 2
n1: 1

Activation record for the main method

num2: 2
num1: 1

Activation record for the swap method

temp: 1
n2: 1
n1: 2

Activation record for the main method

num2: 2
num1: 1

Activation record for the main method

num2: 2
num1: 1

Activation record for the main method

num2: 2
num1: 1

Activation record for the main method

num2: 2
num1: 1

Stack is empty

The main method is invoked.

The swap method is invoked.

The swap method is executed.

The swap method is finished.

The main method is finished.

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.

```java
public static int gcd(int n1, int n2) {
        int gcd = 1; // Initial gcd is 1
        int k = 1; // Possible gcd
        while (k <= n1 && k <= n2) {
        if (n1 % k == 0 && n2 % k == 0)
                gcd = k; // Update gcd k++;
        }
        return gcd; // Return gcd
}

public static boolean isPrime(int number) {
        for (int divisor = 2; divisor <= number / 2; divisor++) {
                if (number % divisor == 0) {
                        return false;
                }
        }
        return true; // number is prime
}
```

Overloading Methods

Overloading methods enable you to define the methods with the same name as long as their parameter lists are different.

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


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


public static double max(double num1, double num2, double num3) {
        return max(max(num1, num2), num3);
}
```

TestMethodOverloading

Ambiguous Invocation: Error

```java
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.

Scope of Local Variables, cont.

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

```java
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

```java
  public static void method2() {

    int i = 1;
    int sum = 0;

    for (int i = 1; i < 10; i++)
      sum += i;
  }
}
```

Scope of Local Variables, cont.

```java
// 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;
    }
}
```
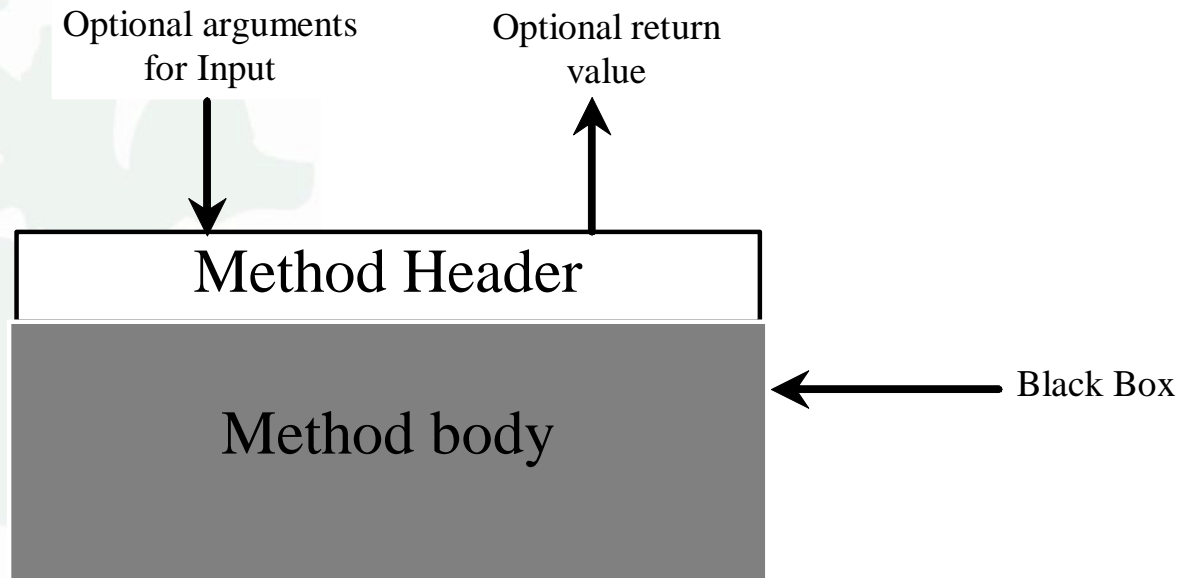
Scope of Local Variables, cont.

```java
// 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.

Optional arguments
for Input

Optional return
value

Method Header

Method body

Black Box

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:

    (char)(ch1 + Math.random() * (ch2 – ch1 + 1))

The RandomCharacter Class

```java
// 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

28

PrintCalender Case Study

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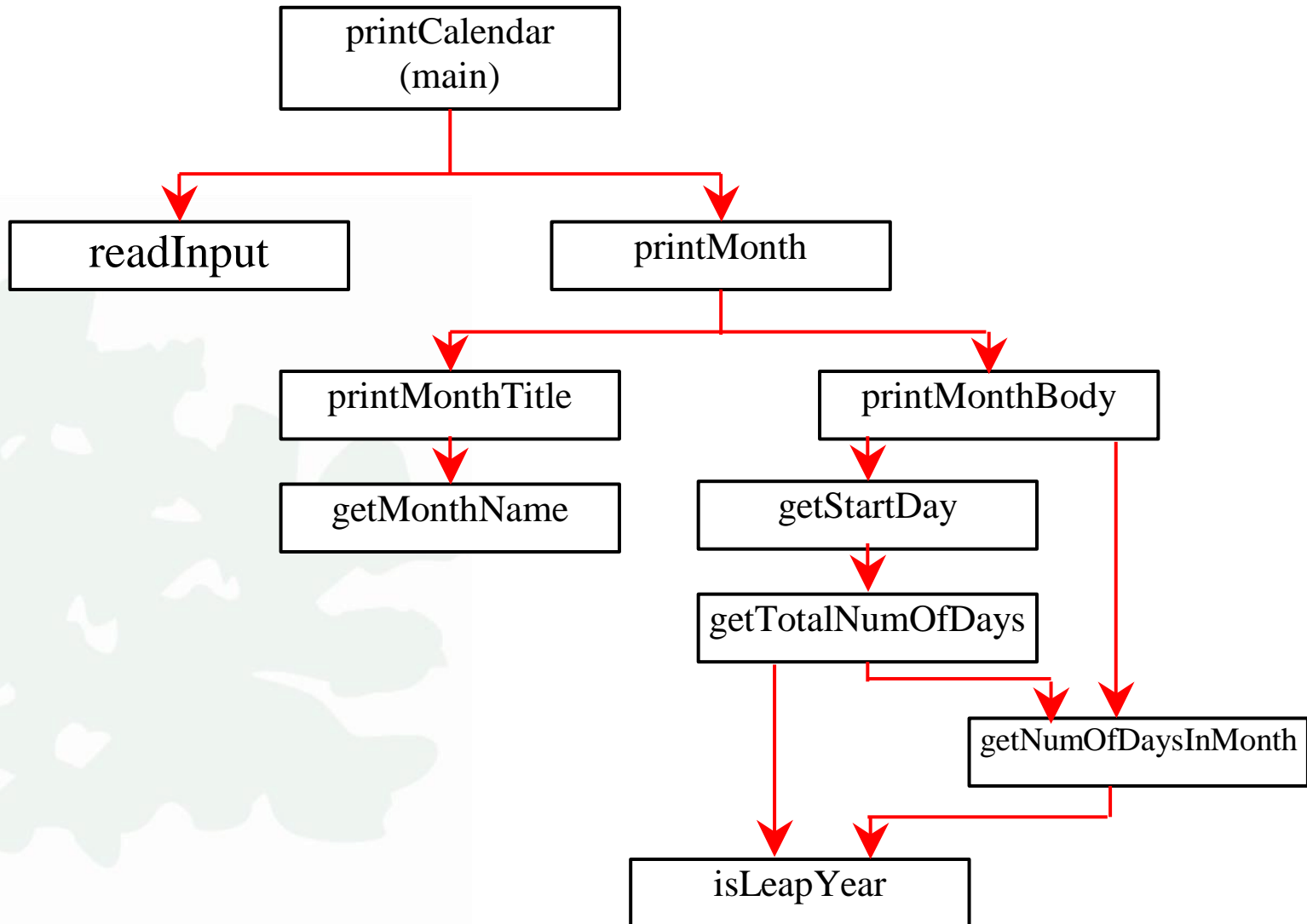

PrintCalendar   Run

Design Diagram

```java
public static boolean isLeapYear(int year) {
    return year % 400 == 0 || (year % 4 == 0 && year % 100 != 0);
}

public static int getNumberOfDaysInMonth(int year, int month) {
    if (month == 1 || month == 3 || month == 5 || month == 7 ||
    month == 8 || month == 10 || month == 12)
        return 31;
    if (month == 4 || month == 6 || month == 9 || month == 11)
        return 30;
    if (month == 2)
        return isLeapYear(year) ? 29 : 28; return 0;
}
```

```java
public static int getTotalNumberOfDays(int year, int month) {
        int total = 0; // Get the total days from 1800 to 1/1/year

        for (int i = 1800; i < year; i++)
                if (isLeapYear(i))
                        total = total + 366;
                else total = total + 365;

        // Add days from Jan to the month prior to the calendar month
        for (int i = 1; i < month; i++)
                total = total + getNumberOfDaysInMonth(year, i);
        return total;

}
```

```java
public static int getStartDay(int year, int month) {
        final int START_DAY_FOR_JAN_1_1800 = 3;

        // Get total number of days from 1/1/1800 to month/1/year
        int totalNumberOfDays = getTotalNumberOfDays(year, month);

        // Return the start day for month/1/year
        return (totalNumberOfDays + START_DAY_FOR_JAN_1_1800) % 7;
}
```

```java
/** Print month body */
public static void printMonthBody(int year, int month) {

        // Get start day of the week for the first date in the month
        int startDay = getStartDay(year, month);

        // Get number of days in the month
        int numberOfDaysInMonth = getNumberOfDaysInMonth(year, month);

        // Pad space before the first day of the month
        int i = 0;
        for (i = 0; i < startDay; i++)
                System.out.print(" ");
        for (i = 1; i <= numberOfDaysInMonth; i++) {
                System.out.printf("%4d", i);
                if ((i + startDay) % 7 == 0) System.out.println();
        }
        System.out.println();
}
```

```java
public static String getMonthName(int month) {
        String monthName = "";
        switch (month) {
                case 1: monthName = "January"; break;
                case 2: monthName = "February"; break;
                case 3: monthName = "March"; break;
                case 4: monthName = "April"; break;
                case 5: monthName = "May"; break;
                case 6: monthName = "June"; break;
                case 7: monthName = "July"; break;
                case 8: monthName = "August"; break;
                case 9: monthName = "September"; break;
                case 10: monthName = "October"; break;
                case 11: monthName = "November"; break;
                case 12: monthName = "December"; }
        return monthName;
}
```

```java
public static void printMonth(int year, int month) {
        // Print the headings of the calendar
        printMonthTitle(year, month);

        // Print the body of the calendar
        printMonthBody(year, month);
}

/** Print the month title, e.g., May, 1999 */
public static void printMonthTitle(int year, int month) {
        System.out.println("  " + getMonthName(month) + " " + year);
        System.out.println("-----------------------------");
        System.out.println(" Sun Mon Tue Wed Thu Fri Sat");
}
```