CIS2571 – Intro to Java

Chapter 5 → Methods

Information consolidated from a variety of textbook and online resources

Topic Objectives

- Identify the benefits of using methods
- Understand how data is transferred between methods
  - Parameters
  - Value returning and void methods
- Recognize the components of a method definition
- Understand how to invoke methods
- Know how the call stack changes when a method is invoked
- Understand how methods can be overloaded
- Know how variable scope affects variable use
- Recognize the different static constants and methods of the Math class

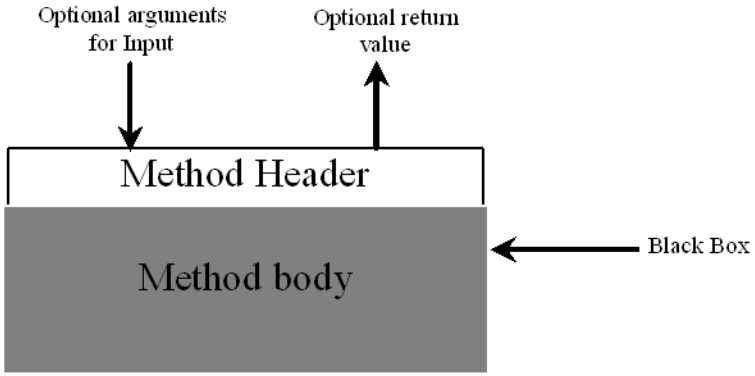
Methods

- Group of statements executed repeatedly can be represented by a **loop**
- Group of statements that
  - accomplish a single objective,
  - can be used multiple times, and
  - differ only by a set of values

  can be represented by a **method**
- Option to return, or not return, a value to the calling statement

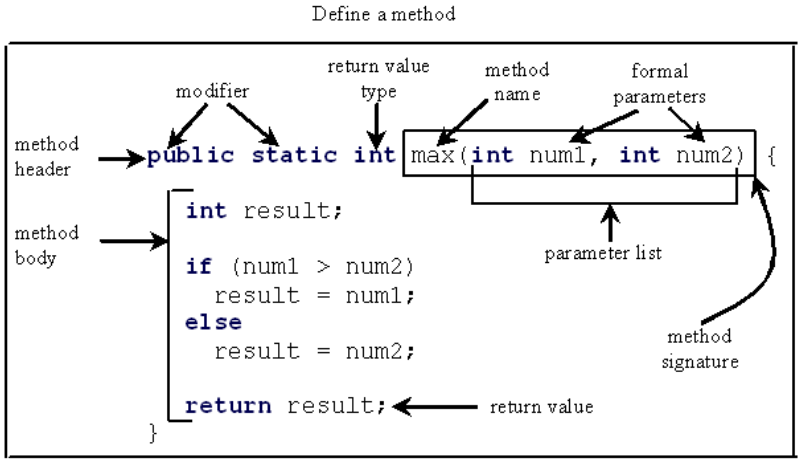
Method Benefits

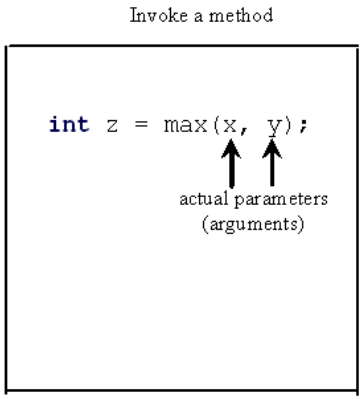
- Minimize program redundancy
- Maximize code reusability
  - Write once, reuse anywhere
- Information hiding (method abstraction)
  - Separate details from use
- Minimize complexity
  - Design
    - Stepwise refinement (divide and conquer)
    - Top-down design
  - Implementation
    - Top-down or bottom-up implementation
    - Easier to write, test, and debug



Methods

- Method header includes modifiers, return value type, method name, and parameters
- Method signature is combination of method name and parameter list
- Method body contains collection of statements that define what method does



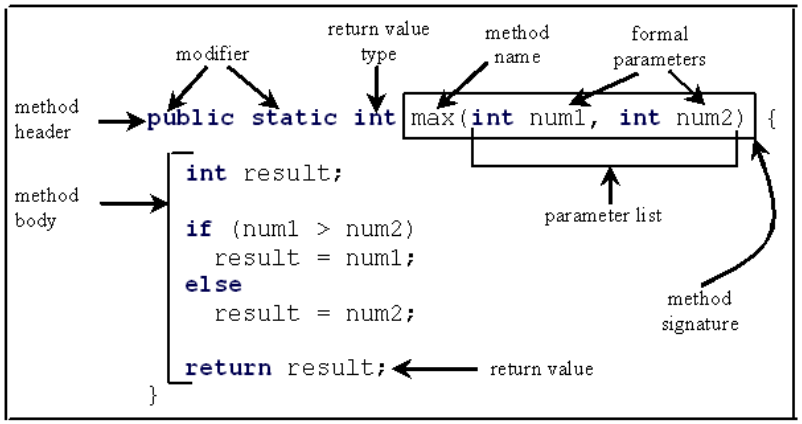


See 5.1 TestMax.java

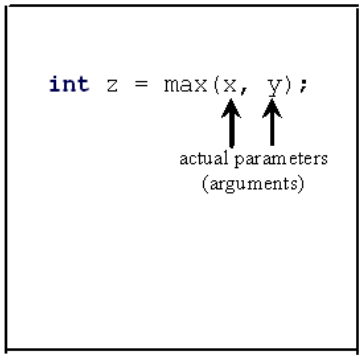
Methods

- Variables defined in method header are known as formal parameters
- When method is **invoked (or called)**, the actual parameter or argument is the value passed to the formal parameter
- Return value type is data type of value the method returns
  - Value returning method invoked as part of larger statement
  - Void (non-value returning) method invoked as standalone statement

Define a method



Invoke a method

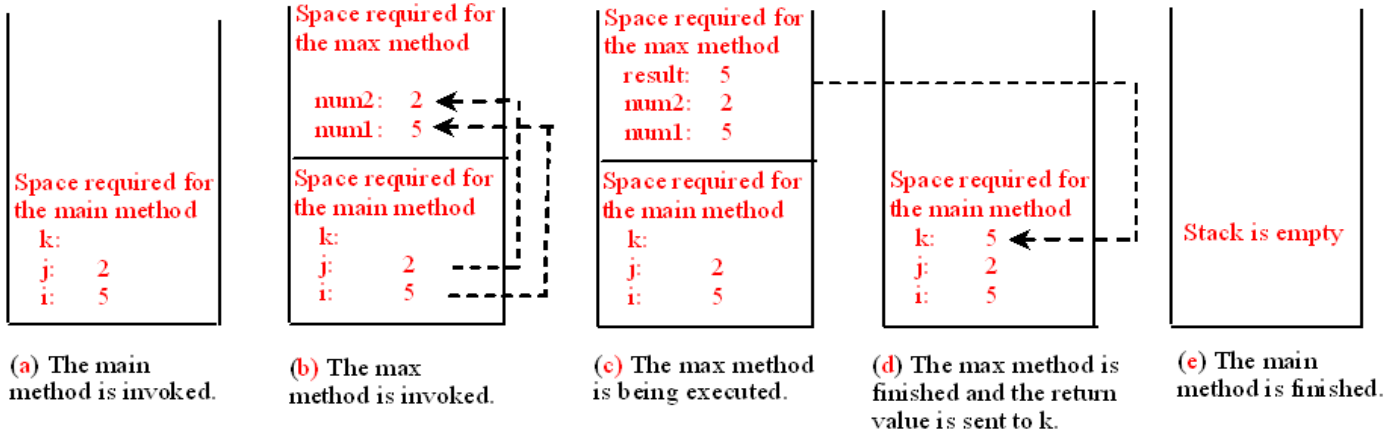


See 5.2 TestVoidMethod.java
See 5.3 TestReturnGradeMethod.java

Call Stack

- Every time method is invoked, parameters and variables are placed on the call stack in LIFO (last-in-first-out) order
  - Stack frame (or activation record) for a method contains:
    - Local variables
    - Return address of calling method
    - Parameters passed to method
- When method finishes and returns to caller, stack frame space is released

```
int i = 5, j = 2;
int k = max(i, j);
```



(a) The main method is invoked. (b) The max method is invoked. (c) The max method is being executed. (d) The max method is finished and the return value is sent to k. (e) The main method is finished.

Passing Parameters

- Arguments provided in method call must be in same order as method signature (*parameter order association*)
- Pass by Value → value of argument is assigned to formal parameter
  - Initialized local variable
  - Any changes to formal parameter do not affect calling argument

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

```
int i = 5, j = 2;
int k = max(i, j);
```

See 5.5 TestPassByValue.java

Method Overloading

- Two or more methods have same name but different parameter lists within same class
  - Can make programs clearer and more readable
  - Different modifiers or return types alone are **not** considered overloading

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

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

Method Overloading

- Compiler determines which method is invoked based upon method signature
  - Chooses most specific method for given arguments

```
int i = 5, j = 2;
int k = max(i, j);
```

```
double i = 5.0, j = 2.0;
double k = max(i, j);
```

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

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

See 5.9 TestMethodOverloading.java

Method Overloading

- Ambiguous invocation (two or more possible matches) causes compilation error
  - int can be promoted to double

```
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;
}
```

See AmbiguousOverloading.java

Local Variables

- Scope → part of program where variable can be referenced
  - Begins at variable declaration
  - Continues to end of block containing variable

```
import javax.swing.JOptionPane;

public class ScopeTest {
   public static void main(String[] args) {
      double sum = 0;
      int answer;
      do {
         double val = Math.random();
         sum += val;
         answer = JOptionPane.showConfirmDialog(null,
            "Do you want to continue?");
      } while (answer == JOptionPane.YES_OPTION);
      JOptionPane.showMessageDialog(null, "sum = " + sum);
   }
}
```

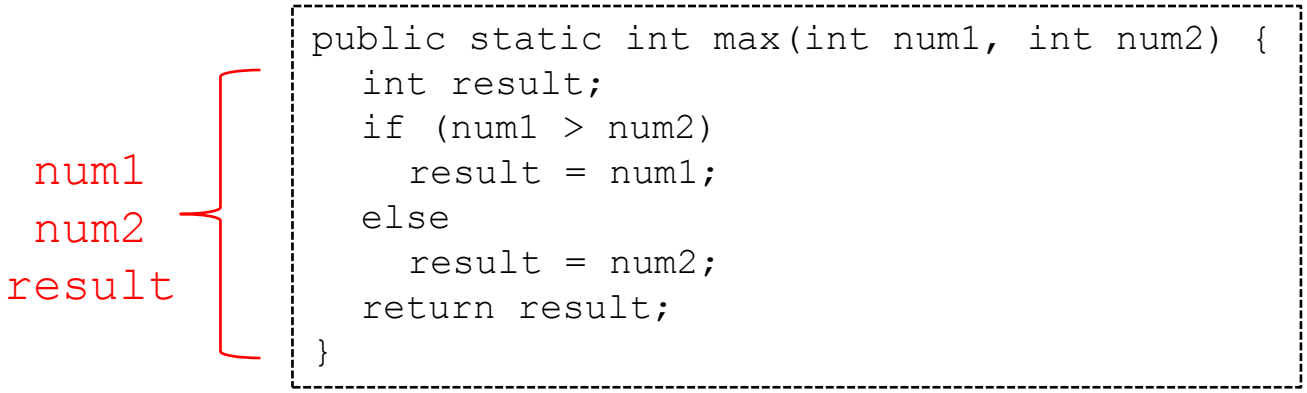
answer
sum
val

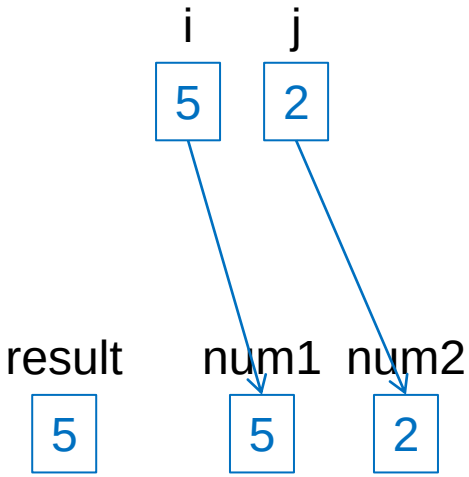
See ScopeTest.java

Local Variables

- Local variable is defined within a method
  - Formal parameter is a local variable for entire method
    - Initialized with argument value when called

```
int i = 5, j = 2;
int k = max(i, j);
```





Local Variables

- Cannot declare local variable twice in same (or nested) blocks
- Variable declared in initial action of for loop has its scope for the entire loop

error to declare i in two nested blocks

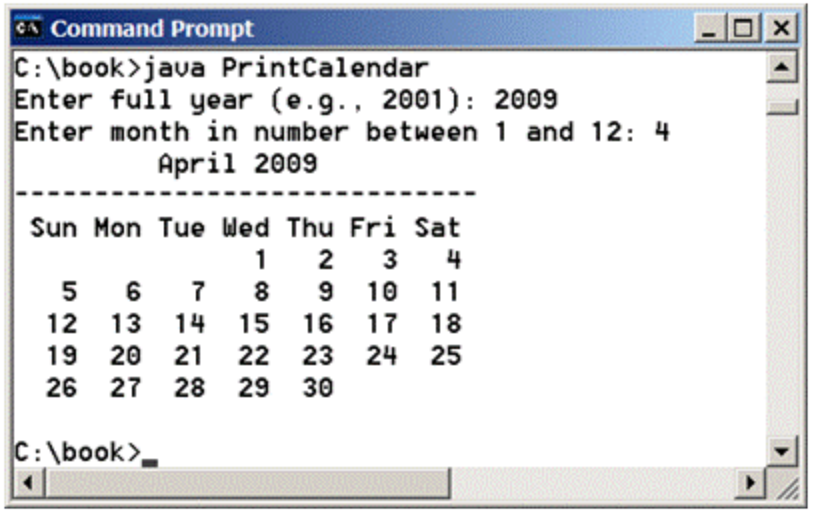
```
public static void myMethod() {
   int i = 1;
   int sum = 0;
   . . .
   for (int i = 1; i < 10; i++) {
      sum += i;
   }
   . . .
}
```

Math Class

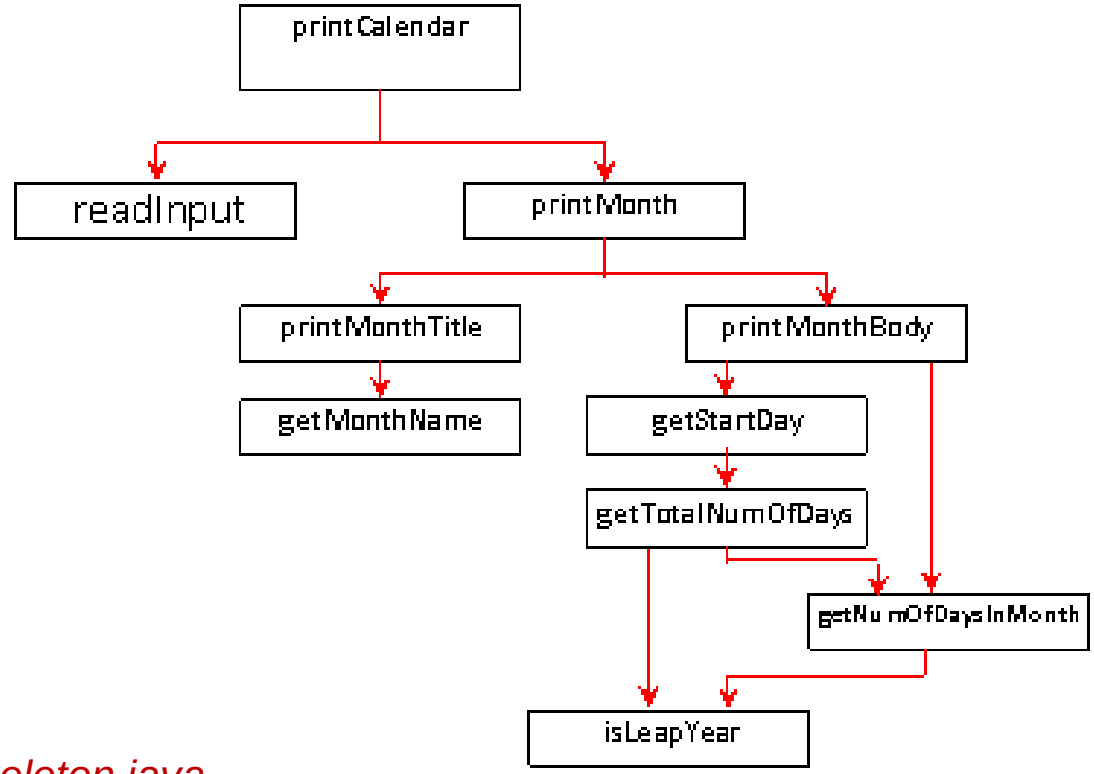
- In java.lang package
  - Automatically imported
- Used to perform basic mathematical functions
- Static class constants:
  - Math.PI
  - Math.E
- Static class methods (invoked with *Math.methodName*):
  - Trigonometric
    - sin, cos, tan, toRadians, toDegrees, asign, acos, atan
  - Exponential
    - exp, log, log10, pow, sqrt
  - Rounding
    - ceil, floor, rint, round
  - min, max, abs, random

Method Design and Implementation

- Minimize complexity
  - Design
    - Stepwise refinement (divide and conquer)
    - Top-down design
  - Implementation
    - Top-down or bottom-up implementation
    - Easier to write, test, and debug
- Example → PrintCalendar



Top-Down Design



See PrintCalendarSkeleton.java
See 5.12 PrintCalendar.java