Programming in the Large I: Methods (Subroutines)

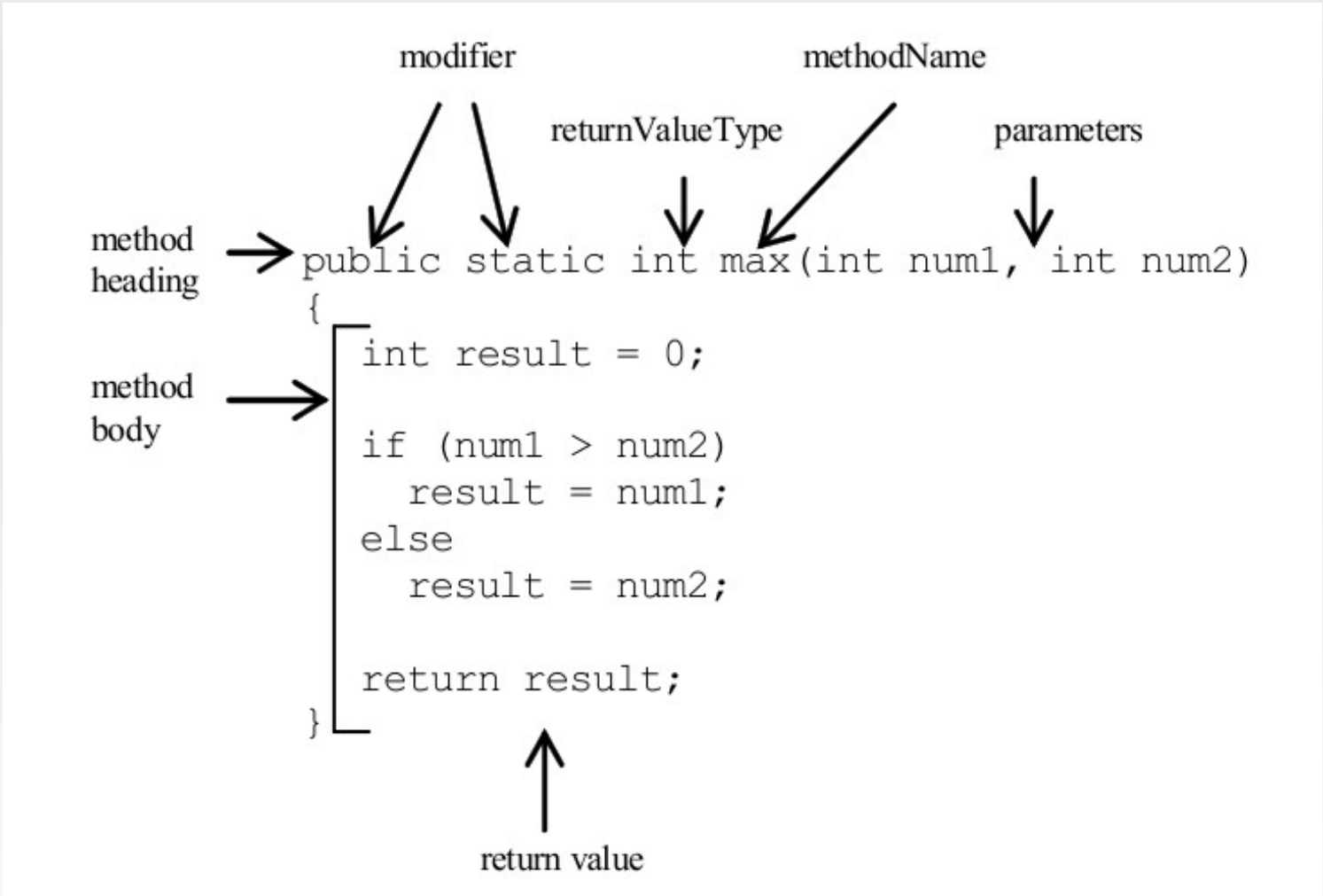
188230 Advanced Computer Programming

Asst. Prof. Dr. Kanda Runapongsa Saikaew
(krunapon@kku.ac.th)
Department of Computer Engineering
Khon Kaen University

Agenda

- Introducing Methods
- Method Abstraction
- Declaring vs. Defining Methods
- Calling Methods
- Passing Parameters
- Overloading Methods
- Recursion
- APIs, Packages, and Javadoc

Introducing Methods



Why Do We Need a Method?

- One way to break up a complex program into manageable pieces is to use methods
- A method consists of the instructions for carrying out a certain task, grouped together and given a name
- Whenever the computer encounters a method name, it executes all the instructions necessary to carry out the task associated with that method.

Methods Usage

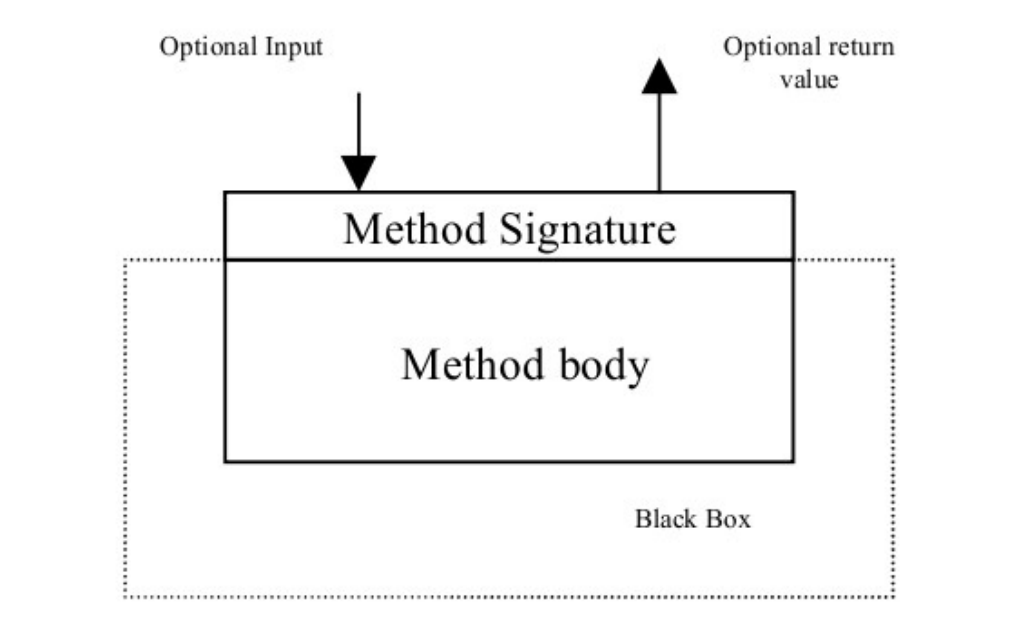
- Methods can be used over and over, at different places in the program
- A method can even be used inside another method
- This allows you to write simple methods and then use them to help write more complex methods
- A very complex program can be built up step-by-step, where each step in the construction is reasonably simple

Method Types

- Methods in Java can be either static or non-static
- This slide covers static methods only
  - Static methods are belong to the class
  - They have the keyword “static” at method header
- Non-static methods which are used in true object-oriented programming, will be covered in the next slide

Method Abstraction

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



Black Boxes

- A physical black box might have
  - Buttons on the outside that you can push
  - Dials that you can set
  - Slots that can be used for passing information back and forth
- A black box needs some kind of interface with the rest of the program
  - Allows some interaction between what's inside the box and what's outside

Interface and Implementation

- The interface of a black box should be fairly straight-forward, well-defined, and easy to understand
  - TV remote control which allows you to turn on/off TV, change channels, and adjust the volumes
  - You don't need to know how TV works
- The inside of a black box is called its implementation
- You shouldn’t need to know anything about its implementation; all you need to know is its interface

Static Methods

- A static method must be defined in a class body
  - All of your defining and declaring methods must be between class {..... and }
- A static method can call only static methods
  - Example: main method which is a static method can only call methods that are declared as static
- A static method cannot call non-static methods
  - Example: main method cannot call methods that are not declared as static

Example

```
public class StaticMethodsDemo {
  static void hello() {
      System.out.println("Hello");
  }
  void bye() {
    System.out.println("Bye");
  }
  public static void main(String[] args) {
     hello();
    bye();
  }
}
```

Static Members

- Static members are static class variables
- They are declared outside every method in the class
- Static methods can only use static class members or their local variables
- If we want to declare constant variables, we need to use keyword final
- We have seen static members of some classes such as Math.PI

Examples of Static Members

```
public class StaticMembersDemo {
static final String COURSE_ID = "188230";
static String courseLocation = "4309";
String grade;
public static void main(String[] args) {
System.out.println("This course is " + COURSE_ID);
System.out.println("It takes place at " + courseLocation);
System.out.println("Grade is" + grade);
}
}
```

Declaring vs. Defining Methods

- Declaring a method

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

- Defining a method

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

Calling a Method

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

Passing Parameters

- In Java, the parameters are passed by values
- Thus, the values that are changed in the method will not be changed outside that method
- What should we do if we want to swap two numbers?
- Before: m = 2, n = 3
- After: m = 3, n = 2

swap Method

```
static void swap(int m, int n) {
      System.out.println("Inside swap method");
       System.out.println("Before swapping m is " + m + " n is " + n);
       // Swapping m with n
       int temp = m;
       m = n;
       n = temp;
      System.out.println("After swapping m is " + m + " n is " + n);
}
```

Calling swap Method

```
public static void main(String[] args) {
   int m = 2;
   int n = 3;
   System.out.println("Before invoking the swap
   method: m is " + m + " n is " + n);
   swap(m,n);
   System.out.println("After invoking the swap
   method: m is " + m + " n is " + n);
}
```

Overloading Methods

- A method signature includes its name, the number of parameters, the return type, and the type of each parameter
- Java allows different methods in the same class to have the same name, provided that their signatures (return types and parameters types) are different.
- Examples:
  - int max(int n1, int n2);
  - double max(double n1, double n2);

Overloading max Methods

```
static int max(int num1, int num2) {
if (num1 > num2) return num1; else return num2;
}
static double max(double num1, double num2) {
if (num1 > num2) return num1; else return num2;
}
static double max(double num1, double num2, double num3) {
return max(max(num1, num2), num3);
}
```

Calling overloaded methods

```
public static void main(String[] args) {
int m = 2, n = 3;
double a = 2.2, b = 3.3, c = 4.4;
System.out.println("Max of " + m + " " + n + " is " +
max(m,n));
System.out.println("Max of " + a + " " + b + " is " +
max(a,b));
System.out.println("Max of " + a + " " + b + " " + c +
" is " + max(a,b,c));
}
```

Recursive Methods

- A method that calls itself is a recursive method
- The usage of recursive methods are found often when we compute Maths formulas
  - factorial(n) = n*factorial(n-1)
    - factorial(0) = 1
  - fib(n) = fib(n-2) + fib(n-1) for n >= 2
    - fib(0) =1
    - fib(1) = 1

Method factorial

```
static long factorial(int n) {
    if (n == 0) // stopping condition
       return 1;
    else
      return n*factorial(n-1);  // call factorial recursively
}
```

Calling Method factorial

```
public static void main(String[] args) {
  int m = 3, n = 5;
  System.out.println("factorial(" + m + ") = " +
   factorial(m));
 System.out.println("factorial(" + n + ") = " +
   factorial(n));
}
```

- What is the output?

Method fibonacci

```
static long fib(int n) {
    if ((n == 0) || (n == 1)) {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
}
```

Calling Method fibnoacci

```
public static void main(String[] args) {
int m = 3, n = 5;
System.out.println("fib(" + m + ") = " + fib(m));
System.out.println("fib(" + n + ") = " + fib(n));
}
```

- What is the output?

API

- Every programming project involves a mixture of innovation and reuse of existing tools
- A software toolbox is a kind of black box, and it presents a certain interface to the programmer
- Such interface is called Applications Programming Interface (API)
- The Java programming language is supplemented by a large, standard API
  - Math.sqrt(), System.out.println()

Java's Standards Package

- The entire standard Java API is implemented in several packages
- One of these, which is named “java”, contains several non-GUI packages as well as the original AWT graphics user interface classes.
- Another package, “javax”, was added in Java version 1.2 and contains the classes used by the Swing graphical user interface and other additions to the API

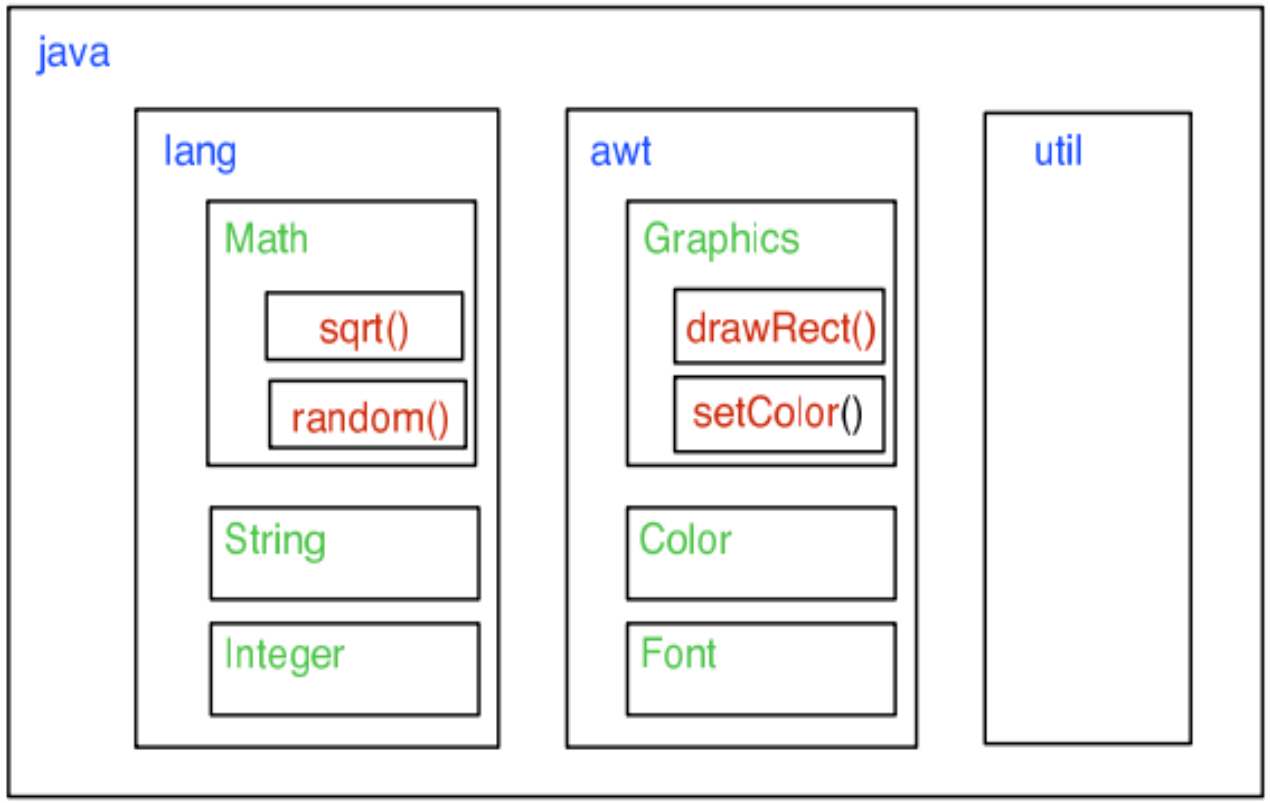
Java Package Structure & Names

- A package can contain both classes and other packages
- A package that is contained in another package is sometimes called a “sub-package.”
- Both the java package and the javax package contain sub-packages
- One of the sub-packages of java, for example, is called “lang”
  - Thus the full name of package lang is java.lang

Java Package Structure & Names

- java.lang contains a number of classes that form the foundation for all Java programming
- Inside java.lang package, there are many classes that we have used, such as String and Math
  - The full name of class String is java.lang.String
  - The full name of class Math is java.lang.Math

Java Packages, Classes, and Methods

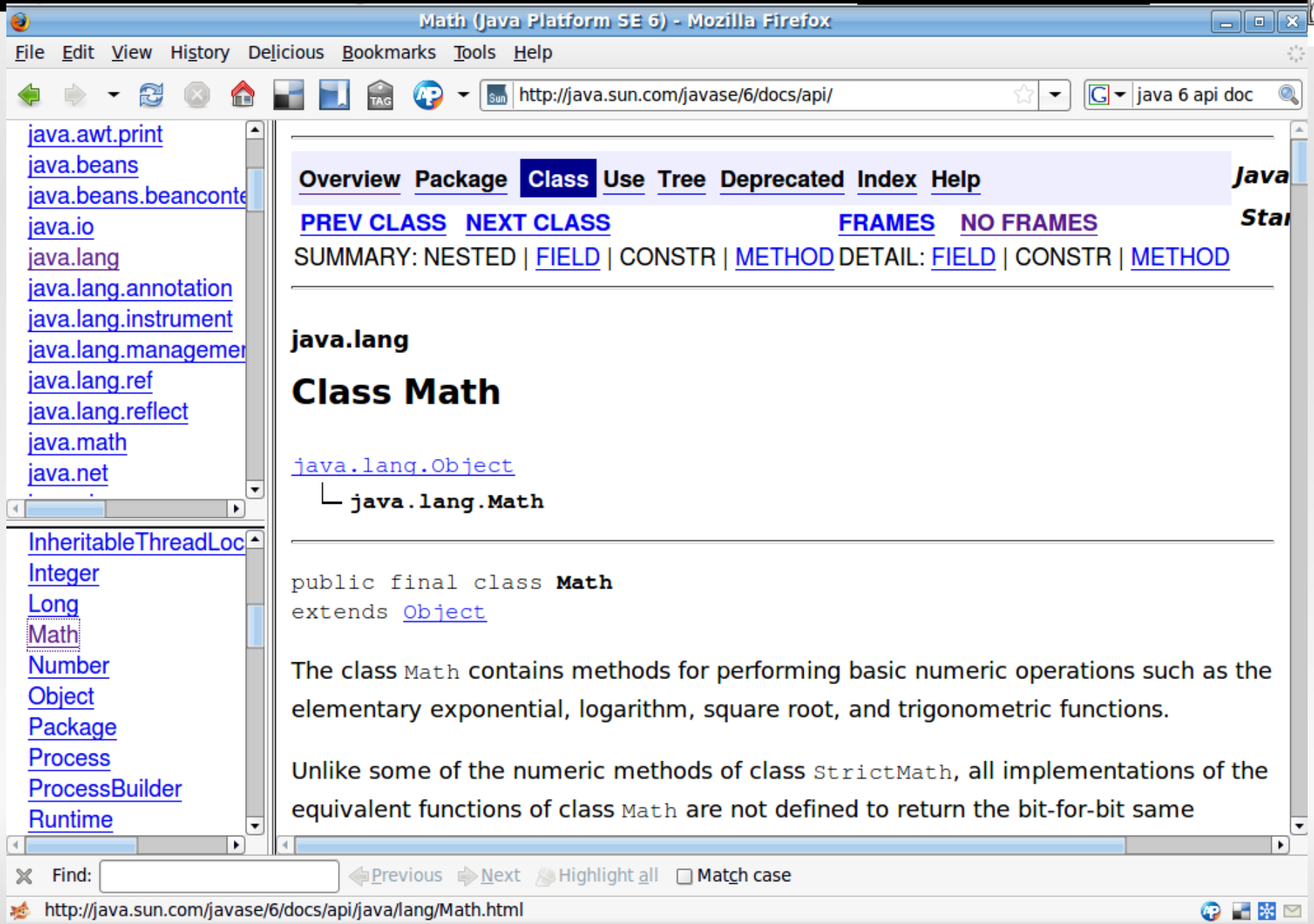


Subroutines nested in classes nested in two layers of packages.
The full name of sqrt() is java.lang.Math.sqrt()

Java API Doc

- View http://java.sun.com/javase/6/docs/api/
- Download http://java.sun.com/javase/downloads/index.jsp#docs
- The API will tell you about all Java packages, Java classes in each package, static members and methods in each class as well as non-static methods
- It's a tool that Java programmers must learn how to use it

The Math Class



Using Classes from Packages

- Let's say that you want to use the class java.util.Scanner in a program that you are writing
- Like any class, java.util.Scanner is a type, which means that you can use it to declare variables
- One way to do this is to use the full name of the class as the name of the type
  - java.util.Scanner inputScan;

Using Classes from Packages

- Using the full name of every class can get tiresome
- Java makes it possible to avoid using the full name of a class by importing the class
- If you put import java.util.Scanner at the beginning of a Java file, then you can just use the class name Scanner
  - import java.util.Scanner;
  - ...
  - Scanner inputScan;

import Keyword

- Note that the only effect of the import directive is to allow you to use simple class names instead of full “package.class” names
  - You aren’t really importing anything
- There is a shortcut for importing all the classes from a given package. You can import all the classes from java.util by saying

```
import java.util.*;
```

- The “*” is a wildcard that matches every class in the package

The usage of * with import

- The * does not match sub-packages
- You cannot import the entire contents of all the sub-packages of the java package by saying import java.*
- If we import a few classes in a package, we may want to explicitly tell the class name
- But if we import many classes in a package, we may want to use the wildcard *

The package java.lang

- Because the package java.lang is so fundamental, all the classes in java.lang are automatically imported into every program
- It's as if every program began with the statement "import java.lang.*;". This is why we have been able to use the class name String instead of java.lang.String, and Math.sqrt() instead of java.lang.Math.sqrt()
- It would still, however, be perfectly legal to use the longer forms of the names

Javadoc

- To use an API effectively, you need good documentation for it
- The documentation for most Java APIs is prepared using a system called Javadoc
- Javadoc documentation is prepared from special comments that are placed in the Java source code in the form
  - /** .... */

Javadoc Tool

- Like any comment, a Javadoc comment is ignored by the computer when the file is compiled.
- But there is a tool called javadoc that
  - Reads Java source code files
  - Extracts any Javado comments that it finds
  - Creates a set of Web pages containing the comments in a nicely formatted, interlinked form

Javadoc Tool and Comments

- By default, javadoc will only collect information about public classes, subroutines, and member variables
  - But it allows the option of creating documentation for non-public things as well
- Javadoc comments can include doc tags, which are processed as commands by the javadoc tool
- A doc tag has a name that begins with the character @

Example of Java Codes with Javadoc Comments

```
/**
 * This subroutine computes the area of a rectangle, given its
*  width  and its height.
 * @param width the length of one side of the rectangle
 * @param height the length the second side of the rectangle
 * @return the area of the rectangle
 */
public static double areaOfRectangle( double width, double
    height ) {
   ...
```

JavadocDemo (1/3)

```
package coe.java.demos.c4;
/**
 * This progam is to illustrate the use of Java doc
 * @author Kanda Saikaew
 * @version 1.0, 06/25/09
 *
 */
public class JavadocDemo {
```

JavadocDemo (2/3)

```
/**
 * This method adds two integers
 * @param m the first integer
 * @param n the second integer
 * @return result the addition of m and n
 */
public static int add(int m, int n) {
int result = m +n;
return result;
}
```

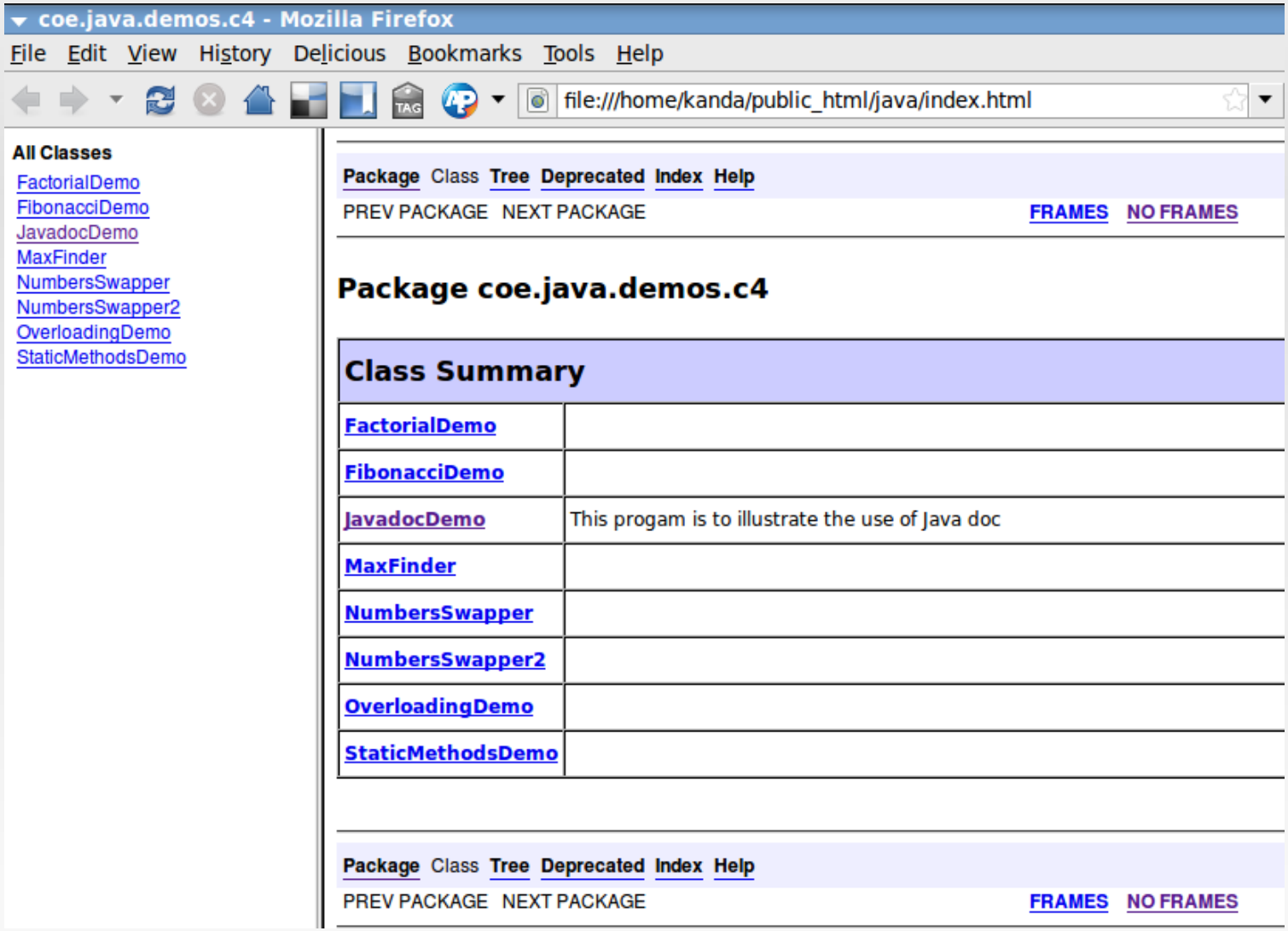
JavadocDemo (3/3)

```
/**
 * The main method of the program
 * @param args Program arguments
 */
public static void main(String[] args) {
System.out.println("A very simple program with Javadoc");
}
}
```

Sample Usage of javadoc

- To create Java Documentation for the whole package
  - javadoc -d [directory to contain HTML files] [package name]
  - Example:javadoc -d /home/kanda/public_html/java coe.java.demos.c4
- To create Java Documentation for a particular Java file
  - Javadoc -d [directory to contain HTML files] [file name]
  - Example:javadoc -d /home/kanda/public_html/java coe/java/demos/c4/JavadocDemo.java

Generated Java Documentation



References

- David J. Eck, ”Introduction to Programming Using Java”, Version 5.0, December 2006 http://math.hws.edu/javanotes/