Advance Computer Programming
SE241
Methods

static Methods, static Fields and Class Math

- To declare a method as static, place the keyword static before the return type in the method's declaration.
  - This method performs a task that does not depend on the contents of any object.
  - This method applies to the class in which it's declared as a whole .
- For any class imported into your program, you can call the class's static methods by specifying the name of the class in which the method is declared, followed by a dot (.) and the method name, as in

  ```
  ClassName.methodName( arguments )
  ```

static Methods, static Fields and Class Math

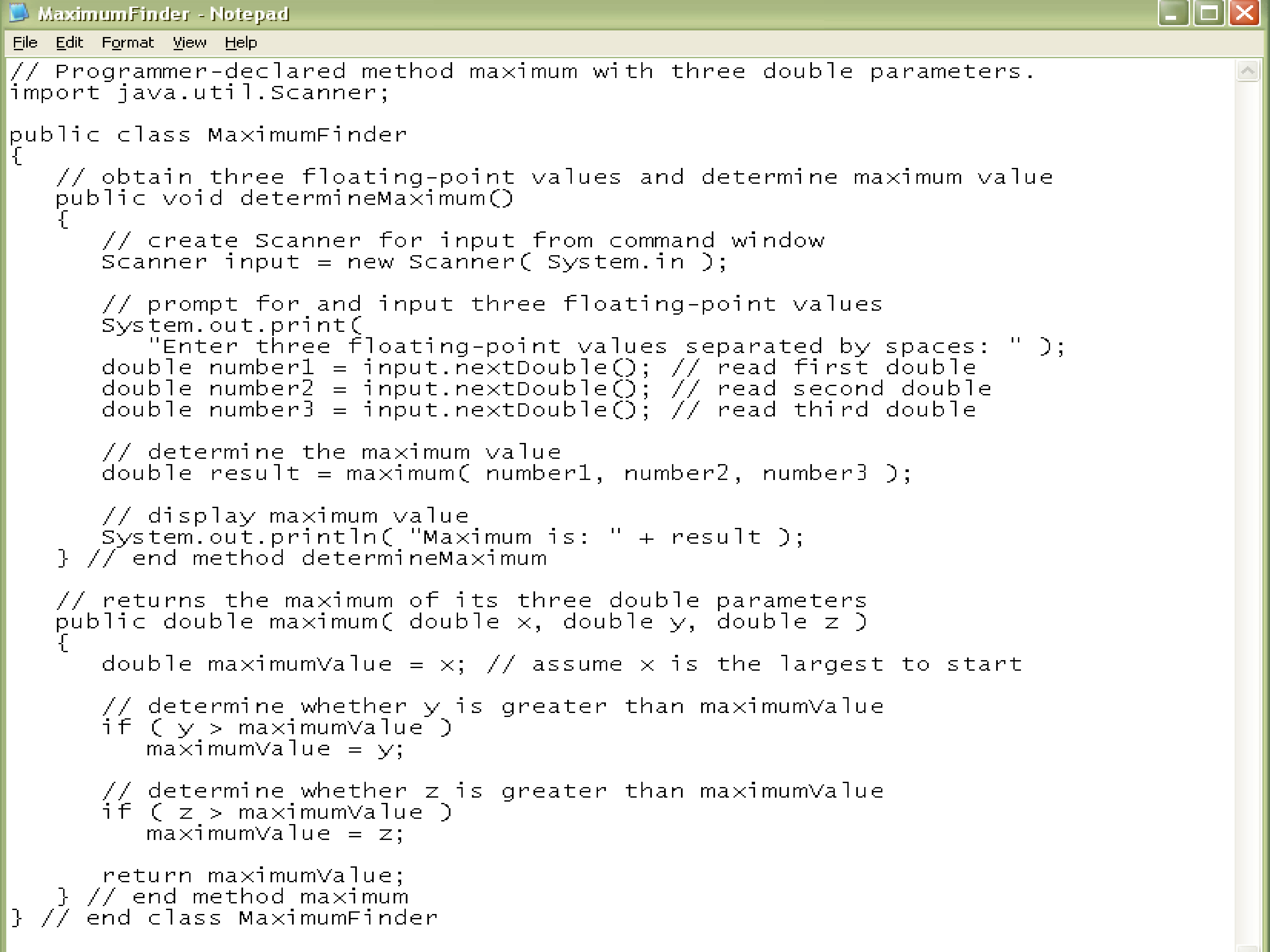
- Class Math provides a collection of static methods that enable you to perform common mathematical calculations.

```
    Math.sqrt( 900.0 )
    System.out.println( Math.sqrt( 900.0 ));
```

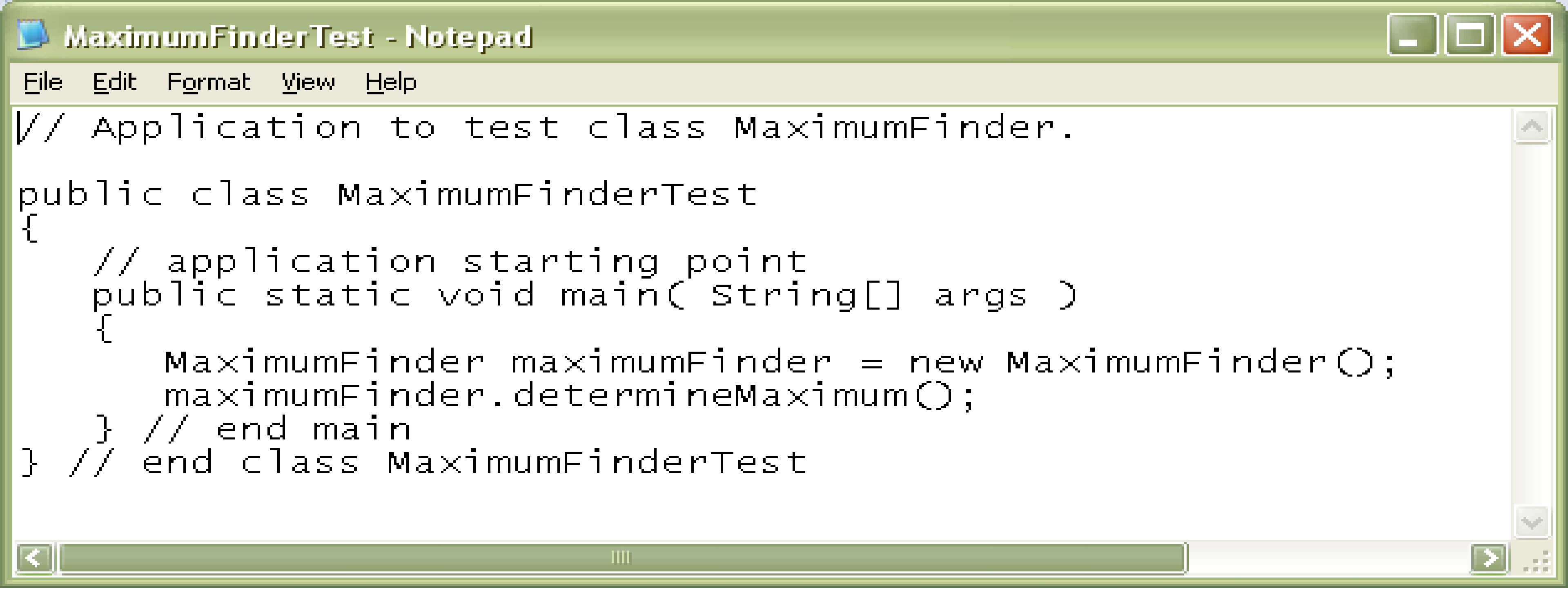
- In this statement, the value that sqrt returns becomes the argument to method println.
- There was **no need to create a Math object** before calling method sqrt.
- Also all **Math class methods are static**, therefore, each is called by preceding its name with the **class name Math and the dot (.) separator.**
- Class Math is **part of the java.lang package**, which is implicitly imported by the compiler, so it’s not necessary to import class Math to use its methods.
- **Method arguments may be constants, variables or expressions.**

static Methods, static Fields and Class Math

Method	Description	Example
`abs( x )`	absolute value of x	`abs( 23.7 )` is 23.7 `abs( 0.0 )` is 0.0 `abs( -23.7 )` is 23.7
`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 )` is 10.0 `ceil( -9.8 )` is -9.0
`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 )` is 1.0
`exp( x )`	exponential method e^x	`exp( 1.0 )` is 2.71828 `exp( 2.0 )` is 7.38906
`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 )` is 9.0 `floor( -9.8 )` is -10.0
`log( x )`	natural logarithm of x (base e)	`log( Math.E )` is 1.0 `log( Math.E * Math.E )` is 2.0
`max( x, y )`	larger value of x and y	`max( 2.3, 12.7 )` is 12.7 `max( -2.3, -12.7 )` is -2.3
`min( x, y )`	smaller value of x and y	`min( 2.3, 12.7 )` is 2.3 `min( -2.3, -12.7 )` is -12.7
`pow( x, y )`	x raised to the power y (i.e., x^y)	`pow( 2.0, 7.0 )` is 128.0 `pow( 9.0, 0.5 )` is 3.0
`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 )` is 0.0
`sqrt( x )`	square root of x	`sqrt( 900.0 )` is 30.0
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 )` is 0.0



```
// Programmer-declared method maximum with three double parameters.
import java.util.Scanner;

public class MaximumFinder
{
   // obtain three floating-point values and determine maximum value
   public void determineMaximum()
   {
      // create Scanner for input from command window
      Scanner input = new Scanner( System.in );

      // prompt for and input three floating-point values
      System.out.print(
         "Enter three floating-point values separated by spaces: " );
      double number1 = input.nextDouble(); // read first double
      double number2 = input.nextDouble(); // read second double
      double number3 = input.nextDouble(); // read third double

      // determine the maximum value
      double result = maximum( number1, number2, number3 );

      // display maximum value
      System.out.println( "Maximum is: " + result );
   } // end method determineMaximum

   // returns the maximum of its three double parameters
   public double maximum( double x, double y, double z )
   {
      double maximumValue = x; // assume x is the largest to start

      // determine whether y is greater than maximumValue
      if ( y > maximumValue )
         maximumValue = y;

      // determine whether z is greater than maximumValue
      if ( z > maximumValue )
         maximumValue = z;

      return maximumValue;
   } // end method maximum
} // end class MaximumFinder
```



```
// Application to test class MaximumFinder.

public class MaximumFinderTest
{
   // application starting point
   public static void main( String[] args )
   {
      MaximumFinder maximumFinder = new MaximumFinder();
      maximumFinder.determineMaximum();
   } // end main
} // end class MaximumFinderTest
```

```
Enter three floating-point values separated by spaces: 9.35 2.74 5.1
Maximum is: 9.35
```

```
Enter three floating-point values separated by spaces: 5.8 12.45 8.32
Maximum is: 12.45
```

```
Enter three floating-point values separated by spaces: 6.46 4.12 10.54
Maximum is: 10.54
```

Declaring Methods with Multiple Parameters

- There must be one argument in the method call for each parameter in the method declaration.
- Also, each argument must be consistent with the type of the corresponding parameter.
- Declaring method parameters of the same type as **float x, y instead of float x, float y is a syntax error,** a type is required for each parameter in the parameter list.
- By reusing Math.max, which finds the larger of two values. Note how .concise this code is

return Math.max(x, Math.max(y, z));

Declaring Methods with Multiple Parameters

Assembling Strings with String Concatenation

- When both operands of operator + are String objects, operator + creates a new String object in which the characters of the right operand are placed at the end of those in the left operand.
- For example, the expression "hello “ + "there“ creates the String "hello there".
- In above code, the expression “Maximum is: “ + result uses operator + with operands of types String and double.

Notes on Declaring and Using Methods

There are three ways to call a method:

1. Using a method name by itself to call an other method of the same class, such as **maximum (number1,number2,number3)** in above code.
2. Using a variable that contains a reference to an object, followed by a dot (.) and the method name to call a method of the referenced object, such as the method call ***maximumFinder.determineMaximum()***, which calls a method of class *MaximumFinder* from the main method of *MaximumFinderTest.*
3. Using the class name and a dot (.) to call a static method of a class, such as **Math.sqrt(900.0)** in the code.

Notes on Declaring and Using Methods …

- **A non-static method can call any method of the same class directly** (i.e., using the method name by itself) and can manipulate any of class’s fields directly.
- **A static method can call only other static methods of the same class directly and can manipulate only static variables in the same class directly.**
- To access the class’s non-static members, **a static method must use a reference to an object of the class.**
- Recall that static methods relate to a class as a whole, whereas non-static methods are associated with a specific instance (object) of the class and may manipulate the instance variables of that object.

Notes on Declaring and Using Methods …

There are two ways to return control to the statement that calls a method.

- If the method does not return a result, control returns when the program **flow reaches the method-ending right .**
- If the method returns a result, the following statement evaluates the expression, then returns the result to the caller.

  return expression;

Notes on Declaring and Using Methods …

Common Programming Error:

1. Declaring a method outside the body of a class declaration or inside the body of another method is a syntax error.
2. Omitting the return-value-type, possibly void, in a method declaration is a syntax error.
3. Placing a semicolon after the right parenthesis enclosing the parameter list of a method declaration is a syntax error.
4. Redeclaring a parameter as a local variable in the method's body is a compilation error.
5. Forgetting to return a value from a method that should return a value is a compilation error. If a return type other than void is specified, the method must contain a return statement that returns a value consistent with the method's return type. Returning a value from a method whose return type has been declared void is a compilation error.

Method-Call Stack and Activation Records

- A data structure (i.e., collection of related data items) known as a stack.
- For example, when a dish is placed on the pile, it’s normally placed at the top (referred to as pushing the dish onto the stack).
- Similarly, when a dish is removed from the pile, it’s always removed from the top (referred to as popping the dish off the stack).
- Stacks are known as last-in, first-out (LIFO) data structures, the last item pushed (inserted) on the stack is the first item popped (removed) from the stack.

Method-Call Stack and Activation Records …

- When a program calls a method, the called method must know how to return to its caller, so the return address of the calling method is pushed onto the program-execution stack (sometimes referred to as the method-call stack).

- If a series of method calls occurs, the successive return addresses are pushed onto the stack in last-in, first-out order so that each method can return to its caller.

Case Study: Random-Number Generation

- The element of chance can be introduced in a program via an object of class **Random** (package java.util) or via the static method **random** of **class Math**.
- Objects of class Random can produce random boolean, byte, float, double, int, long and Gaussian values.
- Math method random can produce only double values in the range $0.0 \le x < 1.0$.
- A new random-number generator object can be created as follows:

  Random randomNumbers = new Random();

  int randomValue = randomNumbers.nextInt();

Case Study: Random-Number Generation ...

- Random method **nextInt** generates a random int value in the range -2,147,483,648 to +2,147,483,647, inclusive.

- Class Random provides another version of method **nextInt** that receives an int argument and returns a value from 0 up to, but not including, the argument's value.
  - For example, for coin tossing, the following statement returns 0 or 1.

```
int randomValue = randomNumbers.nextInt(2 );
```

Case Study: Random-Number Generation …

```java
// Shifted and scaled random integers.
import java.util.Random; // program uses class Random

public class RandomIntegers
{
   public static void main( String[] args )
   {
      Random randomNumbers = new Random(); // random number generator
       int face; // stores each random integer generated

       // loop 20 times
       for ( int counter = 1; counter <= 20; counter++ )
       {
          // pick random integer from 1 to 6
          face = 1 + randomNumbers.nextInt( 6 );

          System.out.printf( "%d  ", face ); // display generated value

          // if counter is divisible by 5, start a new line of output
          if ( counter % 5 == 0 )
             System.out.println();
       } // end for
    } // end main
 } // end class RandomIntegers
```

```
1  5  3  6  2
5  2  6  5  2
4  4  4  2  6
3  1  6  2  2
```

```
6  5  4  2  6
1  2  5  1  3
6  3  2  2  1
6  4  2  6  4
```

Case Study: Random-Number Generation …

- The argument 6, called the scaling factor, represents the number of unique values that next Int should produce (in this case six—0,1,2,3,4 and 5).
- This manipulation is called scaling the range of values produced by Random method nextInt.
- A six-sided die has the numbers 1–6 on its faces, not 0–5.
- So we shift the range of numbers produced by adding a shifting value, in this case 1—to our previous result, as in *face = 1 + randomNumbers.nextInt(6);*
- The shifting value (1) specifies the first value in the desired range of random integers.
- The preceding statement assigns face a random integer in the range 1–6.

Method Overloading

- Methods of the same name can be declared in the same class, as long as they have different sets of parameters (determined by the number, types and order of the parameters), this is called **method overloading**.

- When an overloaded method is called, the compiler selects the appropriate method by examining the **number, types and order of the arguments** in the call.

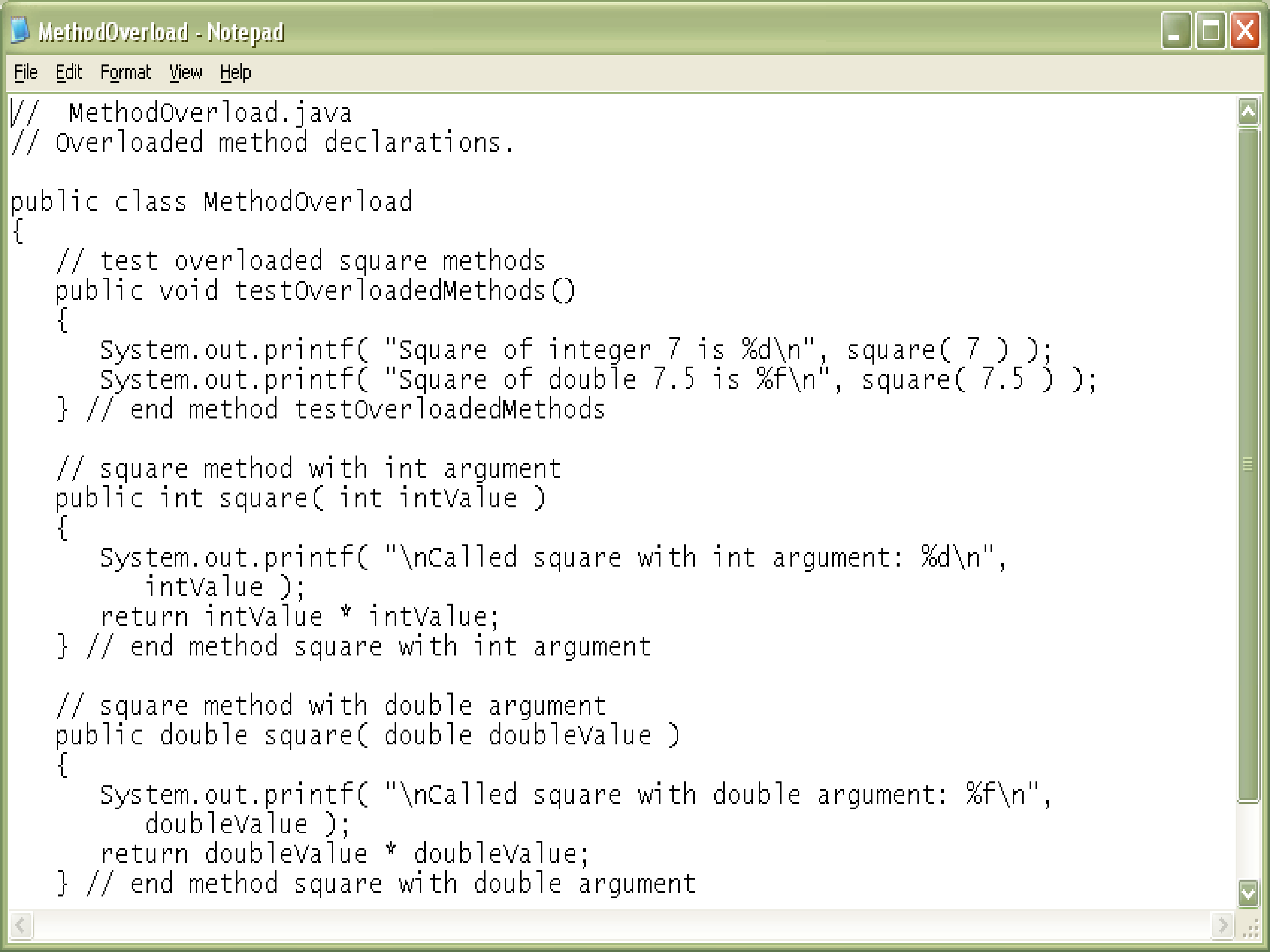
- Method overloading is commonly used to create several methods with the same name that perform the same or similar tasks, but on different types or different numbers of arguments.

Method Overloading …

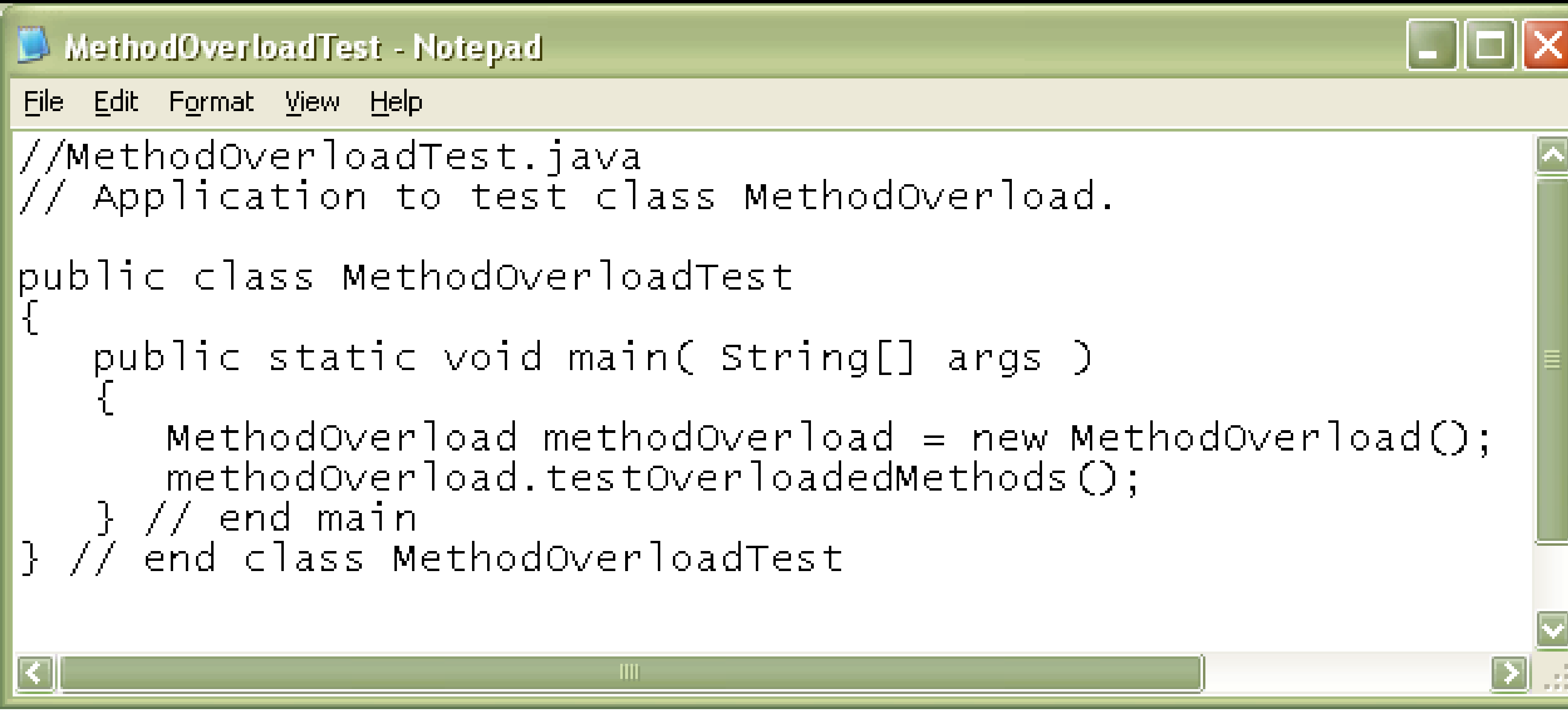
For example, Math methods abs, min and max are overloaded with four versions each:

1. One with two double parameters.
2. One with two float parameters.
3. One with two int parameters.
4. One with two long parameters.

- Class MethodOverload includes two overloaded versions of method square, one that calculates the square of an int (and returns an int) and one that calculates the square of a double (and returns a double).
- Although these methods have the same name and similar parameter lists and bodies, think of them simply as different methods.
- It may help to think of the method names as “square of int” and “square of double”, respectively.



```
//  MethodOverload.java
// Overloaded method declarations.

public class MethodOverload
{
   // test overloaded square methods
   public void testOverloadedMethods()
   {
      System.out.printf( "Square of integer 7 is %d\n", square( 7 ) );
      System.out.printf( "Square of double 7.5 is %f\n", square( 7.5 ) );
   } // end method testOverloadedMethods

   // square method with int argument
   public int square( int intValue )
   {
      System.out.printf( "\nCalled square with int argument: %d\n",
         intValue );
      return intValue * intValue;
   } // end method square with int argument

   // square method with double argument
   public double square( double doubleValue )
   {
      System.out.printf( "\nCalled square with double argument: %f\n",
         doubleValue );
      return doubleValue * doubleValue;
   } // end method square with double argument
```



```java
//MethodOverloadTest.java
// Application to test class MethodOverload.

public class MethodOverloadTest
{
   public static void main( String[] args )
   {
      MethodOverload methodOverload = new MethodOverload();
      methodOverload.testOverloadedMethods();
   } // end main
} // end class MethodOverloadTest
```

```
Called square with int argument: 7
Square of integer 7 is 49

Called square with double argument: 7.500000
Square of double 7.5 is 56.250000
```

Method Overloading …

- Line 9 invokes method square with the argument 7.
- Literal integer values are treated as type int, so the method call in line 9 invokes the version of square at lines 14–19 that specifies an int parameter.
- Similarly, line 10 invokes method square with the argument 7.5.
- Literal floating-point values are treated as type double, so the method call in line 10 invokes the version of square at lines 22–27 that specifies a double parameter.

LAB WORK

- Create a method print with the following overloaded forms
  - Takes 3 arguments of type string and prints them on screen
  - Takes 5 arguments of type float and prints them on screen
- Create appropriate class and test class to call these functions.