Sub-Programs and Java Methods

Software Development 1 (F27SA)

Michael Lones

Week 4, lecture 2

Today’s Lecture

- What is a sub-program?
- Java methods
- Passing arguments and returning results

What is a sub-program?

A sub-program is a self-contained part of a program that does a particular thing

- It is common practice to split up large programs into a number of smaller sub-programs
- This makes them easier to read and understand
- It also allows the same piece of code to be run multiple times in the same program

Sub-programs in Java

This is the general idea:

```
public class MyProgram {
  main(String args[]) {
    // do something
    // do something else
  }
}
```

```
public class MyProgram {
  main(String args[]) {
    subprogram1();
    subprogram2();
  }
  subprogram1() {
    // do something
  }
  subprogram2() {
    // do something else
  }
}
```

Sub-programs in Java

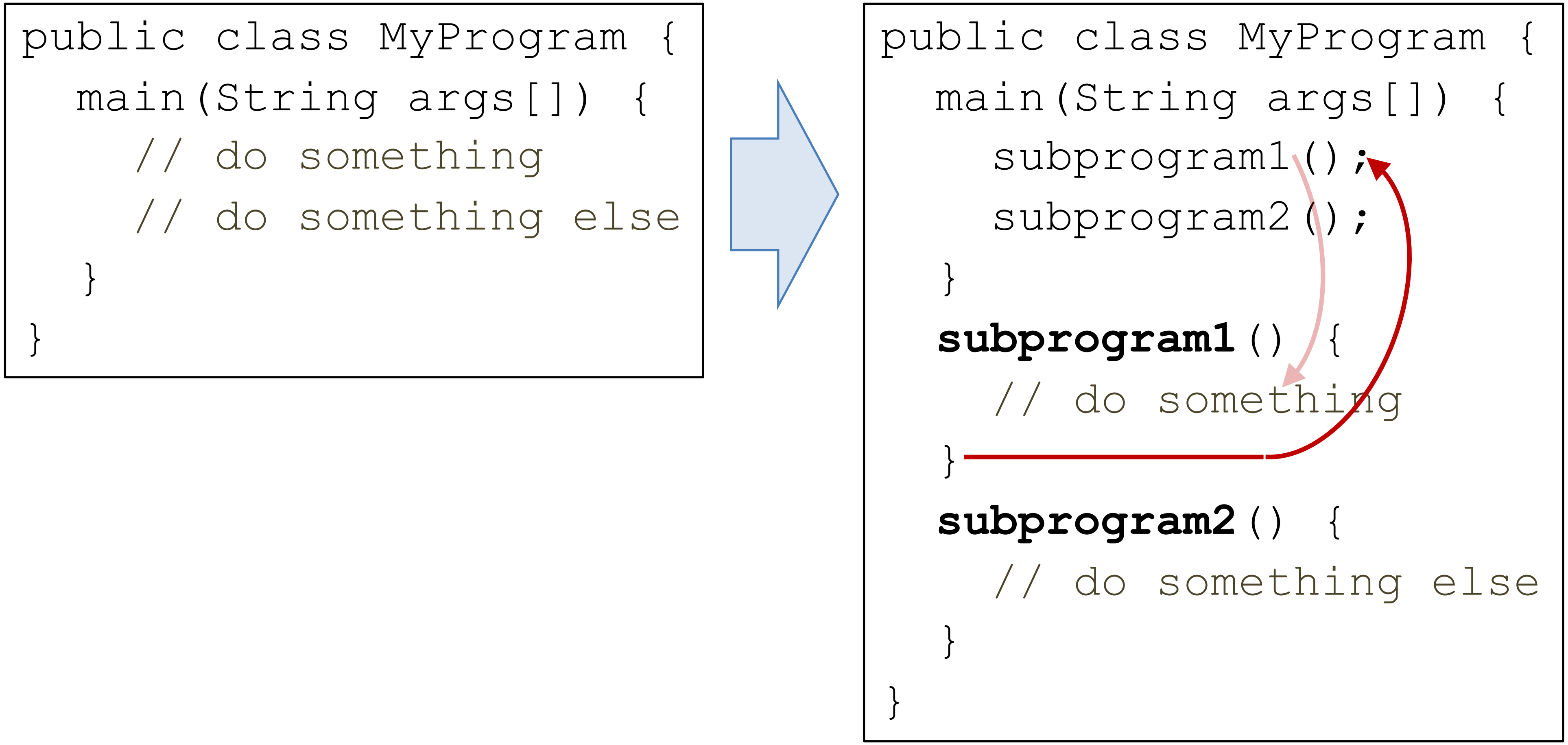
This is the general idea:

```
public class MyProgram {
  main(String args[]) {
    // do something
    // do something else
  }
}
```

```
public class MyProgram {
  main(String args[]) {
    subprogram1();
    subprogram2();
  }
  subprogram1() {
    // do something
  }
  subprogram2() {
    // do something else
  }
}
```

Sub-programs in Java

This is the general idea:

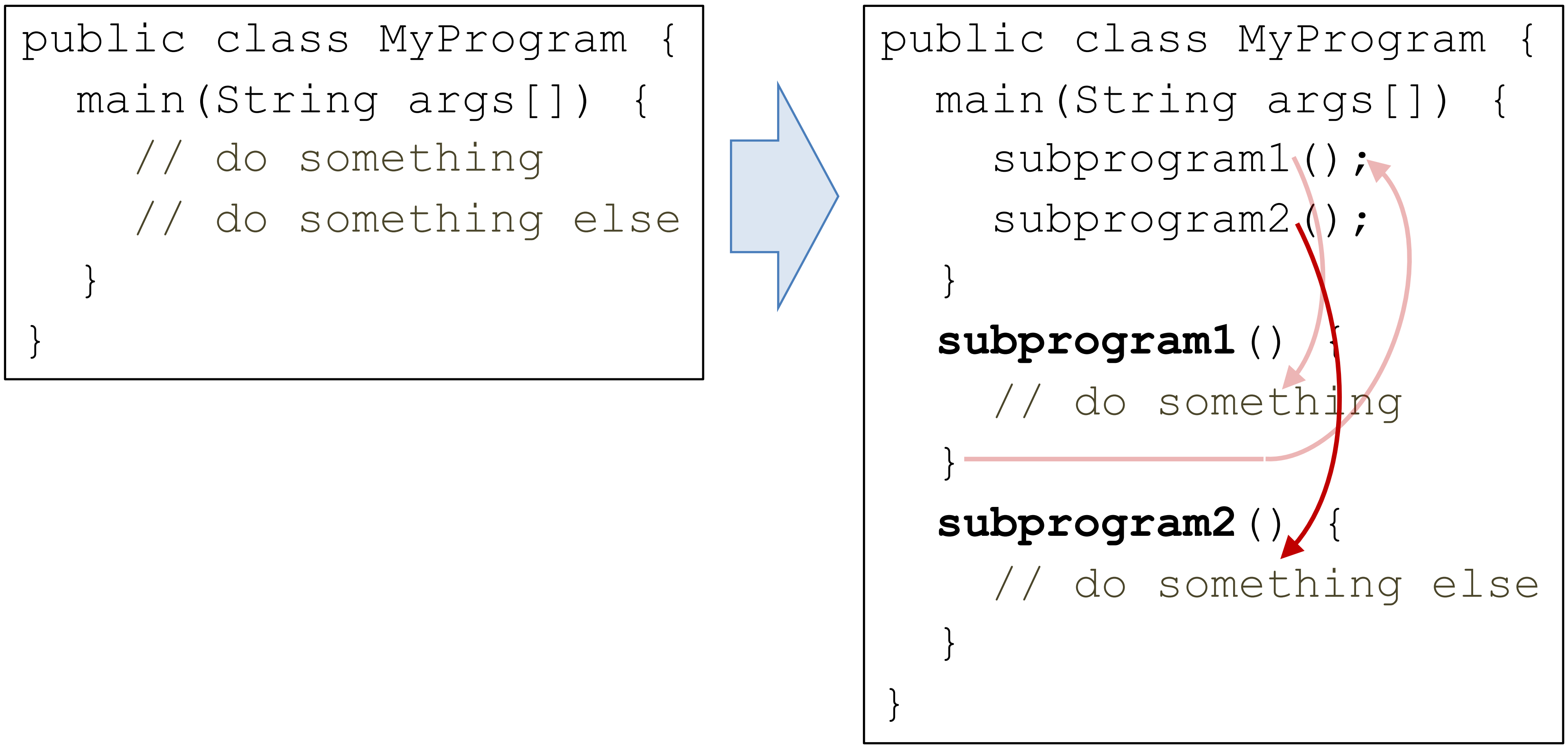


```
public class MyProgram {
  main(String args[]) {
    // do something
    // do something else
  }
}
```

```
public class MyProgram {
  main(String args[]) {
    subprogram1();
    subprogram2();
  }
  subprogram1() {
    // do something
  }
  subprogram2() {
    // do something else
  }
}
```

Sub-programs in Java

This is the general idea:

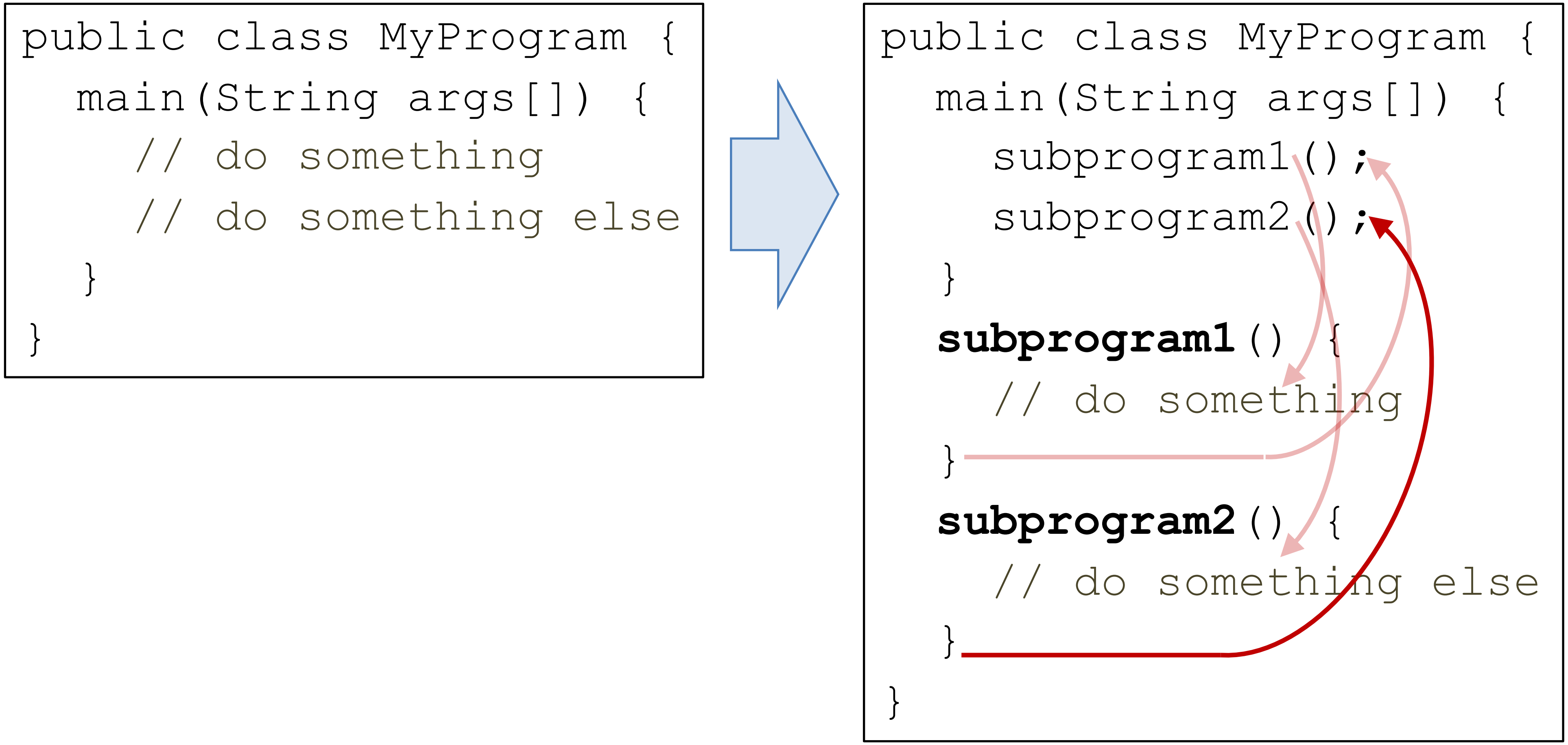


```
public class MyProgram {
  main(String args[]) {
    // do something
    // do something else
  }
}
```

```
public class MyProgram {
  main(String args[]) {
    subprogram1();
    subprogram2();
  }
  subprogram1() {
    // do something
  }
  subprogram2() {
    // do something else
  }
}
```

Sub-programs in Java

This is the general idea:



```
public class MyProgram {
  main(String args[]) {
    // do something
    // do something else
  }
}
```

```
public class MyProgram {
  main(String args[]) {
    subprogram1();
    subprogram2();
  }
  subprogram1() {
    // do something
  }
  subprogram2() {
    // do something else
  }
}
```

Some simple examples

```java
public class SubprogramDemo1 {
  public static void main(String[] args) {
    for(int i=0; i<=10; i++)
      System.out.print(i+" ");
    for(int i=10; i>0; i--)
      System.out.print(i+" ");
  }
}
```

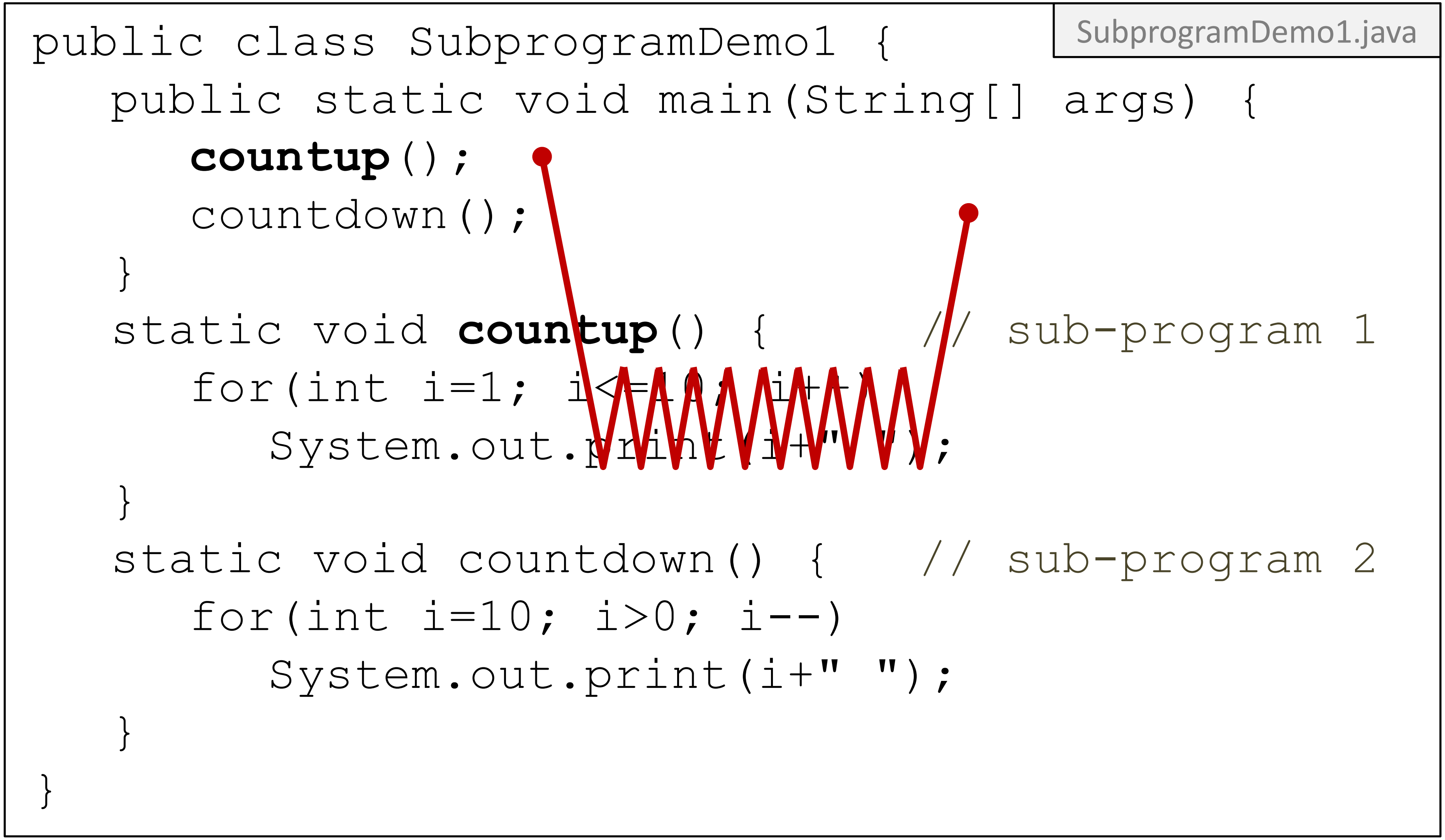
This can be turned into two sub-programs →

Some simple examples

SubprogramDemo1.java

```
public class SubprogramDemo1 {
   public static void main(String[] args) {
      countup();
      countdown();
   }
   static void countup() {     // sub-program 1
      for(int i=1; i<=10; i++)
         System.out.print(i+" ");
   }
   static void countdown() {   // sub-program 2
      for(int i=10; i>0; i--)
         System.out.print(i+" ");
   }
}
```

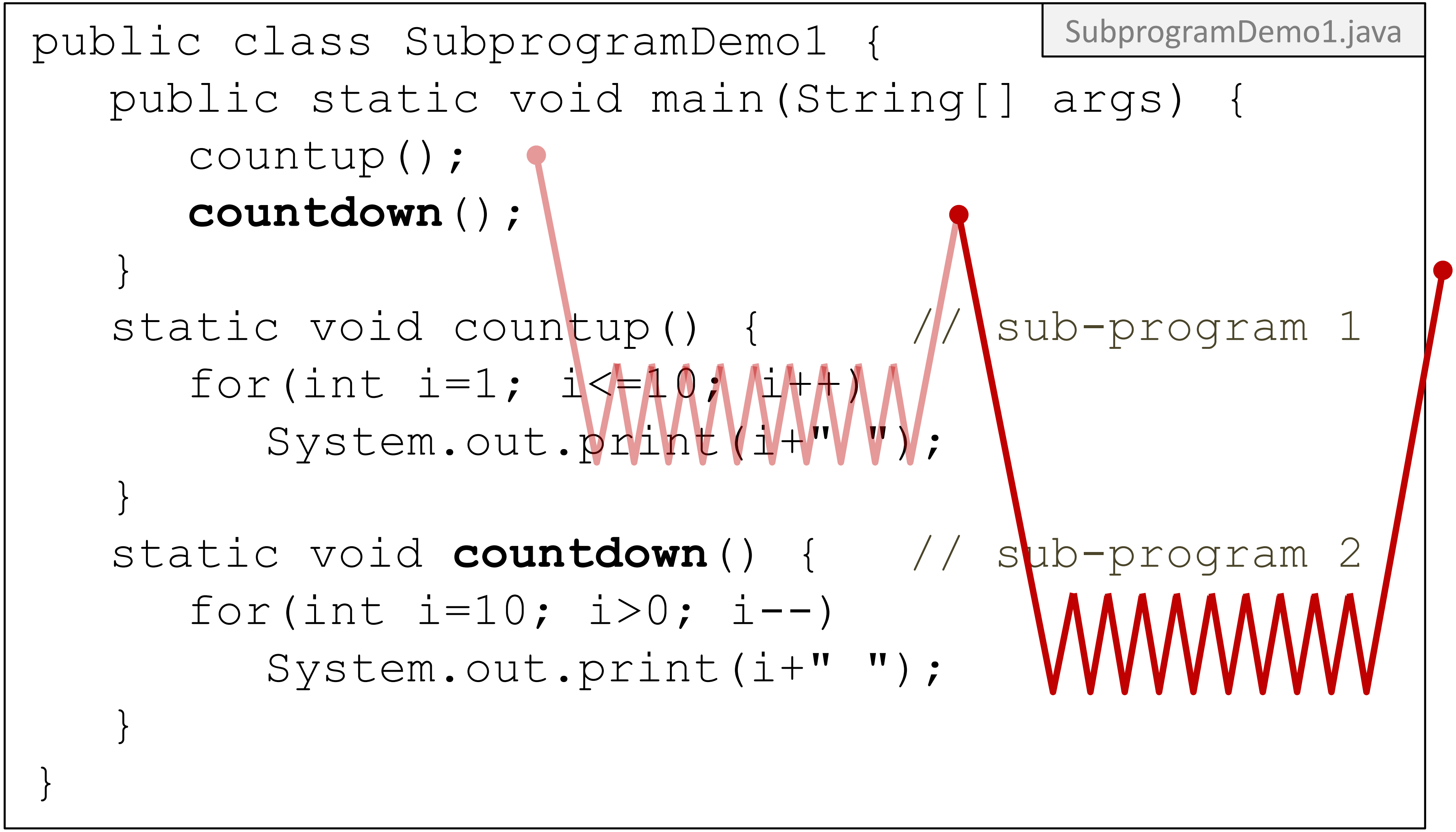
Some simple examples



SubprogramDemo1.java

```
public class SubprogramDemo1 {
   public static void main(String[] args) {
      countup();
      countdown();
   }
   static void countup() {      // sub-program 1
      for(int i=1; i<=10; i++)
         System.out.print(i+" ");
   }
   static void countdown() {    // sub-program 2
      for(int i=10; i>0; i--)
         System.out.print(i+" ");
   }
}
```

Some simple examples



SubprogramDemo1.java

```
public class SubprogramDemo1 {
   public static void main(String[] args) {
      countup();
      countdown();
   }
   static void countup() {      // sub-program 1
      for(int i=1; i<=10; i++)
         System.out.print(i+" ");
   }
   static void countdown() {    // sub-program 2
      for(int i=10; i>0; i--)
         System.out.print(i+" ");
   }
}
```

Some simple examples

We can **call** a sub-program more than once:

SubprogramDemo2.java

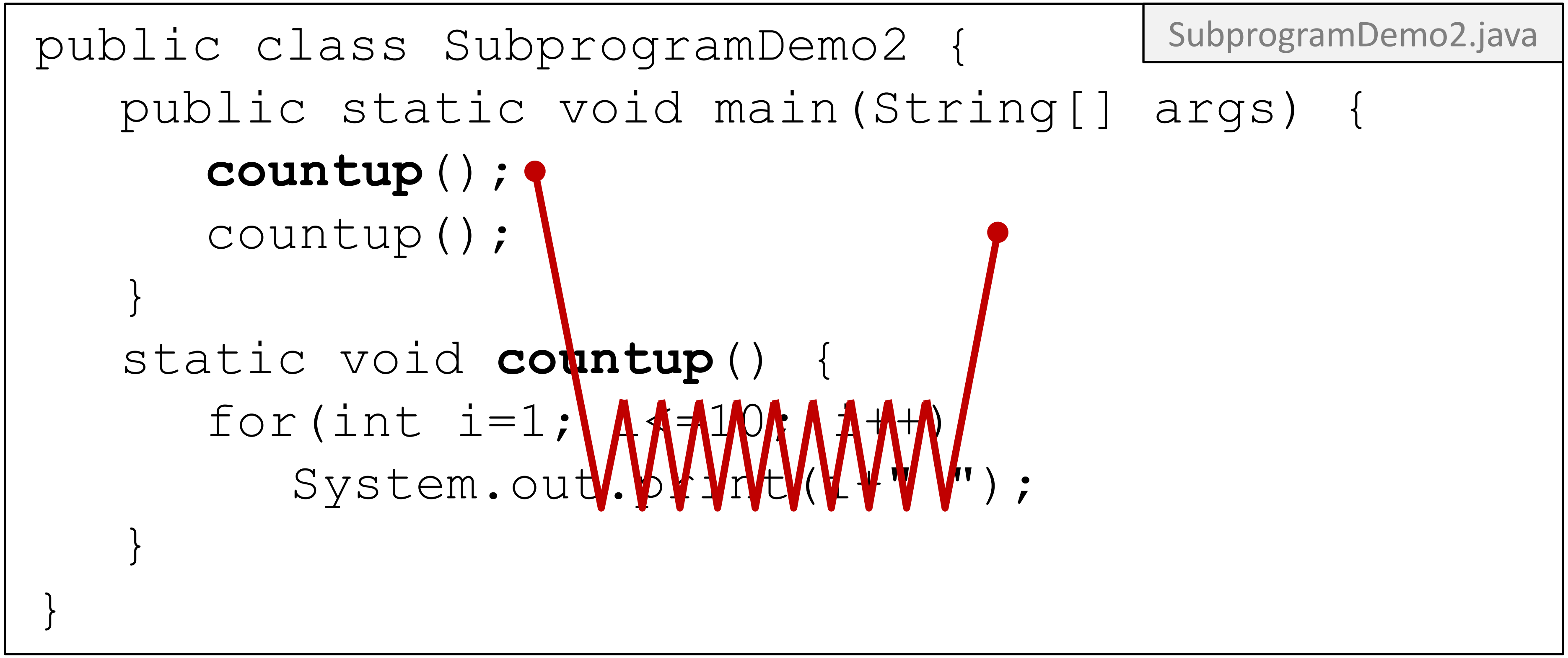
```
public class SubprogramDemo2 {
   public static void main(String[] args) {
      countup();
      countup();
   }
   static void countup() {
      for(int i=1; i<=10; i++)
         System.out.print(i+" ");
   }
}
```

Terminal

```
$ java SubprogramDemo2
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10
```

Some simple examples

We can **call** a sub-program more than once:



SubprogramDemo2.java

```
public class SubprogramDemo2 {
   public static void main(String[] args) {
      countup();
      countup();
   }
   static void countup() {
      for(int i=1; i<=10; i++)
         System.out.print(i+" ");
   }
}
```

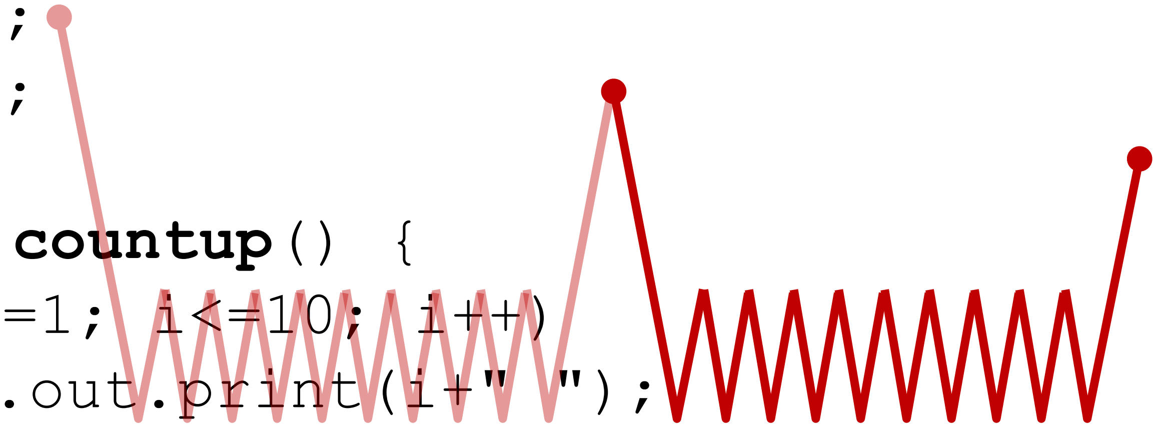
```
$ java SubprogramDemo2
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10
```

Some simple examples

We can **call** a sub-program more than once:

SubprogramDemo2.java

```
public class SubprogramDemo2 {
   public static void main(String[] args) {
      countup();
      countup();
   }
   static void countup() {
      for(int i=1; i<=10; i++)
         System.out.print(i+" ");
   }
}
```



```
$ java SubprogramDemo2
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10
```

Any Questions?

Some simple examples

What if a variable is used in both parts of the program?

```java
public class SubprogramDemo3 {
  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int max = scan.nextInt();

    for(int i=1; i<=max; i++) {
      Sytem.out.print(i+" ");
    }

    for(int i=1; i<=max; i++) {
      Sytem.out.print(i+" ");
    }
  }
}
```

Terminal

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

Some simple examples

Can we still do this?

SubprogramDemo3a.java

```
public class SubprogramDemo3a {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      int max = scan.nextInt();
      countup();
      countup();
   }
   static void countup() {
      for(int i=1; i<=max; i++)
         System.out.print(i+" ");
   }
}
```

Some simple examples

Can we still do this?

SubprogramDemo3a.java

```
public class SubprogramDemo3a {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      int max = scan.nextInt();
      countup();
      countup();
   }
   static void countup() {
      for(int i=1; i<=max; i++)
         System.out.print(i+" ");
   }
}
```

No, this won't work. The code in a sub-program can not see any of the code in `main`, including its variables.

Scope

A variable can only be seen, used or modified within the block where it is declared

- This is known as a variable's **scope**
- It means that a variable declared in one sub-program can not be seen in another sub-program

Scope

A variable can only be seen, used or modified within the block where it is declared

- This is known as a variable's **scope**
- It means that a variable declared in one sub-program can not be seen in another sub-program
- More generally, a variable declared within **`{…}`**s can not be seen outside of the **`{…}`**s.
- This includes variables declared in **`for`** loops, where the **`for`** statement in considered part of the block

Scope

A variable can only be seen, used or modified within the block where it is declared

So, this piece of code won't work:

```
   for(int i=1; i<=10; i++) {
      System.out.println(i);
   }
   System.out.println("Looped "+i+" times.");
```

Since `i` can only be seen in the for loop's block, and not in the code following the for loop.

Information Hiding

Scope is an example of a broader programming concept known as **information hiding**

- Basically, each part of a program should know as little as possible about other parts of the program
- This means different parts of the program are less likely to interfere with each other
- It also means that different parts of a program can be written separately, potentially by different people

Some simple examples

So, how do we get this working?

SubprogramDemo3a.java

```
public class SubprogramDemo3a {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      int max = scan.nextInt();
      countup();
      countup();
   }
   static void countup() {
      for(int i=1; i<=max; i++)
         System.out.print(i+" ");
   }
}
```

Passing arguments

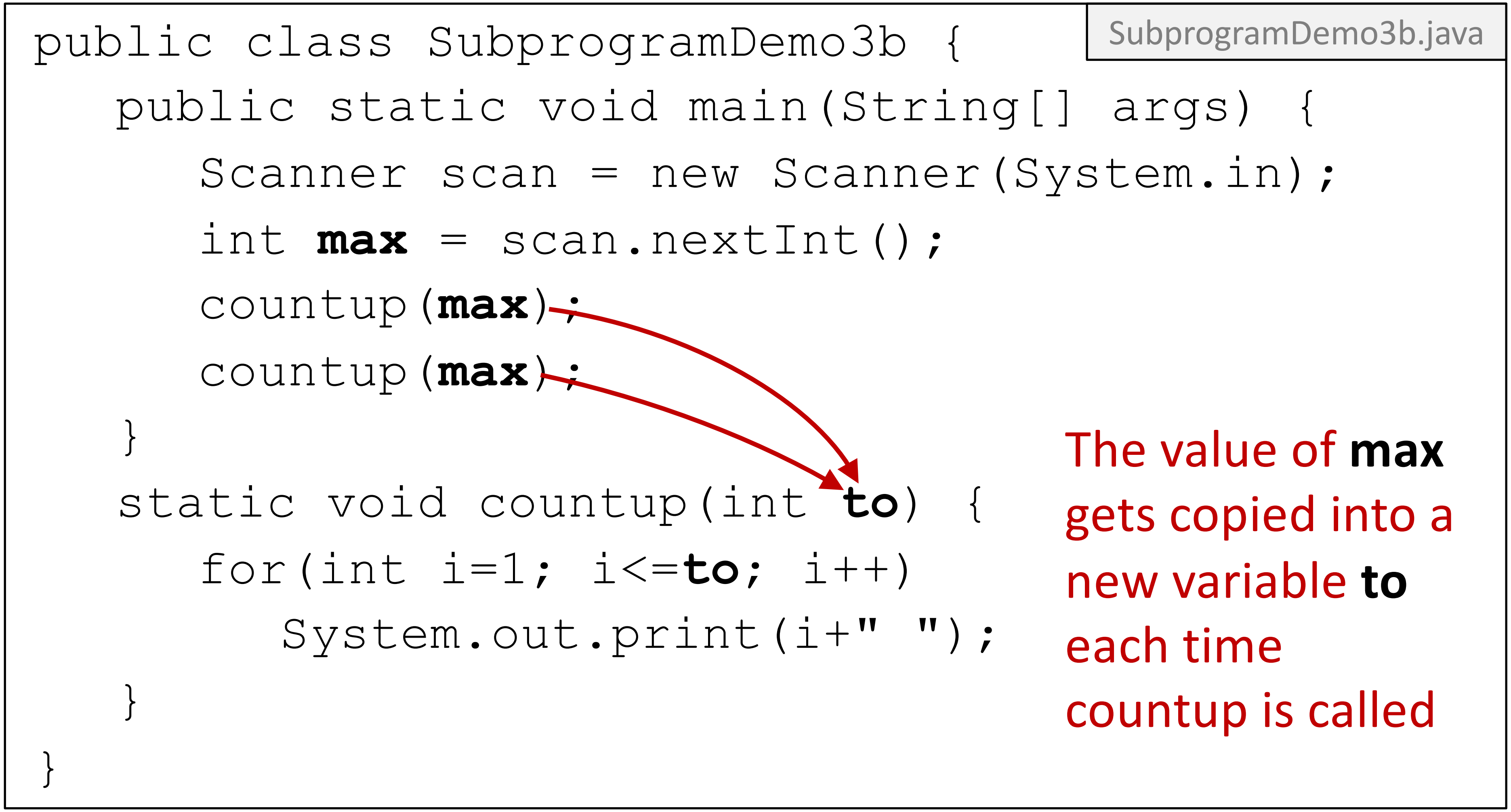
We can **pass an argument** to the sub-program:

SubprogramDemo3b.java

```
public class SubprogramDemo3b {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      int max = scan.nextInt();
      countup(max);
      countup(max);
   }
   static void countup(int to) {
      for(int i=1; i<=to; i++)
         System.out.print(i+" ");
   }
}
```

Passing arguments

We can **pass an argument** to the sub-program:



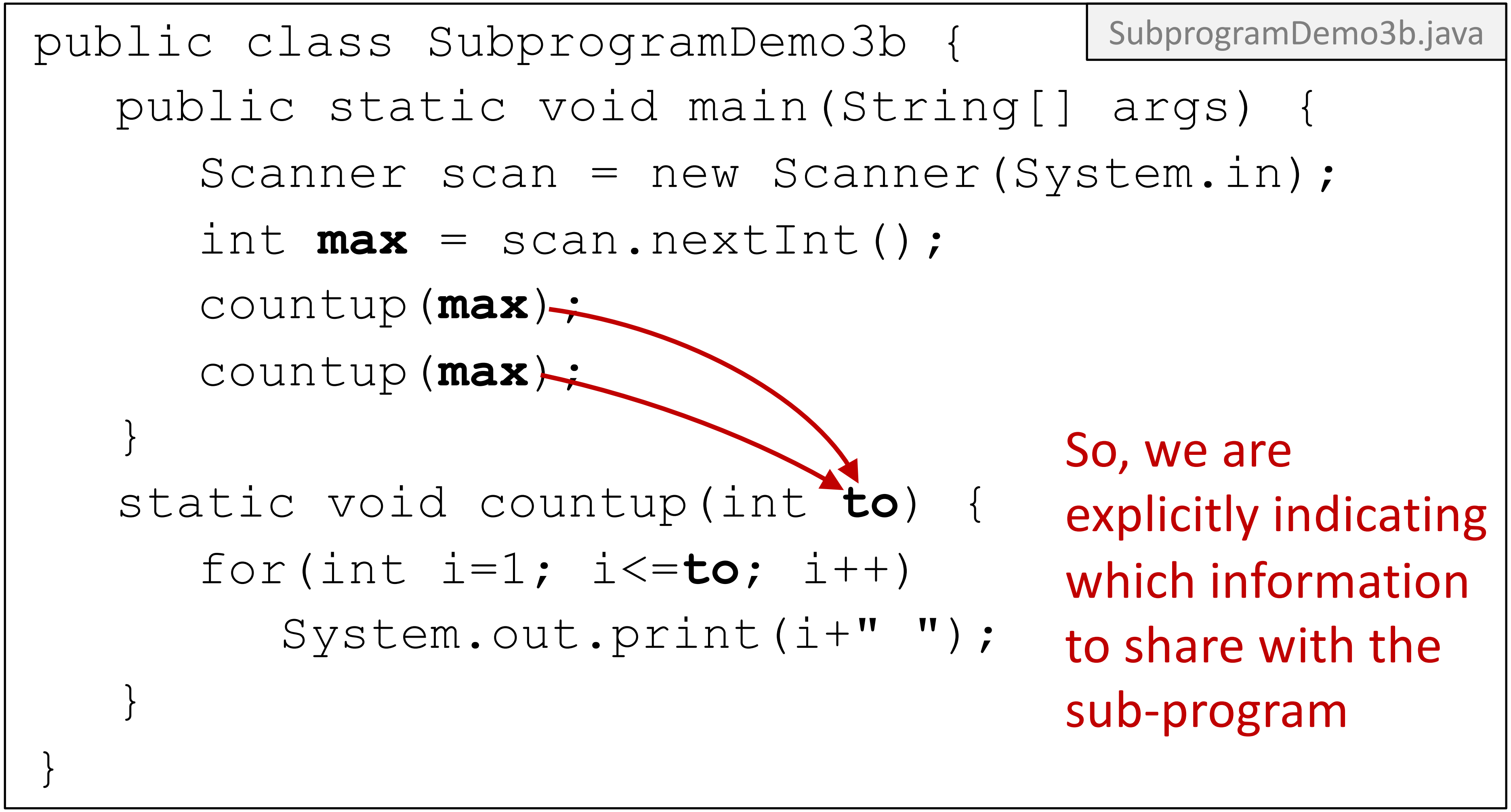
SubprogramDemo3b.java

```
public class SubprogramDemo3b {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      int max = scan.nextInt();
      countup(max);
      countup(max);
   }
   static void countup(int to) {
      for(int i=1; i<=to; i++)
         System.out.print(i+" ");
   }
}
```

The value of **max** gets copied into a new variable **to** each time countup is called

Passing arguments

We can **pass an argument** to the sub-program:



SubprogramDemo3b.java

```
public class SubprogramDemo3b {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      int max = scan.nextInt();
      countup(max);
      countup(max);
   }
   static void countup(int to) {
      for(int i=1; i<=to; i++)
         System.out.print(i+" ");
   }
}
```

So, we are explicitly indicating which information to share with the sub-program

Passing arguments

We can **pass an argument** to the sub-program:

SubprogramDemo3c.java

```
public class SubprogramDemo3c {
    public static void main(String[] args) {
        countup(2);
        countup(4);
        countup(6);
        countup(8);
    }
```

We can call the sub-program multiple times with different arguments.

Terminal

```
$ java SubprogramDemo3c
1 2 1 2 3 4 1 2 3 4 5 6 1 2 3 4 5 6 7 8
```

Passing arguments

We can **pass multiple arguments** to a sub-program:

SubprogramDemo4.java

```
public class SubprogramDemo4 {
   public static void main(String[] args) {
      count(5,43,3);
   }
   static void count(int from, int to, int step) {
      for(int i=from; i<=to; i+=step)
         System.out.print(i+" ");
   }
}
```

Terminal

```
$ java SubprogramDemo4
5 8 11 14 17 20 23 26 29 32 35 38 41
```

Any Questions?

Returning values

How do we get values back from a sub-program?

```
public class ReturnValueDemo1 {
    public static void main(String[] args) {
        double value = 5;
        square(value);
        System.out.println(value);
    }
    static void square(double arg) {
        arg = arg * arg;
    }
}
```

Not like this! If you change the value of an argument, its value won't get updated in the calling code. Passing arguments is a one-way process (at least for primitives…)

Terminal

```
$ java SubprogramDemo1
5
```

Returning values

A sub-program can explicitly **return a value**:

ReturnValueDemo1.java

```java
public class ReturnValueDemo1 {
   public static void main(String[] args) {
      double y = square(5);
      System.out.println(y);
   }
   static double square(double arg) {
      double result = arg * arg;
      return result;
   }
}
```

Terminal

```
$ java SubprogramDemo1
25
```

Returning values

A sub-program can explicitly **return a value**:

ReturnValueDemo1.java

```
public class ReturnValueDemo1 {
   public static void main(String[] args) {
      double y = square(5);
      System.out.println(y);
   }
   static double square(double arg) {
      double result = arg * arg;
      return result;
   }
}
```

We replace void with the **return type**, and use the **return** keyword to return a value

Terminal

```
$ java SubprogramDemo1
25
```

Returning values

A sub-program can explicitly **return a value**:

ReturnValueDemo1.java

```
public class ReturnValueDemo1 {
    public static void main(String[] args) {
        double y = square(5);
        System.out.println(y);
    }
    static double square(double arg) {
        double result = arg * arg;
        return result;
    }
}
```

The **method call** then **evaluates to** this return value

We replace void with the **return type**, and use the **return** keyword to return a value

Terminal

```
$ java SubprogramDemo1
25
```

Returning values

A sub-program can contain **multiple return statements**:

ReturnValueDemo2.java

```
public class ReturnValueDemo2 {
    public static void main(String[] args) {
        System.out.println( max(2, 4) );
    }
    static double max(double arg1, double arg2) {
        if(arg1>arg2) return arg1;
        else return arg2;
    }
}
```

Terminal

```
$ java SubprogramDemo2
4
```

Returning values

Return values can be used in any expression:

ReturnValueDemo2.java

```
public class ReturnValueDemo3 {
    public static void main(String[] args) {
        System.out.println( square(5) * max(2, 4) );
    }
    static double square(double arg) {
        return arg * arg;
    }
    static double max(double arg1, double arg2) {
        if(arg1>arg2) return arg1;;
        else return arg2;
    }
}
```

Terminal

```
20
```

Returning values

Return values can be used as arguments:

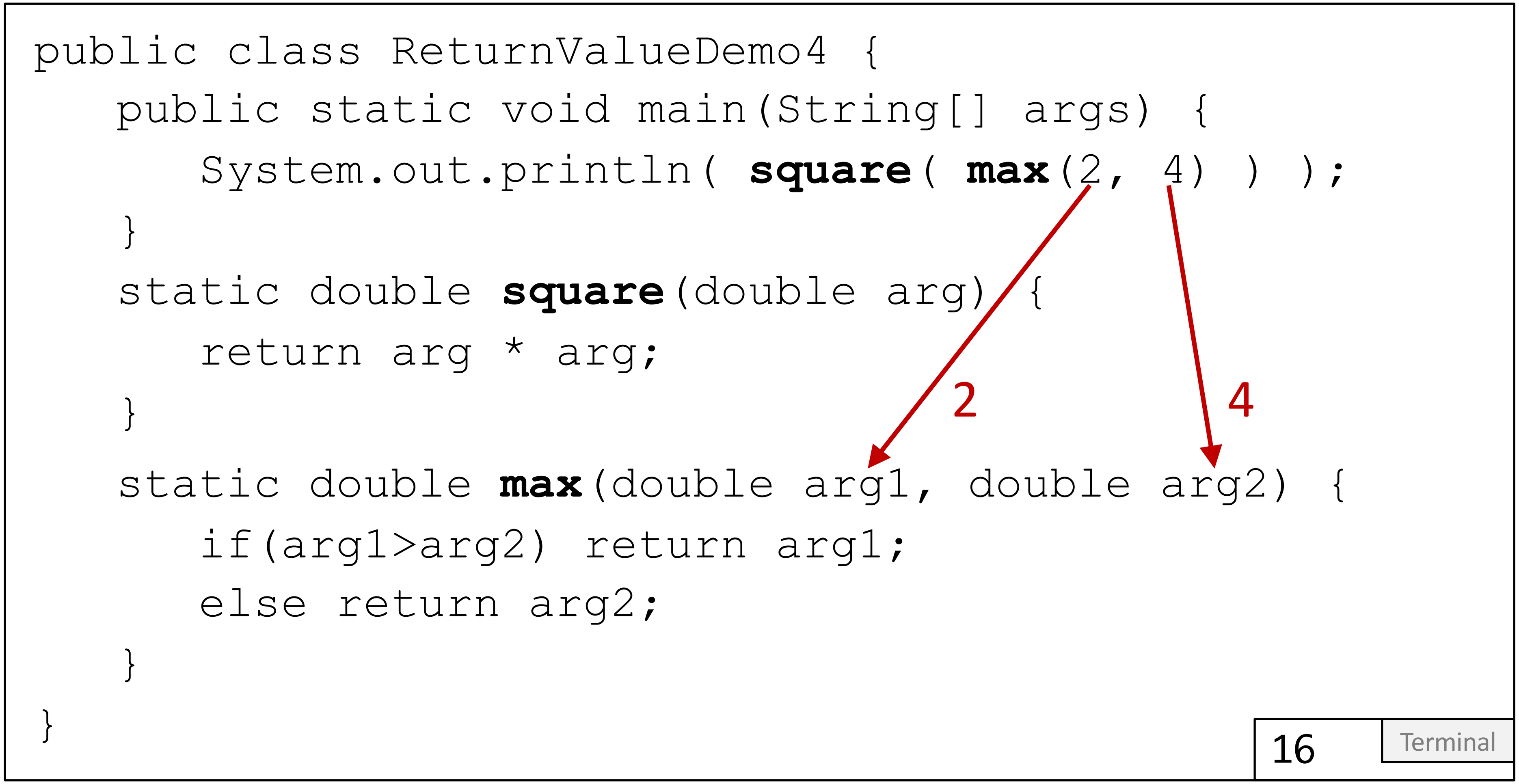
```
public class ReturnValueDemo4 {
   public static void main(String[] args) {
      System.out.println( square( max(2, 4) ) );
   }
   static double square(double arg) {
      return arg * arg;
   }
   static double max(double arg1, double arg2) {
      if(arg1>arg2) return arg1;
      else return arg2;
   }
}
```

Terminal

```
16
```

Returning values

Return values can be used as arguments:

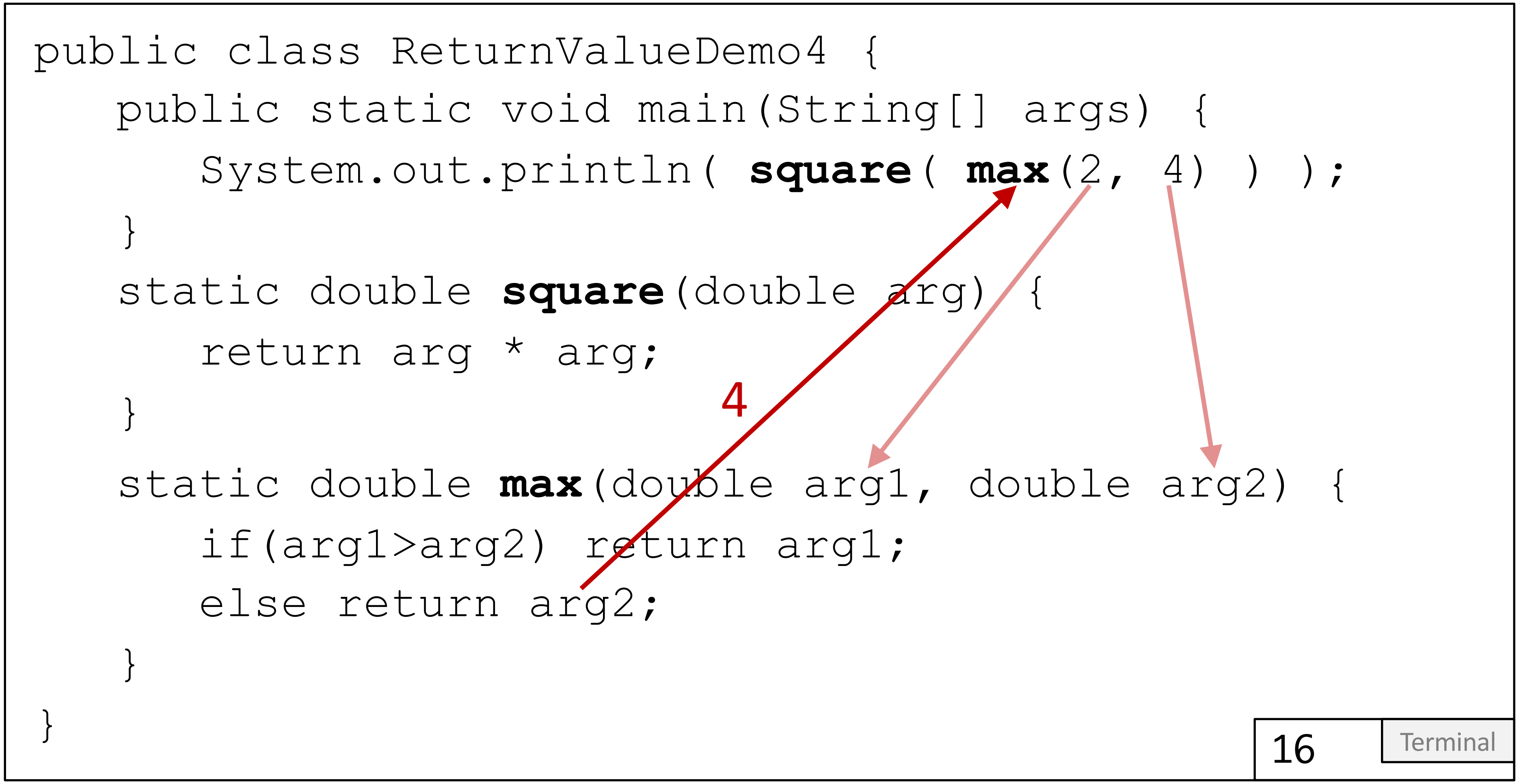


```
public class ReturnValueDemo4 {
   public static void main(String[] args) {
      System.out.println( square( max(2, 4) ) );
   }
   static double square(double arg) {
      return arg * arg;
   }
   static double max(double arg1, double arg2) {
      if(arg1>arg2) return arg1;
      else return arg2;
   }
}
```

```
16
```

Returning values

Return values can be used as arguments:

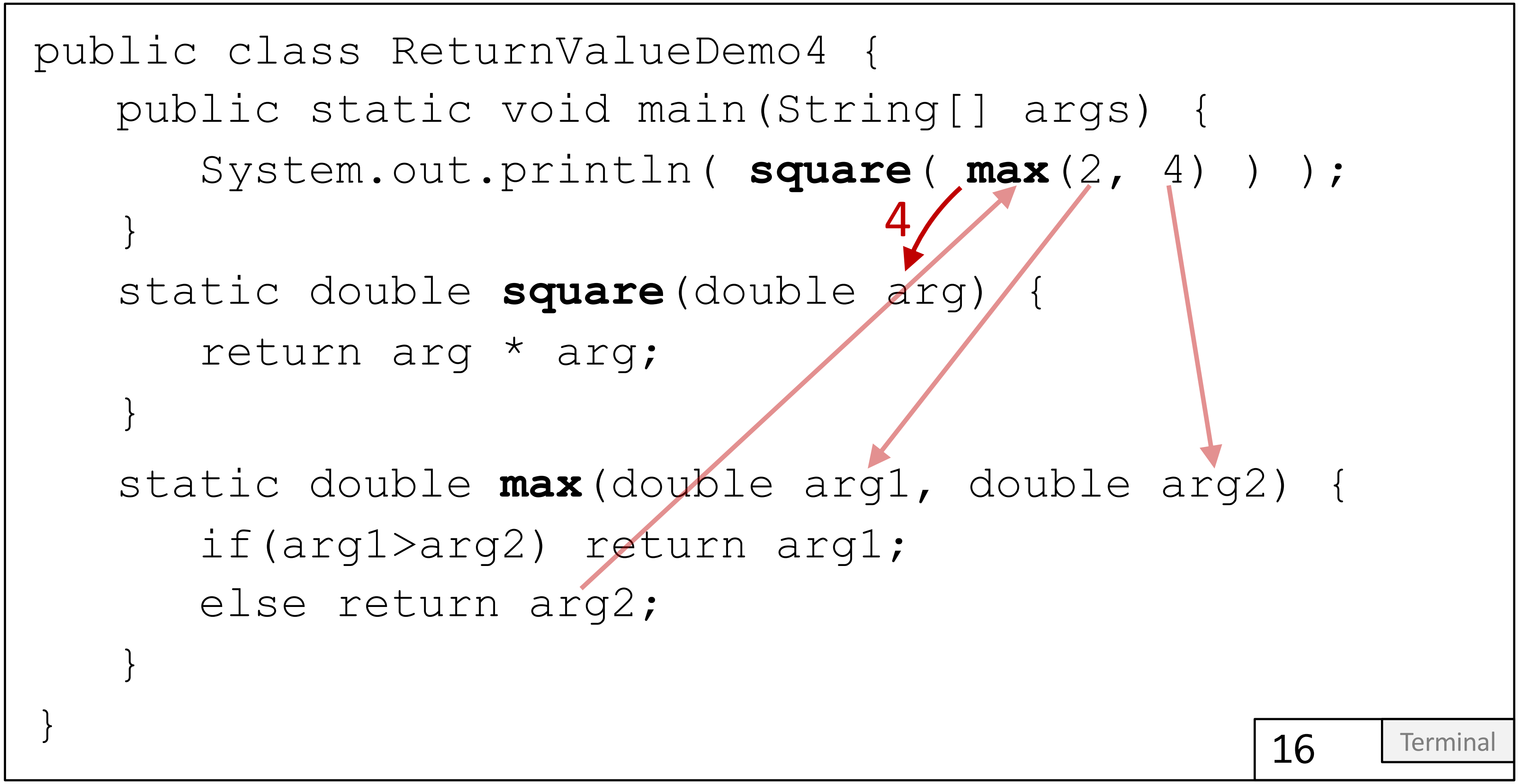


```
public class ReturnValueDemo4 {
   public static void main(String[] args) {
      System.out.println( square( max(2, 4) ) );
   }
   static double square(double arg) {
      return arg * arg;
   }
   static double max(double arg1, double arg2) {
      if(arg1>arg2) return arg1;
      else return arg2;
   }
}
```

```
16
```

Returning values

Return values can be used as arguments:



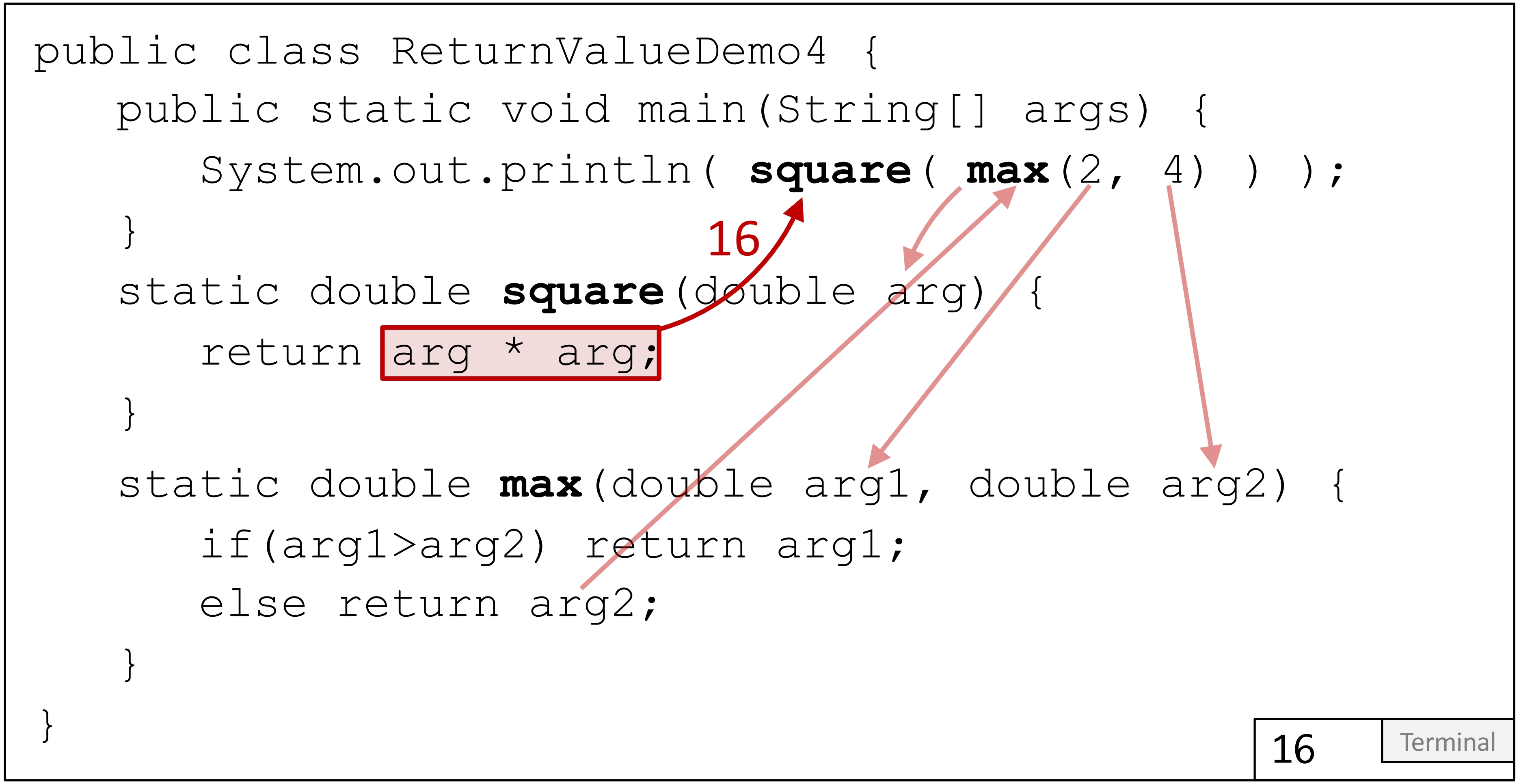
```
public class ReturnValueDemo4 {
   public static void main(String[] args) {
      System.out.println( square( max(2, 4) ) );
   }
   static double square(double arg) {
      return arg * arg;
   }
   static double max(double arg1, double arg2) {
      if(arg1>arg2) return arg1;
      else return arg2;
   }
}
```

Terminal

```
16
```

Returning values

Return values can be used as arguments:



```
public class ReturnValueDemo4 {
   public static void main(String[] args) {
      System.out.println( square( max(2, 4) ) );
   }
   static double square(double arg) {
      return arg * arg;
   }
   static double max(double arg1, double arg2) {
      if(arg1>arg2) return arg1;
      else return arg2;
   }
}
```

Terminal

```
16
```

Any Questions?

Sub-programs calling sub-programs

It's common for sub-programs to call one another:

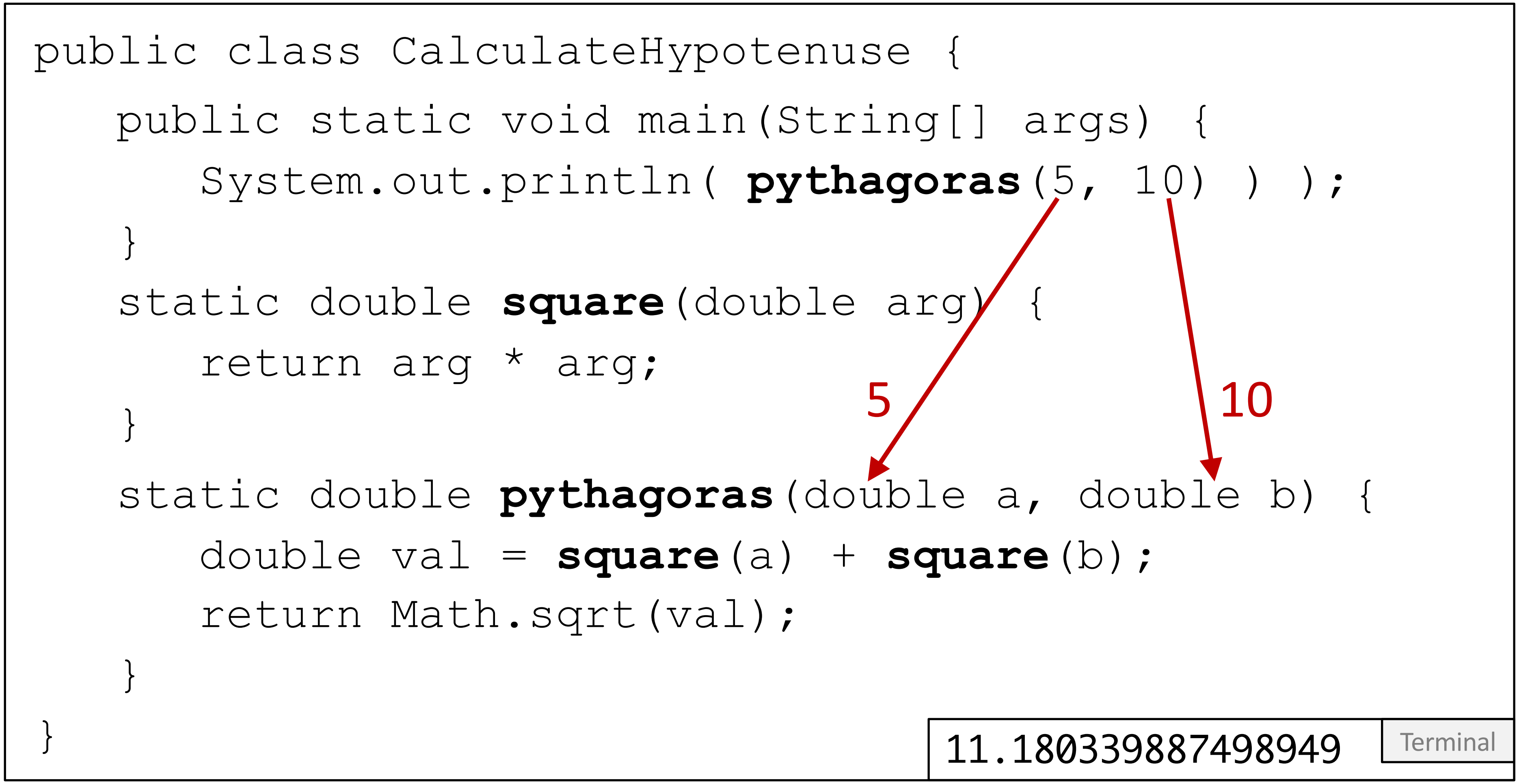
```
public class CalculateHypotenuse {
    public static void main(String[] args) {
        System.out.println( pythagoras(5, 10) );
    }
    static double square(double arg) {
        return arg * arg;
    }
    static double pythagoras(double a, double b) {
        double val = square(a) + square(b);
        return Math.sqrt(val);
    }
}
```

Terminal

```
11.180339887498949
```

Sub-programs calling sub-programs

It's common for sub-programs to call one another:

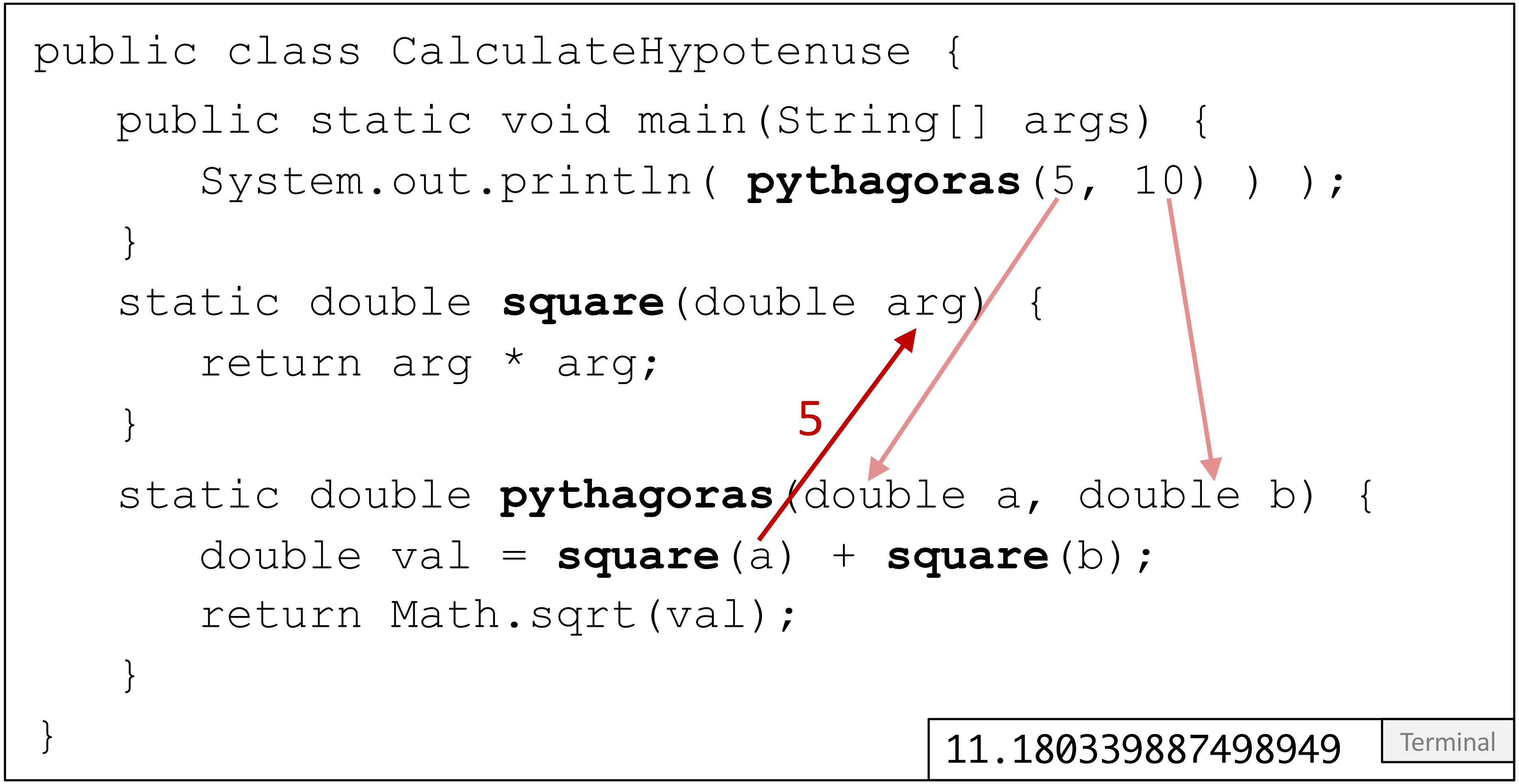


```
public class CalculateHypotenuse {
   public static void main(String[] args) {
      System.out.println( pythagoras(5, 10) ) );
   }
   static double square(double arg) {
      return arg * arg;
   }
   static double pythagoras(double a, double b) {
      double val = square(a) + square(b);
      return Math.sqrt(val);
   }
}
```

```
11.180339887498949
```

Sub-programs calling sub-programs

It's common for sub-programs to call one another:

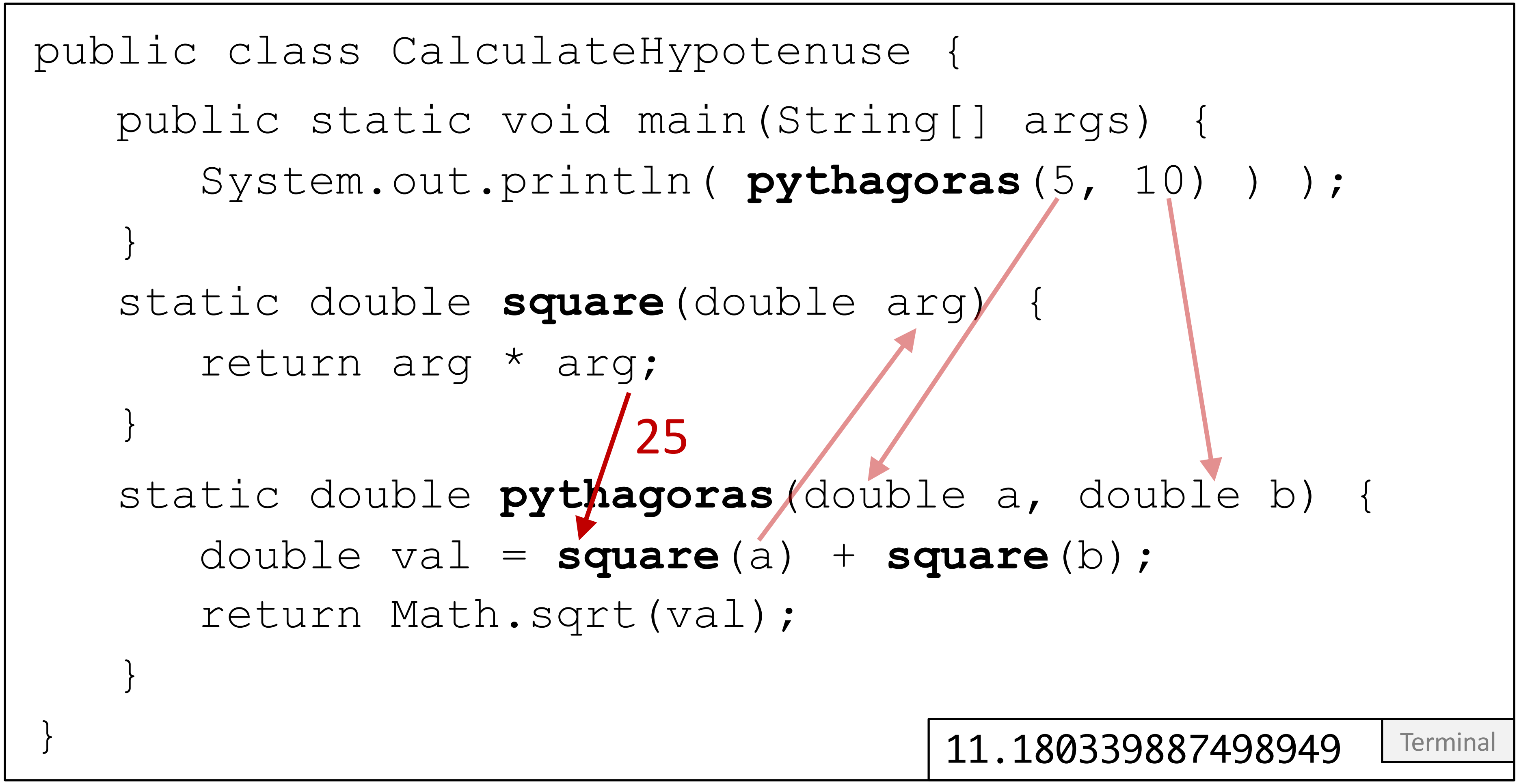


```
public class CalculateHypotenuse {
   public static void main(String[] args) {
      System.out.println( pythagoras(5, 10) ) );
   }
   static double square(double arg) {
      return arg * arg;
   }
   static double pythagoras(double a, double b) {
      double val = square(a) + square(b);
      return Math.sqrt(val);
   }
}
```

```
11.180339887498949
```

Sub-programs calling sub-programs

It's common for sub-programs to call one another:

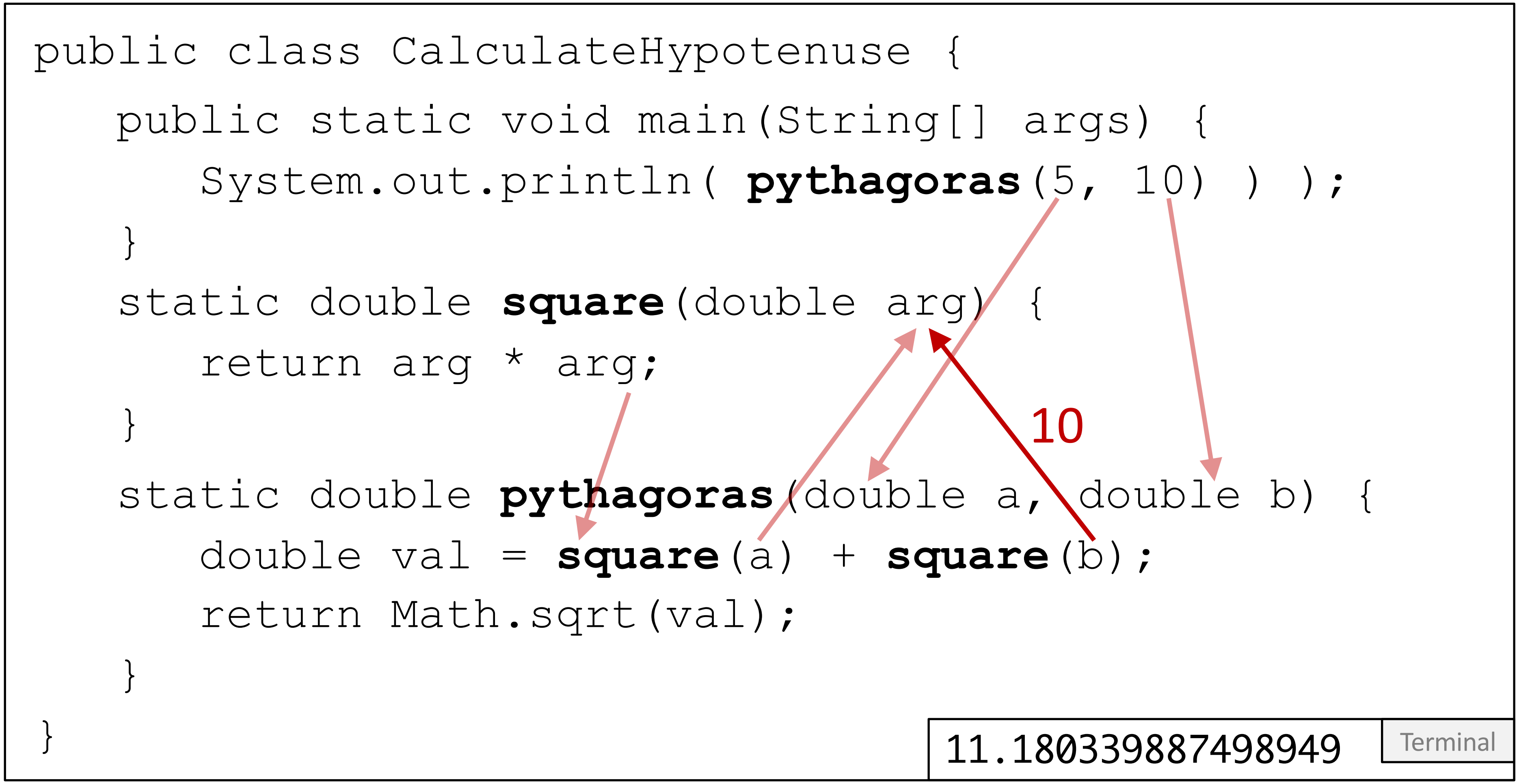


```
public class CalculateHypotenuse {
    public static void main(String[] args) {
        System.out.println( pythagoras(5, 10) ) );
    }
    static double square(double arg) {
        return arg * arg;
    }
    static double pythagoras(double a, double b) {
        double val = square(a) + square(b);
        return Math.sqrt(val);
    }
}
```

```
11.180339887498949
```

Sub-programs calling sub-programs

It's common for sub-programs to call one another:

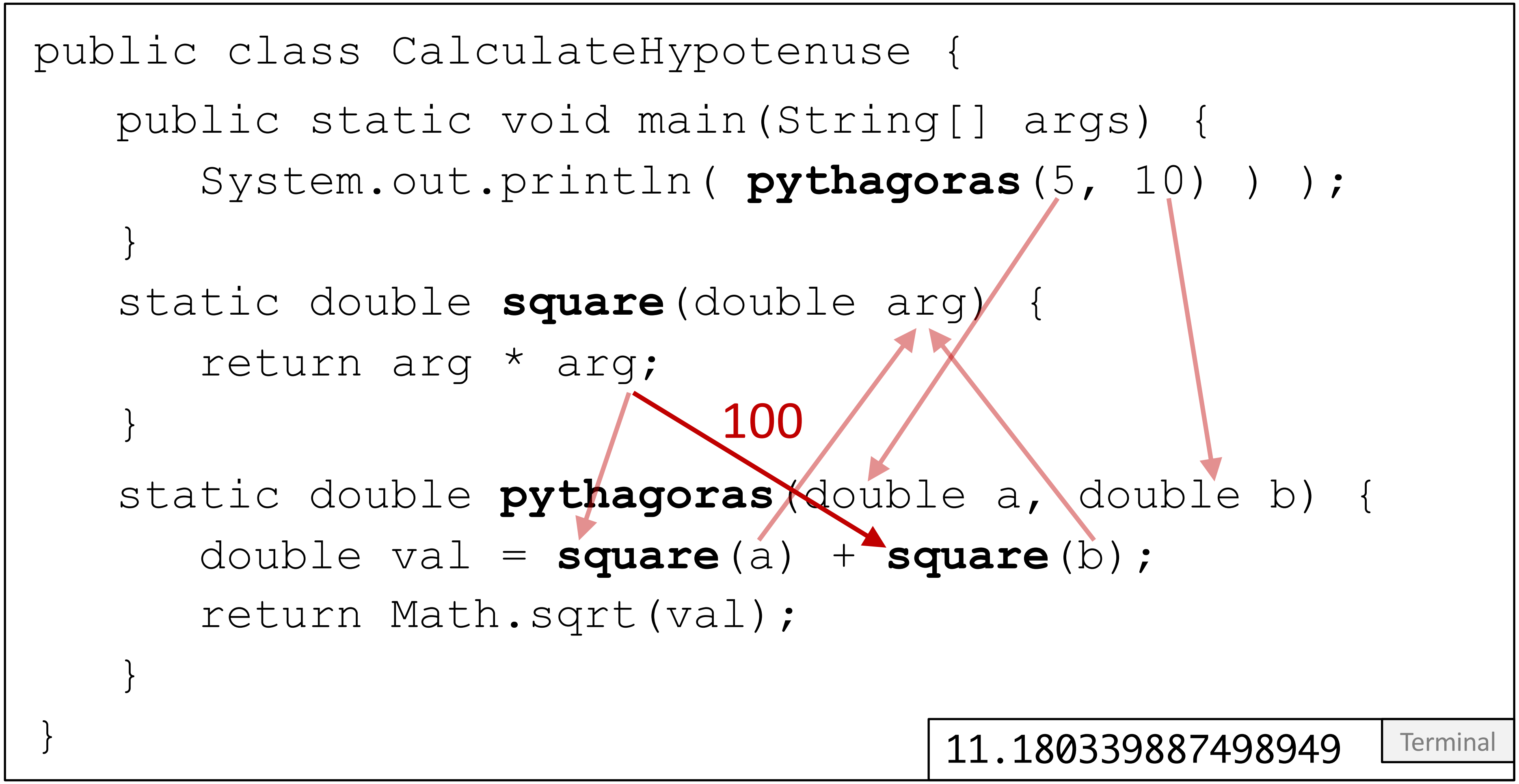


```
public class CalculateHypotenuse {
   public static void main(String[] args) {
      System.out.println( pythagoras(5, 10) ) );
   }
   static double square(double arg) {
      return arg * arg;
   }
   static double pythagoras(double a, double b) {
      double val = square(a) + square(b);
      return Math.sqrt(val);
   }
}
```

```
11.180339887498949
```

Sub-programs calling sub-programs

It's common for sub-programs to call one another:

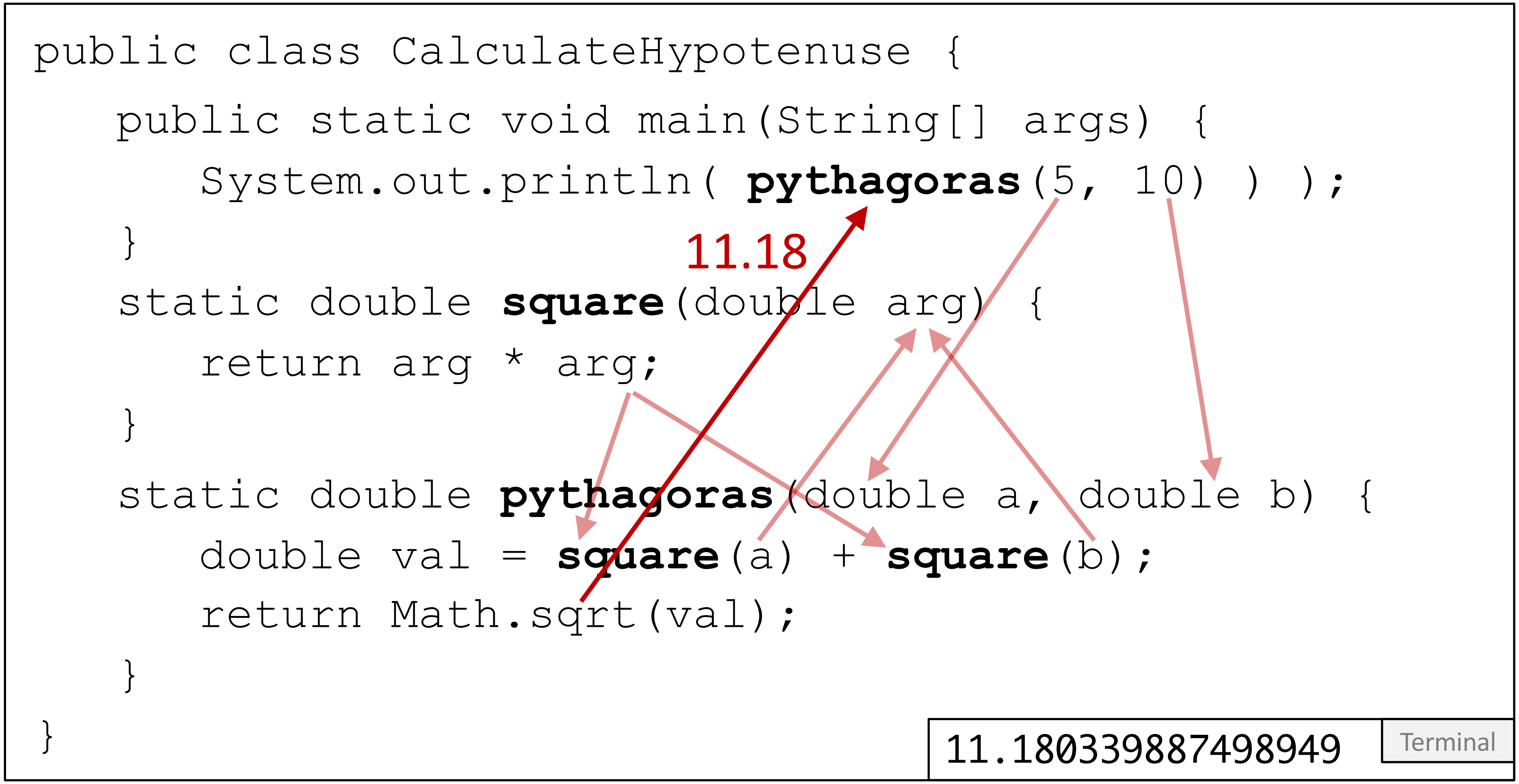


```
public class CalculateHypotenuse {
   public static void main(String[] args) {
      System.out.println( pythagoras(5, 10) ) );
   }
   static double square(double arg) {
      return arg * arg;
   }
   static double pythagoras(double a, double b) {
      double val = square(a) + square(b);
      return Math.sqrt(val);
   }
}
```

```
11.180339887498949
```

Sub-programs calling sub-programs

It's common for sub-programs to call one another:



```
public class CalculateHypotenuse {
   public static void main(String[] args) {
      System.out.println( pythagoras(5, 10) ) );
   }
   static double square(double arg) {
      return arg * arg;
   }
   static double pythagoras(double a, double b) {
      double val = square(a) + square(b);
      return Math.sqrt(val);
   }
}
```

```
11.180339887498949
```

Returning multiple values

Is it possible for a sub-program to return multiple values?

- Not in most programming languages, Java included
- Some languages (e.g. Go) do support this
- It can be achieved in Java using **data structures**
- More on this in the next lecture

Method signatures

The correct name for a sub-program in Java is a **method**. Every method has a **signature**.

A method's signature indicates:

- the name that is used to call it
- the arguments it expects
- the type of the value it returns
- other things, such as `public` and `static`, which will be explained later in SD1

Method signatures

The correct name for a sub-program in Java is a **method**. Every method has a **signature**, e.g.

```
int countLetters(String s)

String join(String s1, String s2)

double calculateVolume(double radius)

void outputMessage(String message)
```

void means that the method does not return anything

Any Questions?

Quiz

Will this work?

```
public class Q1 {
   public static void main(String[] args) {
      int a = 10;
      int b = 5;
      add();
   }
   static void add() {
      System.out.println(a+"+"+b+"="+(a+b));
   }
}
```

Quiz

Will this work?

```
public class Q2 {
   public static void main(String[] args) {
      int a = 10;
      int b = 5;
      add(a, b);
   }
   static void add(int a, int b) {
      System.out.println(a+"+"+b+"="+(a+b));
   }
}
```

Quiz

Will this work?

```
public class Q3 {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        add(a, b);
    }
    static int add(int a, int b) {
        System.out.println(a+"+"+b+"="+(a+b));
    }
}
```

Quiz

Will this work?

```
public class Q4 {
   public static void main(String[] args) {
      int a = 10;
      int b = 5;
      c = add(a, b);
      System.out.println(a+"+"+b+"="c);
   }
   static void add(int a, int b) {
      return a+b;
   }
}
```

Quiz

Will this work?

```
public class Q5 {
   public static void main(String[] args) {
      int a = 10;
      int b = 5;
      int c = divide(a, b);
      System.out.println(a+"/"+b+"="c);
   }
   static double divide(double a, double b) {
      return a / b;
   }
}
```

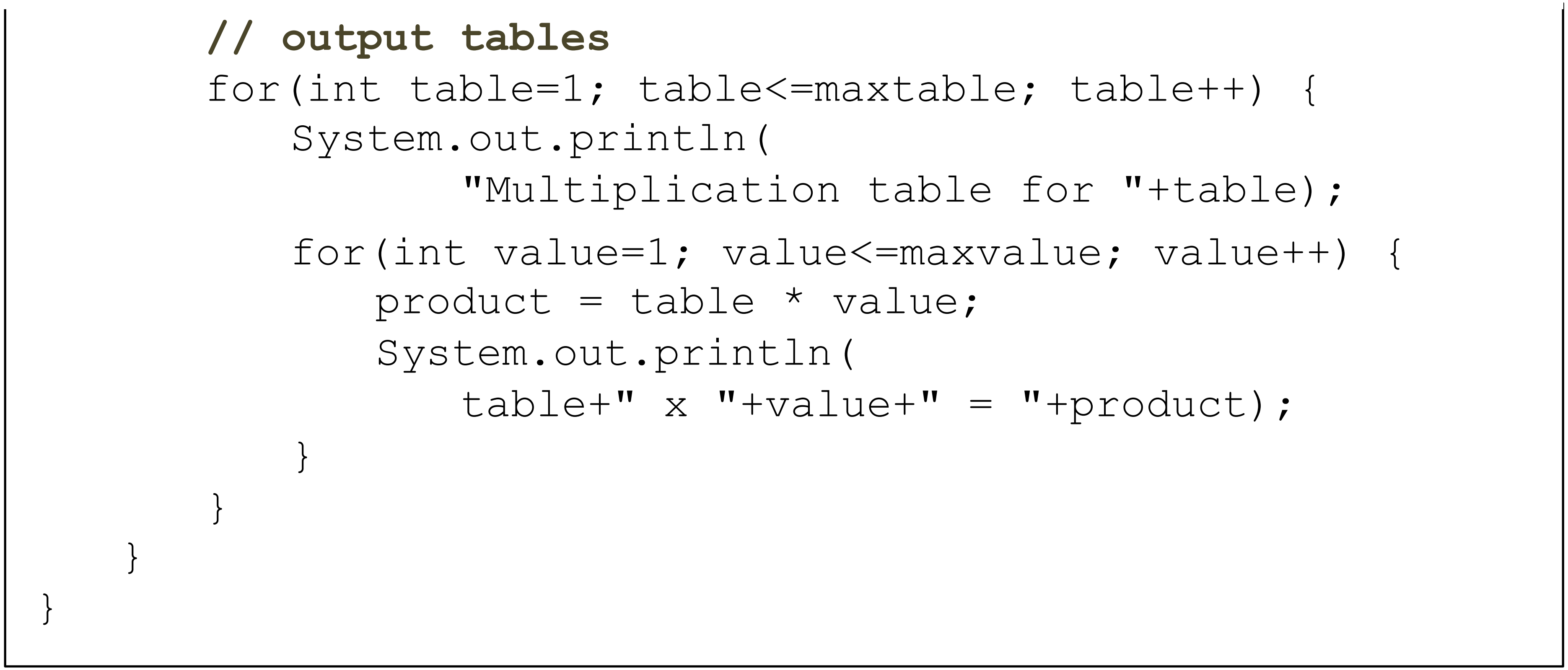
Multiplication Tables Revisited

MultiplicationTables.java

```
public class MultiplicationTables {
   public static void main(String[] args) {
      // variable declarations
      int maxtable; // largest multiplicand
      int maxvalue; // largest multiplier
      int product;  // multiplier x multiplicand

      // get input from user
      Scanner scan = new Scanner(System.in);
      System.out.println("What number would you like
                  to produce tables up to?");
      maxtable = scan.nextInt();
      System.out.println("What is the maximum
                  multiplier for each table?");
      maxvalue = scan.nextInt();
```

Multiplication Tables Revisited



```
        // output tables
        for(int table=1; table<=maxtable; table++) {
            System.out.println(
                    "Multiplication table for "+table);
            for(int value=1; value<=maxvalue; value++) {
                product = table * value;
                System.out.println(
                    table+" x "+value+" = "+product);
            }
        }
    }
}
```

Remember this program from the iteration lecture? Let's move the table printing code into a separate method:

Multiplication Tables Revisited

```
        // output tables
        for(int table=1; table<=maxtable; table++) {
            outputTable(table, maxvalue);
        }
    }

    // output a single multiplication table
    static void outputTable(int table, int upto) {
        int product;  // multiplier x multiplicand
        System.out.println(
                    "Multiplication table for "+table);
        for(int value=1; value<=upto; value++) {
            product = table * value;
            System.out.println(table+" x "+value+" = "+product);
        }
    }
}
```

Now move all the calculation code into a separate method:

Multiplication Tables Revisited

```
        outputTables(maxtable, maxvalue);
    }

    // output multiplication tables
    static void outputTables(int uptotable, int uptovalue) {
        for(int table=1; table<=uptotable; table++) {
            outputTable(table, uptovalue);
        }
    }

    // output a single multiplication table
    static void outputTable(int table, int upto) {
        int product;  // multiplier x multiplicand
        System.out.println("Multiplication table for "+table);
        for(int value=1; value<=upto; value++) {
            product = table * value;
            System.out.println(table+" x "+value+" = "+product);
        }
    }
}
```

Multiplication Tables Revisited

```
        outputTables(maxtable, maxvalue);
    }

    // output multiplication tables
    static void outputTables(int uptotable, int uptovalue) {
        for(int table=1; table<=uptotable; table++) {
            outputTable(table, uptovalue);
        }
    }

    // output a single multiplication table
    static void outputTable(int table, int upto) {
        int product;  // multiplier x multiplicand
        System.out.println("Multiplication table for "+table);
        for(int value=1; value<=upto; value++) {
            product = table * value;
            System.out.println(table+" x "+value+" = "+product);
        }
    }
}
```

Note that any method can call any other method!

Multiplication Tables Revisited

Now let's add to the behaviour of the program:

ArithmeticTables.java

```
public class ArithmeticTables {
    public static void main(String[] args) {
        // variable declarations
        int maxtable; // largest table
        int maxvalue; // largest argument

        // get input from user
        Scanner scan = new Scanner(System.in);
        System.out.println("What number would you like
                      to produce tables up to?");
        maxtable = scan.nextInt();
        System.out.println("What is the maximum
                      argument for each table?");
        maxvalue = scan.nextInt();

        outputTables(maxtable, maxvalue); // print out tables
```

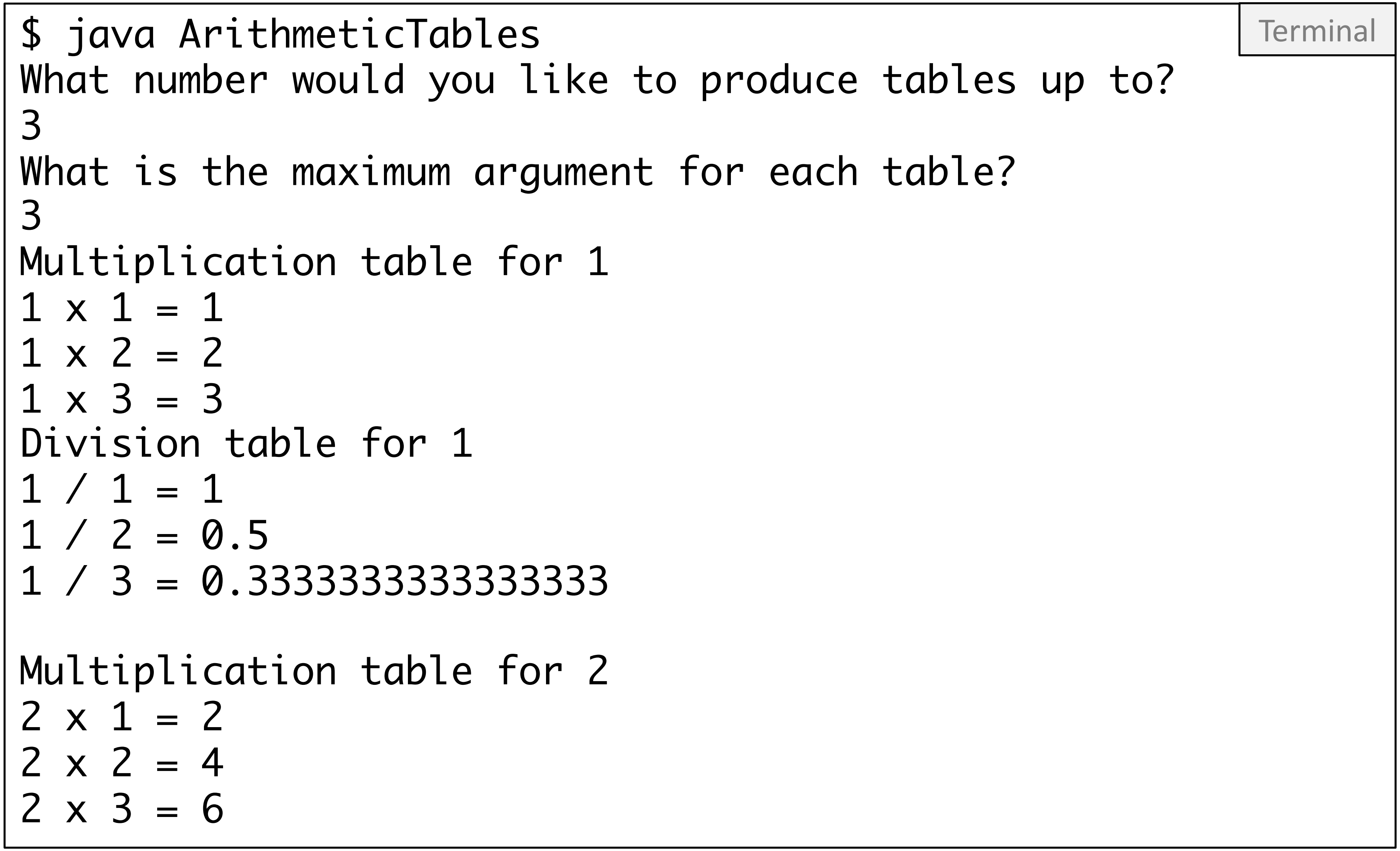
Multiplication Tables Revisited

```
    // output arithmetic tables
    static void outputTables(int uptotable, int uptovalue) {
        for(int table=1; table<=uptotable; table++) {
            outputMultTable(table, uptovalue);
            outputDivisionTable(table, uptovalue);
        }
    }

    // output a single multiplication table
    static void outputMultTable(int table, int upto) { ... }

    // output a single division table
    static void outputDivisionTable(int table, int upto) {
        double quotient;  // dividend / dividor
        System.out.println("Division table for "+table);
        for(int value=1; value<=upto; value++) {
            quotient = table / (double) value;
            System.out.println(table+" / "+value+" = "+product);
        }
    }
}
```

Multiplication Tables Revisited



```
$ java ArithmeticTables
What number would you like to produce tables up to?
3
What is the maximum argument for each table?
3
Multiplication table for 1
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
Division table for 1
1 / 1 = 1
1 / 2 = 0.5
1 / 3 = 0.3333333333333333

Multiplication table for 2
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
```

Multiplication Tables Revisited

What does this example demonstrate?

- Methods can break up code into manageable chunks, making a program more readable and maintainable
- They make it easier to add new functionality, without disturbing the functionality that's already there
- They are also useful for separating **interface** code from **implementation** code (more on this in SD2)
- In short, they are an important aspect of software development

Next Lecture

- Multi-dimensional arrays
- Passing arrays to/from methods

Tutorial 3 and Lab 4

- Array and Method exercises

Summary

- Sub-programs are used to break up a program into smaller chunks, and allow code to be reused
- Sub-programs are known as **methods** in Java
- Methods can be passed one or more arguments
- Methods can return exactly one value
- The **method signature** specifies the arguments and the return type