APPinc.
Making Technology Reachable

Java Programming

RITA M. BARRIOS, Ph.D.

PRES. TRAINING & EDUCATION

Week One

Objectives – By Day

- Intro to Java and the Eclipse Luna IDE
- Programming 101.5
- Objects
- **Lists, logic and looping**
- Classes and Methods

Exercise 1: Day Counter.java – Pg 82

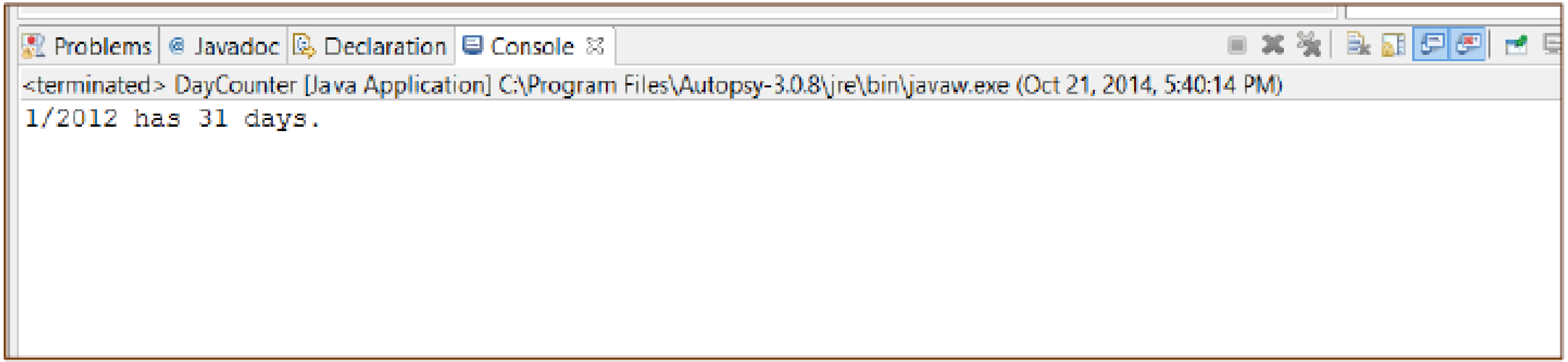
DayCounter application takes two arguments, month and year, then displays the number of days in that month

Use the switch statement, an if statement/else statements

Run as Java application (We'll use command line in a moment)

(note: the code is getting too complex to display here – please refer to the text)

DayCounter.java - Output



```
<terminated> DayCounter [Java Application] C:\Program Files\Autopsy-3.0.8\jre\bin\javaw.exe (Oct 21, 2014, 5:40:14 PM)
1/2012 has 31 days.
```

Let’s look at the code

We use command line arguments here

- public static void main(String[] **arguments**)
- The first is the month (1-12)
- The second is the year (CCYY)
- If we omit the command line arguments, it will use 1/2012

Lines 13 -40: Switch Statement to count the days in the month

```
static int countDays(int month, int year) {
       int count = -1;
       switch (month) {
           case 1:
           case 3:
           case 5:
           case 7:
           case 8:
           case 10:
           case 12:
               count = 31;
               break;
           case 4:
           case 6:
           case 9:
           case 11:
               count = 30;
               break;
           case 2:
               if (year % 4 == 0)
                   count = 29;
               else
                   count = 28;
               if ((year % 100 == 0) & (year % 400 != 0))
                   count = 28;
       }
       return count;
   }
```

DayCounter.java – Command Line

Right click on DayCounter.java

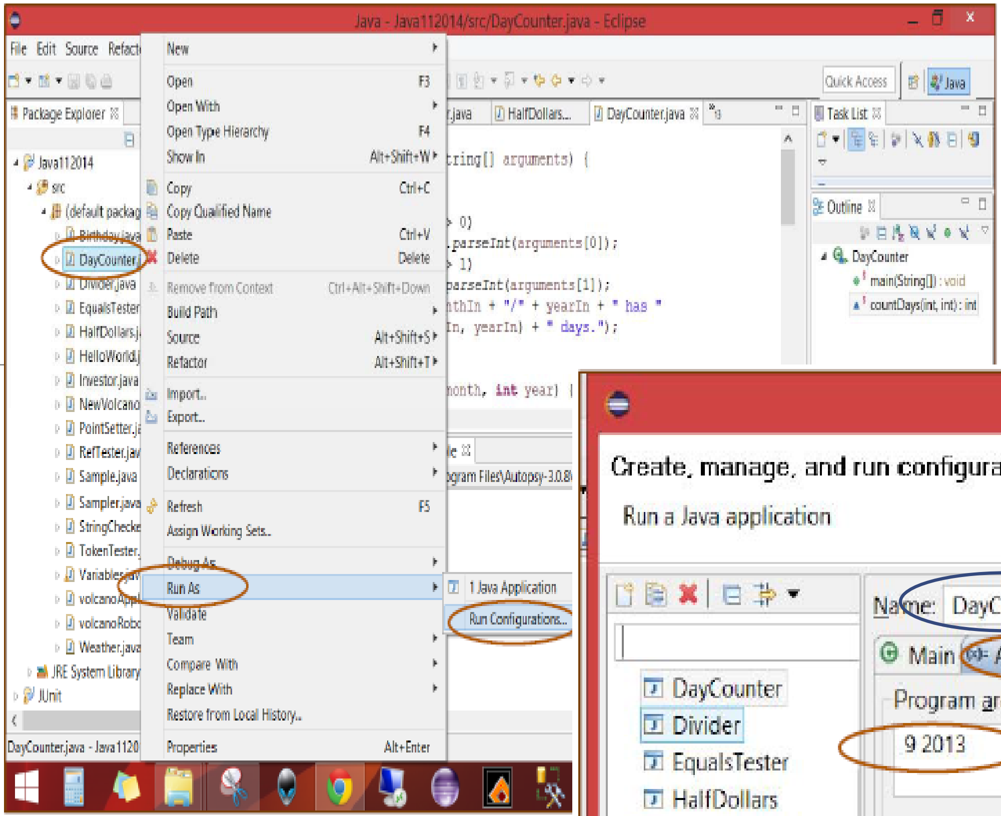
Select Run As

Run Configurations

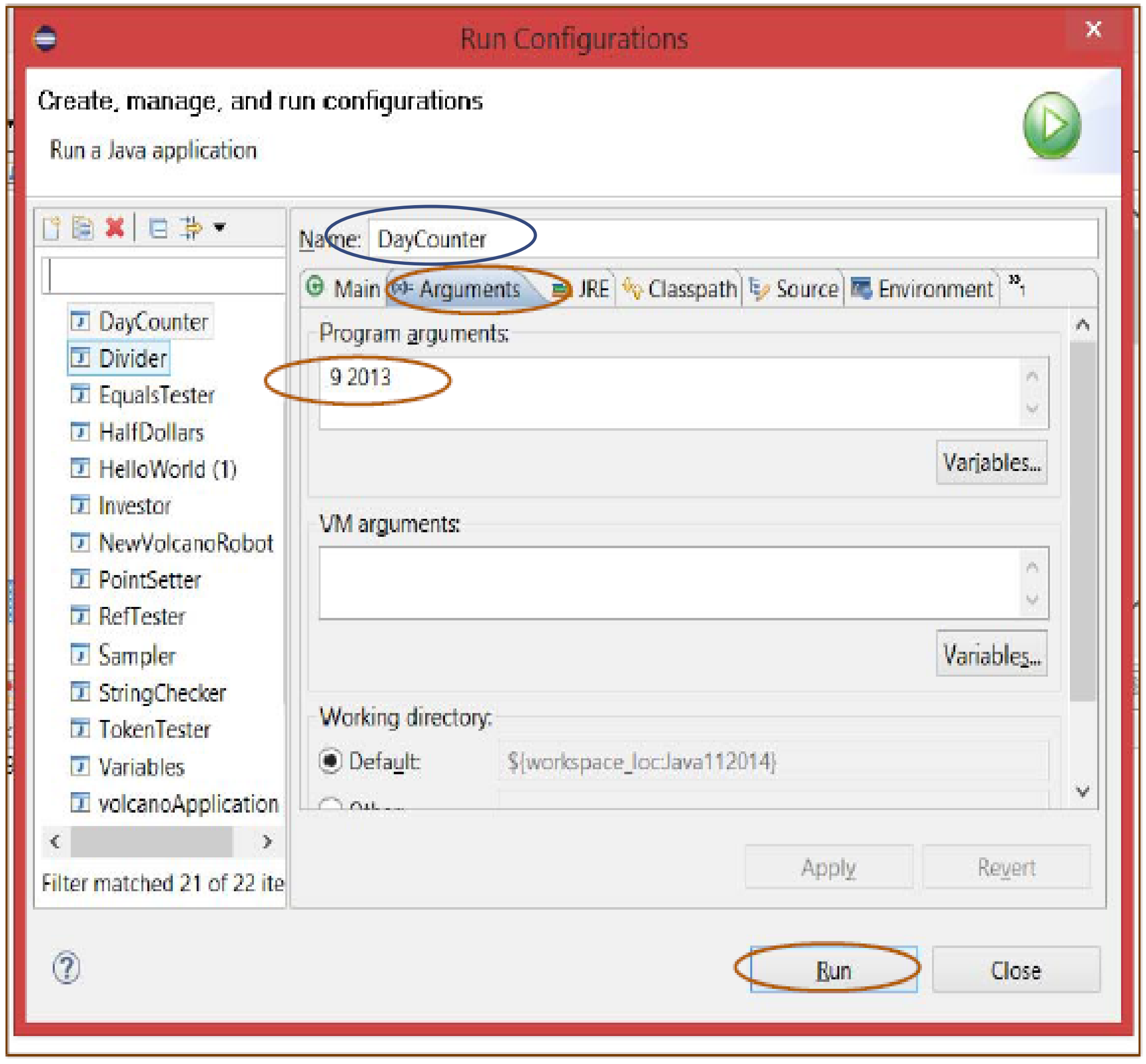
Select the Arguments Tab

Enter a Month and full year (CCYY) separated by a space in the arguments field

Click Run



```
tring[] arguments) {
 0)
.parseInt(arguments[0]);
 1)
parseInt(arguments[1]);
nthIn + "/" + yearIn + " has "
In, yearIn) + " days.");
month, int year) {
```



Name: DayCounter

Program arguments: 9 2013

VM arguments:

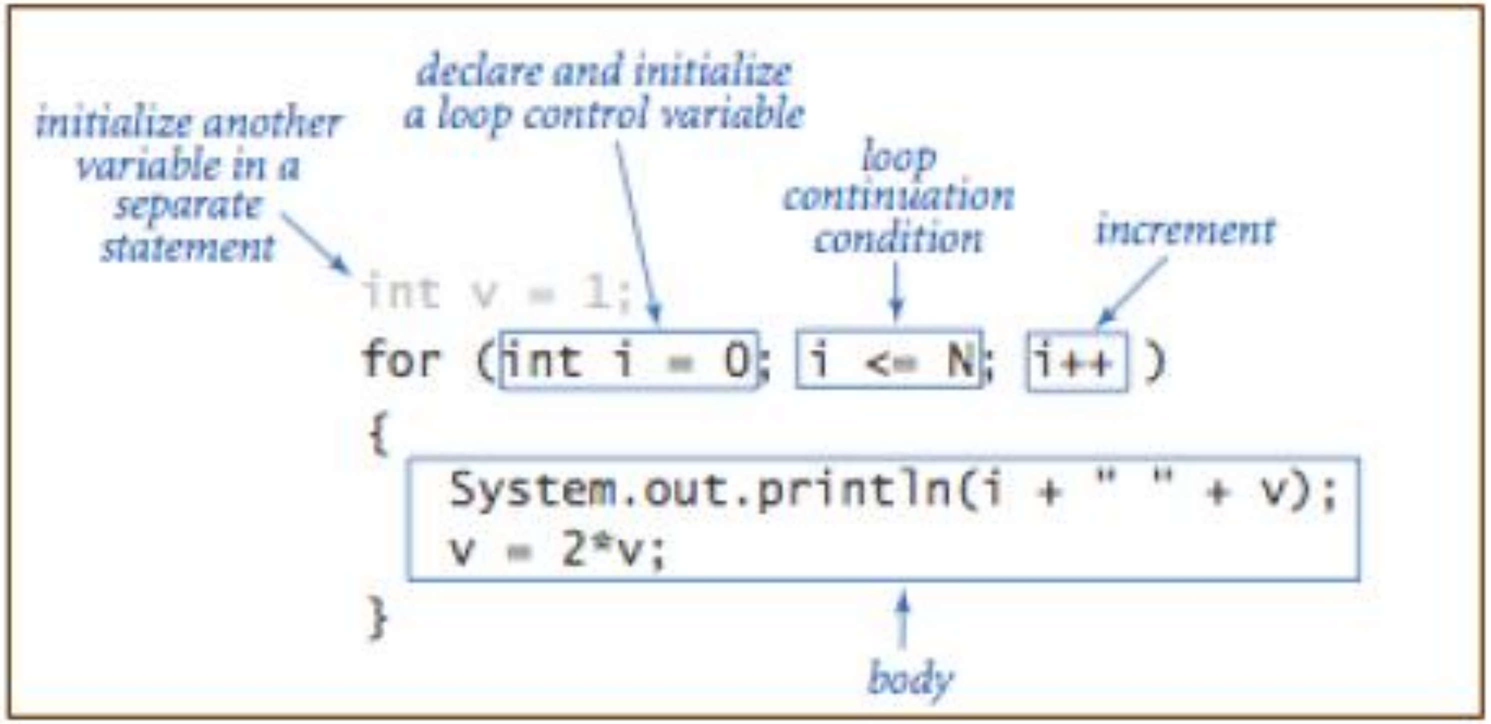
Working directory: Default: ${workspace_loc:Java112014}

For Loops

For Loops are used to repeat a set of statements until a certain condition is reached

Can be used for just about any kind of loop

Takes on the following structure



```
int v = 1;
for (int i = 0; i <= N; i++ )
{
   System.out.println(i + " " + v);
   v = 2*v;
}
```

initialize another variable in a separate statement
declare and initialize a loop control variable
loop continuation condition
increment
body

The *for* Loop

Ideal when you know the number of iterations to perform before the loop begins

Examples:

- Find the sum of 5 numbers
- Find the maximum of 20 numbers
- Print the odd numbers from 1 to 10

The for Statement

A *for statement* has the following syntax:

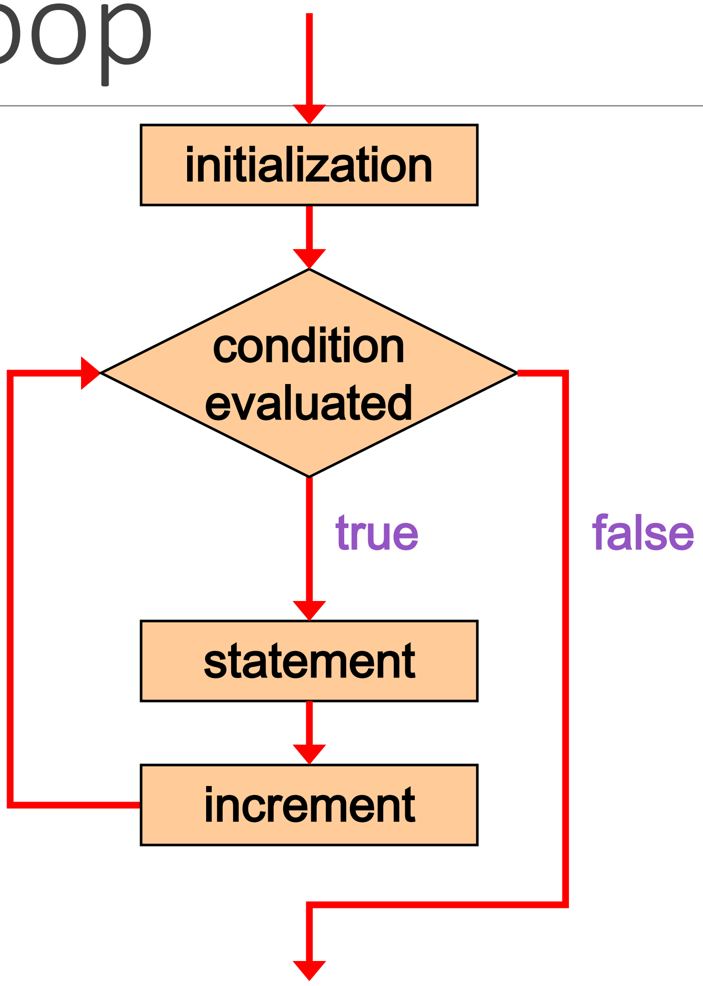
The `initialization` is executed once before the loop begins

The `statement` is executed until the `condition` becomes false

```
for ( initialization ; condition ; increment )
    statement;
```

The `increment` portion is executed at the end of each iteration

Logic of a for loop



The for Statement

An example of a `for` loop:

```
for (int count=1; count <= 5; count++)
   System.out.println (count);
```

- The initialization section can be used to declare a variable
- Like a while loop, the condition of a for loop is tested prior to executing the loop body
- Therefore, the body of a for loop will execute zero or more times

The for Statement

The increment section can perform any calculation

```
for (int num=100; num > 0; num -= 5)
   System.out.println (num);
```

- A for loop is well suited for executing statements a specific number of times that can be calculated or determined in advance
- Each expression in the header of a for loop is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
- If the increment is left out, no increment operation is performed

`for` loop Exercises: How many times is the loop body repeated?

-
  ```
  for (int x = 3; x <= 15; x++)
   System.out.println(x);
  ```

-
  ```
  for (int x = 1; x <= 5; x++)
   System.out.println(x);
  ```

-
  ```
  for (int x = 12; x >= 2; x++)
   System.out.println(x);
  ```

Write the `for` statement that print the following sequences of values.

- 1, 2, 3, 4, 5, 6, 7
- 3, 8, 13, 18, 23
- 20, 14, 8, 2, -4, -10
- 19, 27, 35, 43, 51

Loop Order of Operation

First Iteration

- ✓ initialization
- ✓ Test
- ✓ statements
  - ✓ increment

Second

  - ✓ test
  - ✓ Statements
  - ✓ Increment

Third

  - ✓ test

```
for ( initialization; test; increment )
   statement
```

How many times is the loop body repeated?

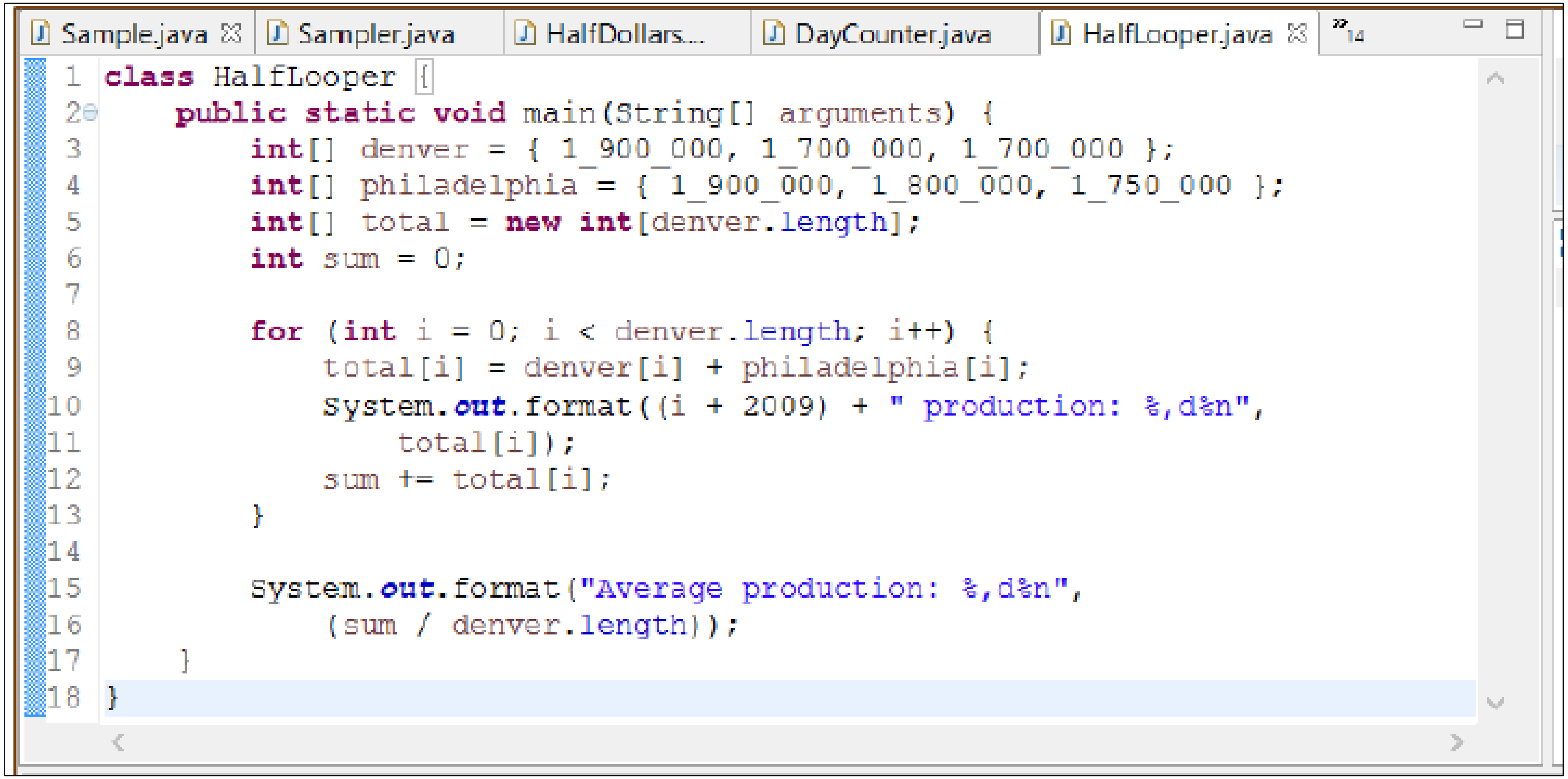
```
1. for (int i = 10; i < 0; i++)
2. for (int i = 0; i < 10; i--)
```

Exercise 1: HalfLooper.java – Pg.93

In this exercise, we rewire the HalfDollar.java application to use loops in an attempt to remove the redundant code

The original used an array that is only three elements – the new version is shorter and more flexible in that it doesn’t matter how many elements we have and will return the same output

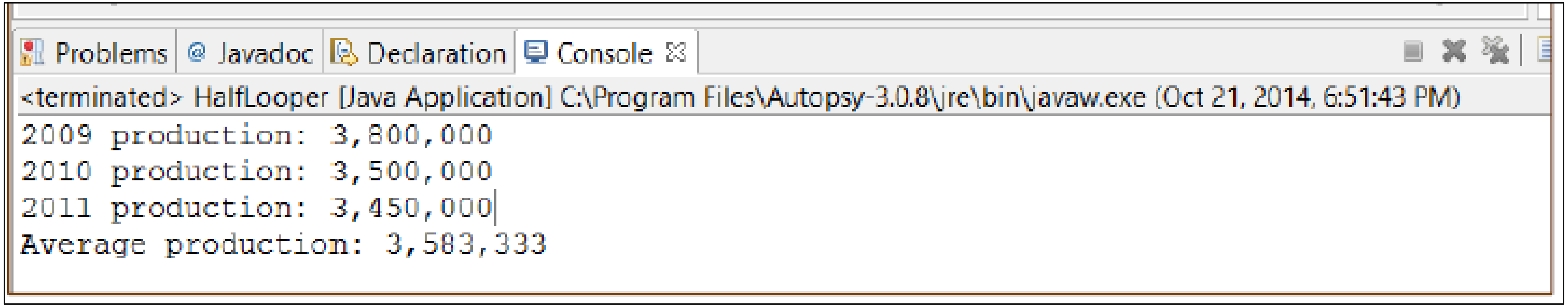
Run as a Java Application



```java
class HalfLooper {
    public static void main(String[] arguments) {
        int[] denver = { 1_900_000, 1_700_000, 1_700_000 };
        int[] philadelphia = { 1_900_000, 1_800_000, 1_750_000 };
        int[] total = new int[denver.length];
        int sum = 0;

        for (int i = 0; i < denver.length; i++) {
            total[i] = denver[i] + philadelphia[i];
            System.out.format((i + 2009) + " production: %,d%n",
                total[i]);
            sum += total[i];
        }

        System.out.format("Average production: %,d%n",
            (sum / denver.length));
    }
}
```

Halflooper.java

HalfLooper.java – Output – Pg 93



```
<terminated> HalfLooper [Java Application] C:\Program Files\Autopsy-3.0.8\jre\bin\javaw.exe (Oct 21, 2014, 6:51:43 PM)
2009 production: 3,800,000
2010 production: 3,500,000
2011 production: 3,450,000
Average production: 3,583,333
```

Let’s examine the code

Instead of going thru each element in an array, we spin through them using a for loop

Line 8 – create the loop with an int variable called i (short for index) which is the counter – increment i by 1 for each pass of the loop then stop when it is greater than the length.denver (total number of elements) in the array

Lines 9 – 11: set the value of one of the elements

Line 12: add the value of total element to the sum variable to calculate the total production

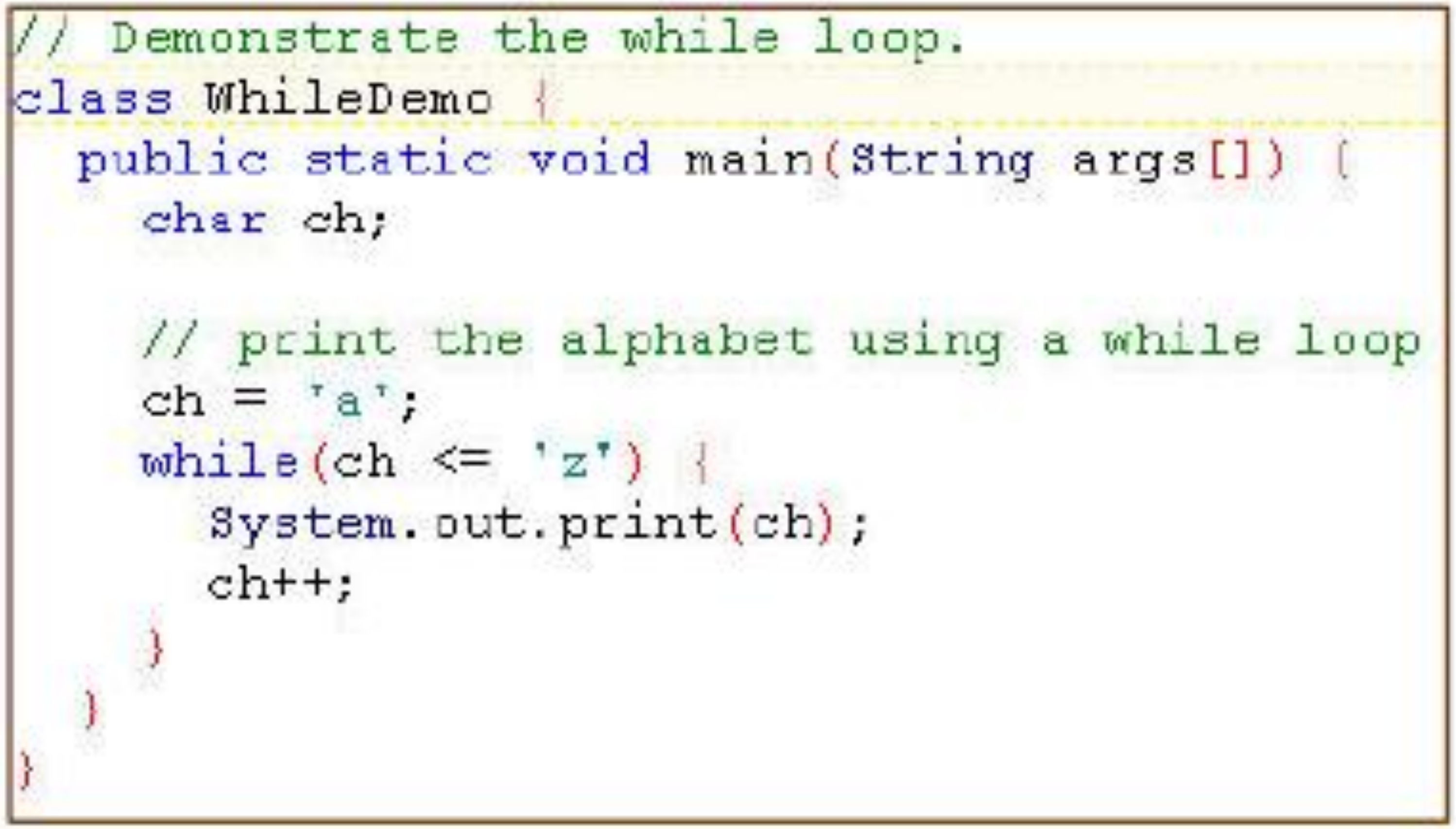
Notes on Loops

When we spin thru an array using a loop structure, we call it "iterating over the array"

Use this same process of lists, hash maps, and other data collections

While Loops

Execute a block repeatedly **while** a specific condition exists (remains true)



```
// Demonstrate the while loop.
class WhileDemo {
  public static void main(String args[]) {
    char ch;

    // print the alphabet using a while loop
    ch = 'a';
    while(ch <= 'z') {
      System.out.print(ch);
      ch++;
    }
  }
}
```

Exercise 1: ArrayCopier.java – Pg. 94

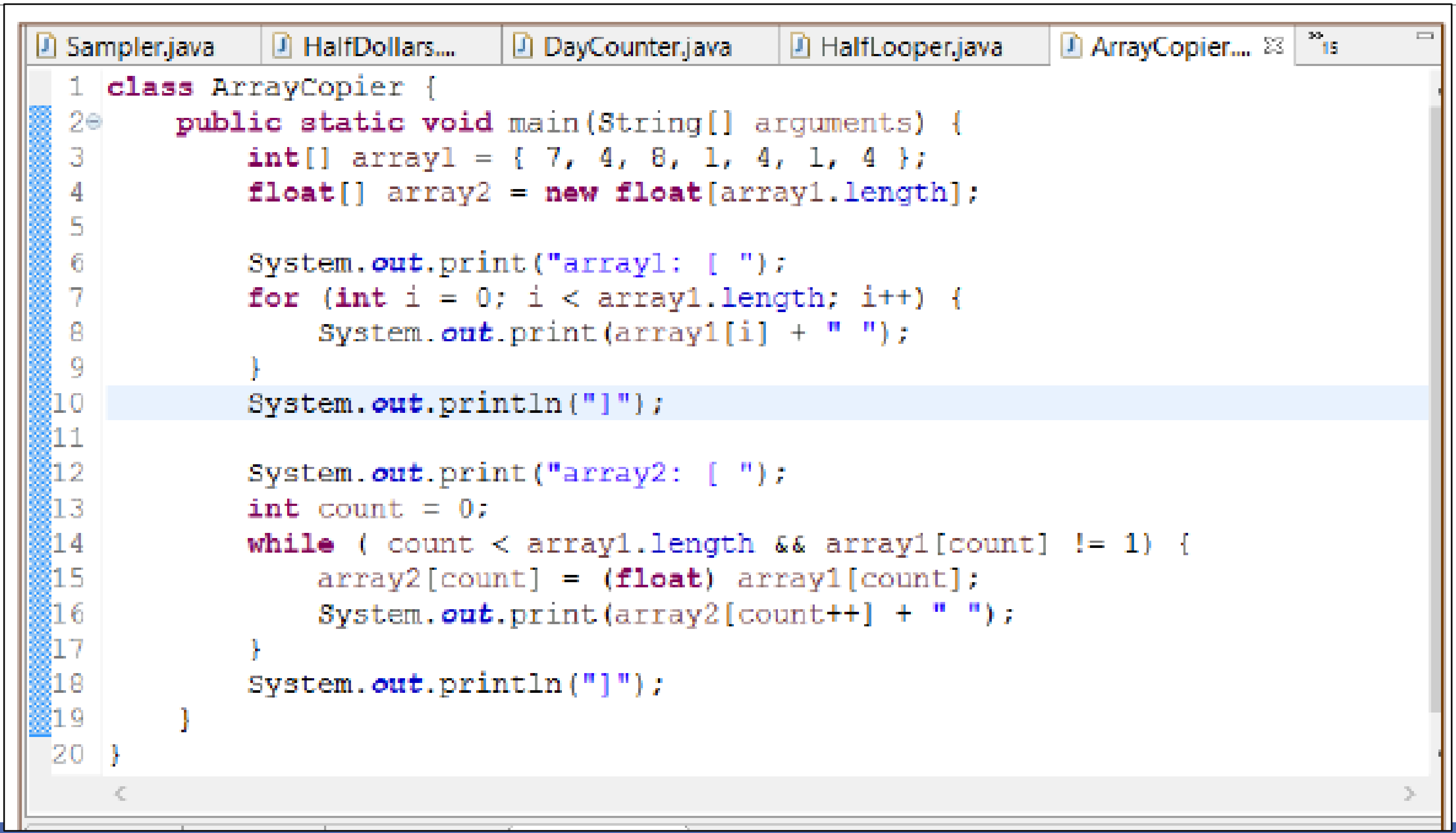
In this exercise, we use a while loop to copy the elements of an array of integers, array1, to an array of float variables in array2.

We cast each element to a float as we go. If any element in array1 is equal to 1, the loop should immediately terminate at that point and drop out of the construct

Save the file as ArrayCopier.java

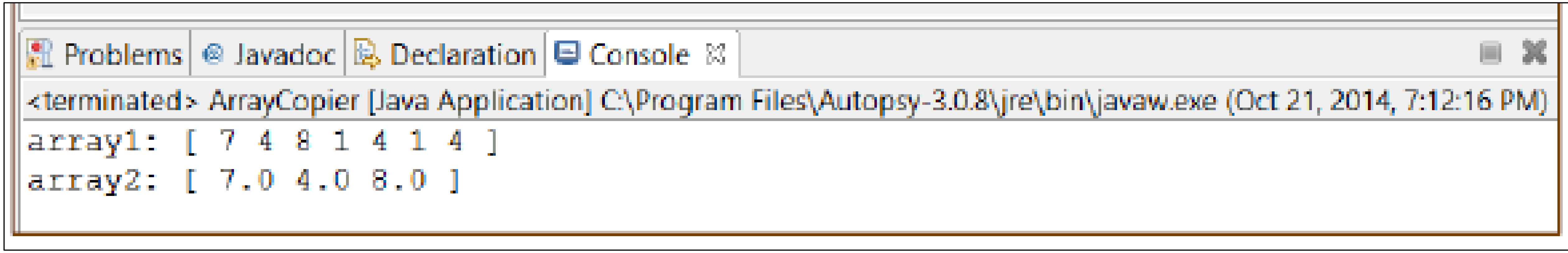
Run as a Java Application

ArrayCopier.java – Page 94



```java
class ArrayCopier {
    public static void main(String[] arguments) {
        int[] array1 = { 7, 4, 8, 1, 4, 1, 4 };
        float[] array2 = new float[array1.length];

        System.out.print("array1: [ ");
        for (int i = 0; i < array1.length; i++) {
            System.out.print(array1[i] + " ");
        }
        System.out.println("]");

        System.out.print("array2: [ ");
        int count = 0;
        while ( count < array1.length && array1[count] != 1) {
            array2[count] = (float) array1[count];
            System.out.print(array2[count++] + " ");
        }
        System.out.println("]");
    }
}
```

ArrayCopier.java - Output



```
<terminated> ArrayCopier [Java Application] C:\Program Files\Autopsy-3.0.8\jre\bin\javaw.exe (Oct 21, 2014, 7:12:16 PM)
array1: [ 7 4 8 1 4 1 4 ]
array2: [ 7.0 4.0 8.0 ]
```

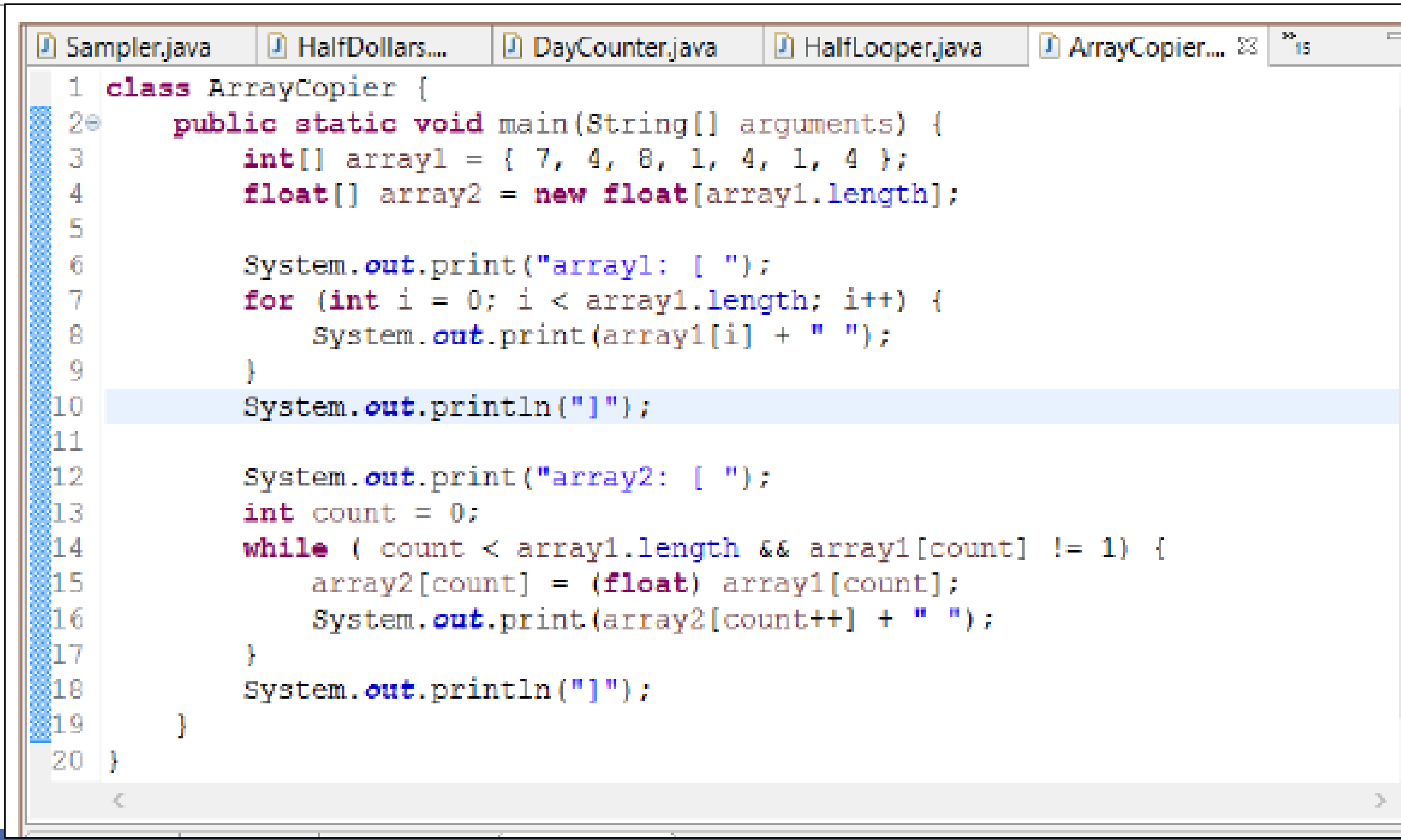
ArrayCopier.java – Examine the Code

Lines 3 – 4: Declare the array

Lines 6 – 10: output: iterates over array1 using the for loop to print its values

Lines 12 – 18: block assigns the values to array2 while converting and prints them out

Line 14: keeps track of running out of elements in array1 and encountering 1 in array1



```
 1 class ArrayCopier {
 2     public static void main(String[] arguments) {
 3         int[] array1 = { 7, 4, 8, 1, 4, 1, 4 };
 4         float[] array2 = new float[array1.length];
 5 
 6         System.out.print("array1: [ ");
 7         for (int i = 0; i < array1.length; i++) {
 8             System.out.print(array1[i] + " ");
 9         }
10         System.out.println("]");
11 
12         System.out.print("array2: [ ");
13         int count = 0;
14         while ( count < array1.length && array1[count] != 1) {
15             array2[count] = (float) array1[count];
16             System.out.print(array2[count++] + " ");
17         }
18         System.out.println("]");
19     }
20 }
```

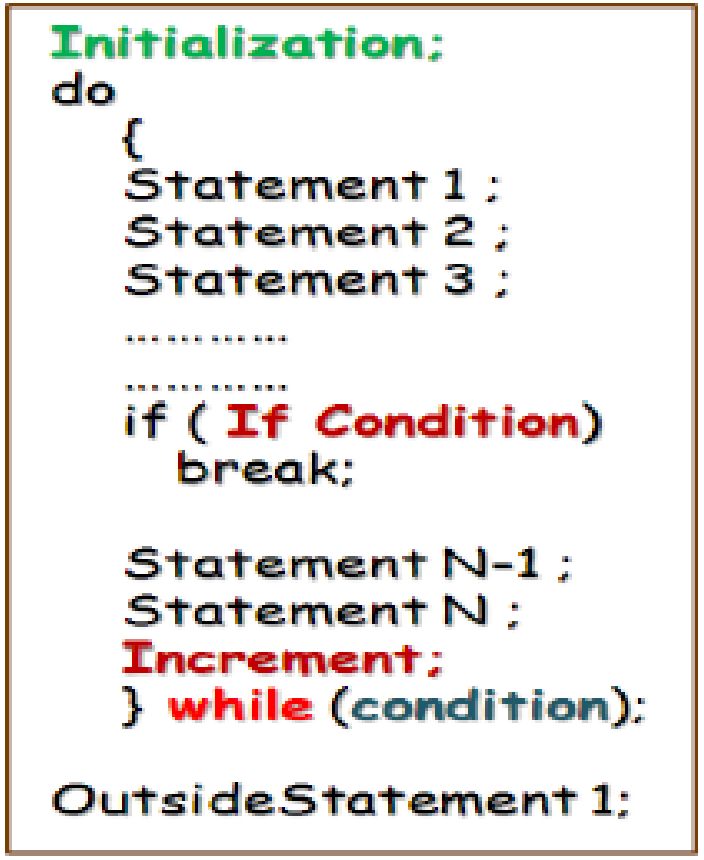
Statements **break** and **continue** allow us to break out of a loop early – Break drops out of the structure; continue starts the loop over with the next iteration

Do While Loops

Do something while the condition exists (is true)

Will execute the body of the loop at least once before testing the condition – even if the condition is false at the outset

- While will always check first then decide whether or not to execute the body



```
Initialization;
do
   {
   Statement 1 ;
   Statement 2 ;
   Statement 3 ;
   …………
   …………
   if ( If Condition)
     break;

   Statement N-1 ;
   Statement N ;
   Increment;
   } while (condition);

OutsideStatement 1;
```

Looping

In computing, we often need to perform the same operations on multiple items.

Typically, these tasks follow this pattern:

- initialize values (set total to 0)
- process items one at a time (add price to total)
- report results (report total)

The flow of control that programmers use to complete jobs with this pattern is called **looping**, or **repetition**.

Some Definitions

iration

- one execution of the loop body

loop update

- One or more statements that could cause the loop condition to evaluate to *false* (to end the looping)

loop termination condition

- the event that causes the loop condition to evaluate to *false*

The Endless Loop

- Also called an **infinite loop**
- If the loop condition never evaluates to *false,* the loop body is executed continuously, without end
- If the loop body has no output, the endless loop makes the computer appear to hang.
- If the loop body produces output, the endless loop results in that output being repeatedly written without end.
- Aborting the program will interrupt the endless loop.

Looping Techniques

There are standard patterns and techniques for performing these common operations:

- Accumulation
- Counting Items
- Finding an Average
- Finding Maximum or Minimum Values

Accumulation

Approach: <u>The Running Total</u>

- We start by initializing a total variable to 0.
- Each time we read a value, we add it to the total.
- When we have no more values to read, the total is complete.

Counting Items

Approach: <u>THE RUNNING COUNT</u>

- We start by initializing a count variable to 0.
- Each time we read a value, we check whether that value meets the criteria as something we want to count. If so, we increment the count variable by 1.
- When we are finishing reading values, the count is complete.

Calculating an Average

Approach: <u>Combine Accumulation And Counting</u>

We start by initializing a total variable and count variable to 0.

Each time we read an item, we add its value to the total variable and increment the count variable

When we have no more items to read, we calculate the average by dividing the total by the count of items.

Finding Maximum/Minimum Values

Approach: <u>The Running Maximum Or Minimum</u>

For the maximum (minimum is similar):

- Read the first item and save its value as the current maximum
- Each time we read a new value, we compare it to the current maximum.
  - If the new value is greater than the current maximum, we replace the current maximum with the new value.
- When we have no more items to read, the current maximum is the maximum for all values.

Using a Loop Control Variable

A **loop control variable** is usually used for counting.

- We set its initial value in the initialization statement
- We check its value in the loop condition
- We increment or decrement its value in the loop update statement

Exercise - How many times is the following loop body repeated? What is printed during each repetition of the loop body and after exit?

```
x = 3;
for (int count = 0; count < 3; count++)
{
 x = x * x;
 System.out.println(x);
}
System.out.println(x);
```

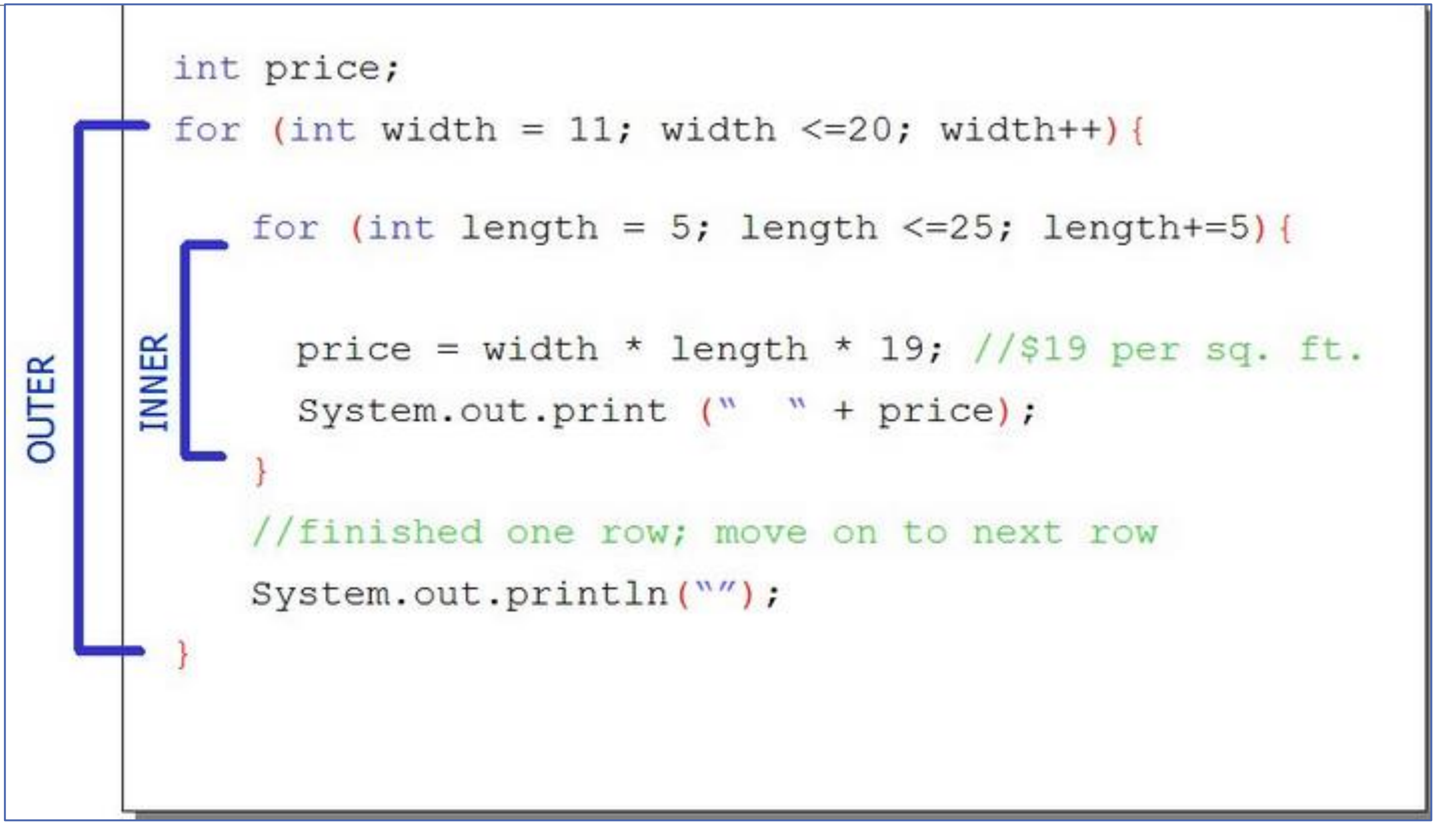
Exercise: What mathematical result does the following fragment compute and display?

```
System.out.print("Enter x: ");
int x = scan.nextInt();

System.out.print("Enter y: ");
int yFirst = scan.nextInt();

int product = 1;

for (int y = yFirst; y > 0; y--){
     product *= x;
     System.out.println(“result = “ + product);
}
```

Nested Loops

Loops can be nested inside other loops; that is, the body of one loop can contain another loop.

for (initialization; test; increment)
  statement



```
int price;
for (int width = 11; width <=20; width++){
    for (int length = 5; length <=25; length+=5){

      price = width * length * 19; //$19 per sq. ft.
      System.out.print ("  " + price);
    }
    //finished one row; move on to next row
    System.out.println("");
}
```

Nested loops. What do these print?
Code and run in a Java class to see what they do

```
for (int i = 1; i < 4; i++)
   for (int j = 1; j < i; j++)
      System.out.println(i + " " + j);
```

```
for (int i = 1; i < 4; i++)
   for (int j = 1; j < i; j++)
      System.out.println(i + " " + j);
      System.out.println("******");
```

```
for (int i = 0; i < 4; i++)
    for (int j = 1; j < i; j++)
      System.out.println(i + " " + j);
```

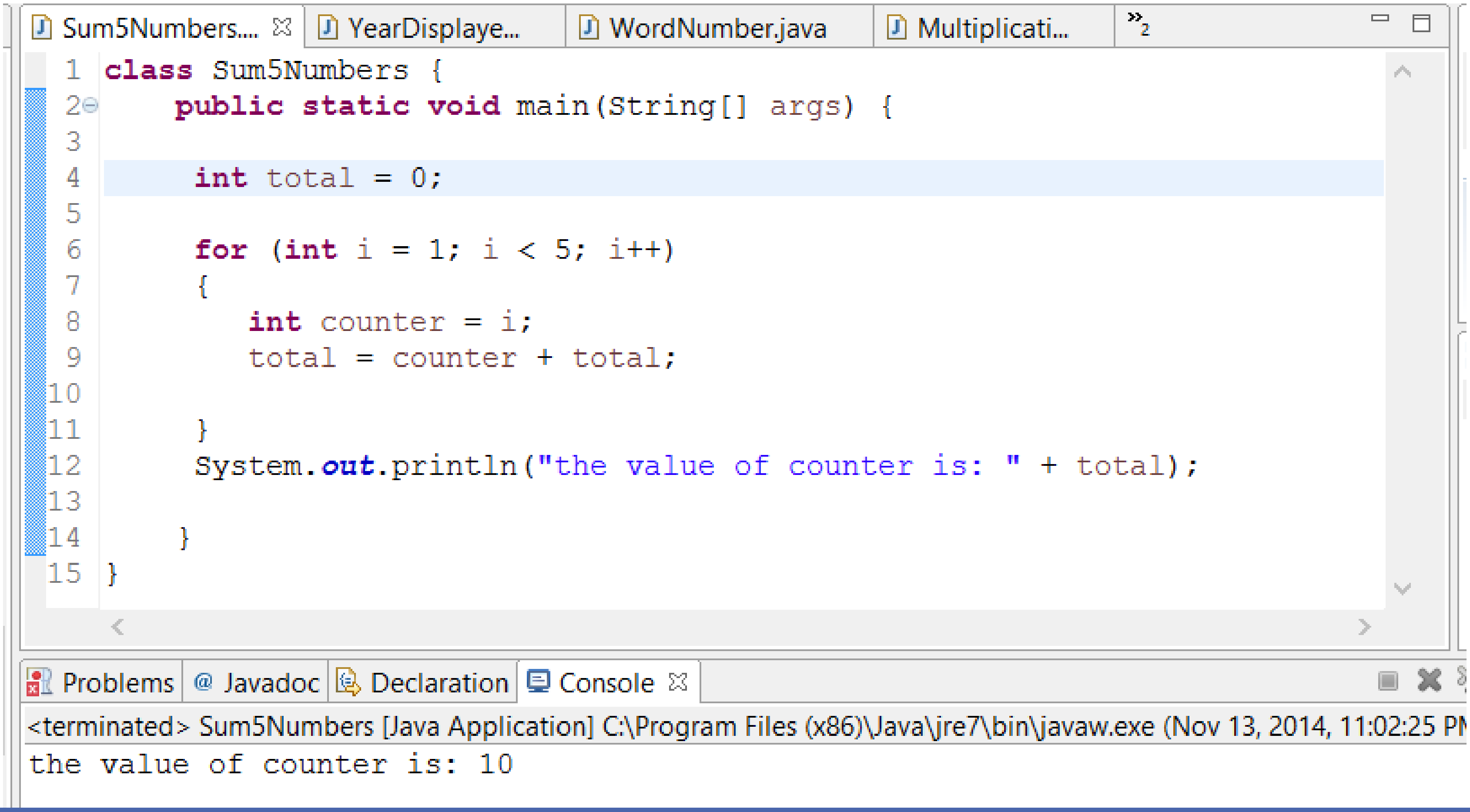
```
int T = 0;
for (int i = 1; i < 4; i++) {
    for (int j = 1; j < 2*i; j += 2)
         T += j * i;
       System.out.println("T = " + T);
}
```

Exercise: Find Sum of 5 Integers

class Sum5Numbers.java – convert the following pseudo-code (almost code) to a Java class called Sum5Number.java – run as a Java Application

```
{
        set total to 0
        for i = 1 to 5 by 1  {
            set counter = i
            add counter to total
        }
        print the total
}
```

Sum5Numbers.java Solution



```java
class Sum5Numbers {
    public static void main(String[] args) {

     int total = 0;

     for (int i = 1; i < 5; i++)
     {
        int counter = i;
        total = counter + total;

     }
     System.out.println("the value of counter is: " + total);

    }
}
```

```
<terminated> Sum5Numbers [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (Nov 13, 2014, 11:02:25 PM
the value of counter is: 10
```

Day 5 – Classes and Methods

Java programs are made up a main class and any other classes needed to support the main()

We are going to **expand** our understanding of:

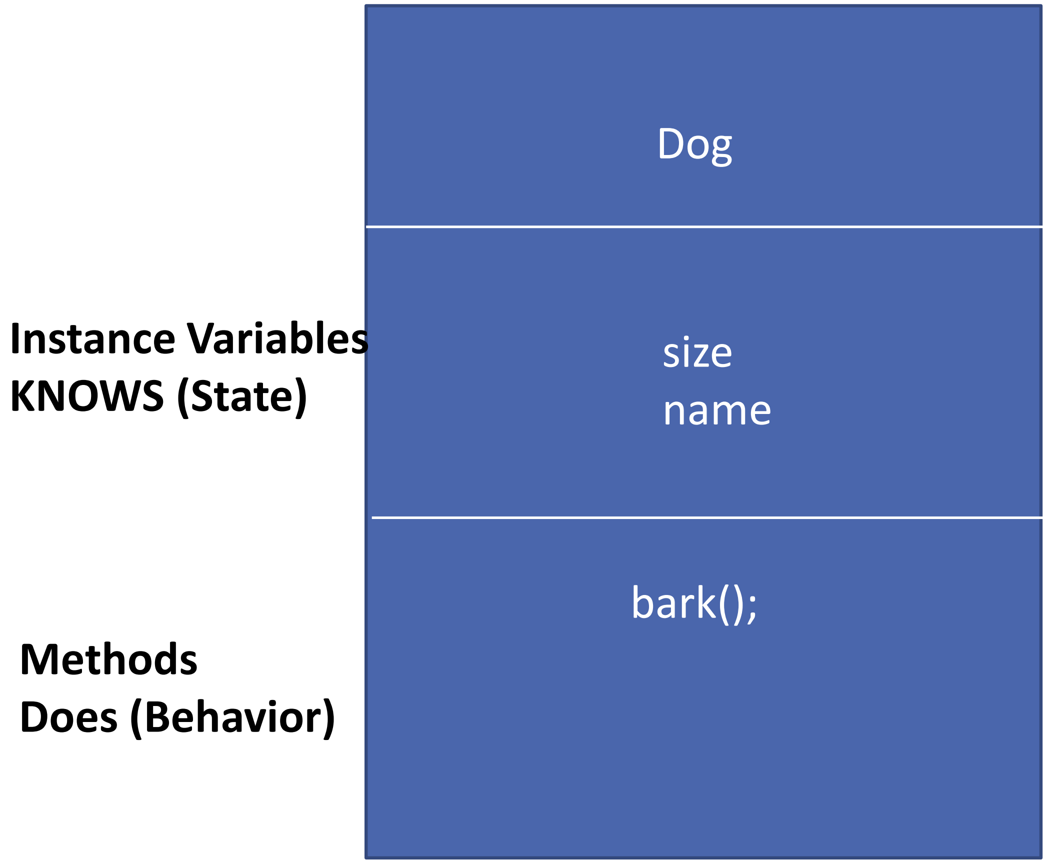
Classes and their definitions
- instance variables
- main() method
- Method creation
- Method overloading
- Constructors

Defining a Class

Reminder: A class describes what an object knows and what it can do

- Blue print for the object
- Inherit from the Object class (the super class of all classes
  - Use the extends keyword to indicate the superclass of a class

        class myReadFile extends ReadFile
- Objects can have different instance variables
- Can have same methods but each method can behave differently based on the values of the instance variables



Creating Methods

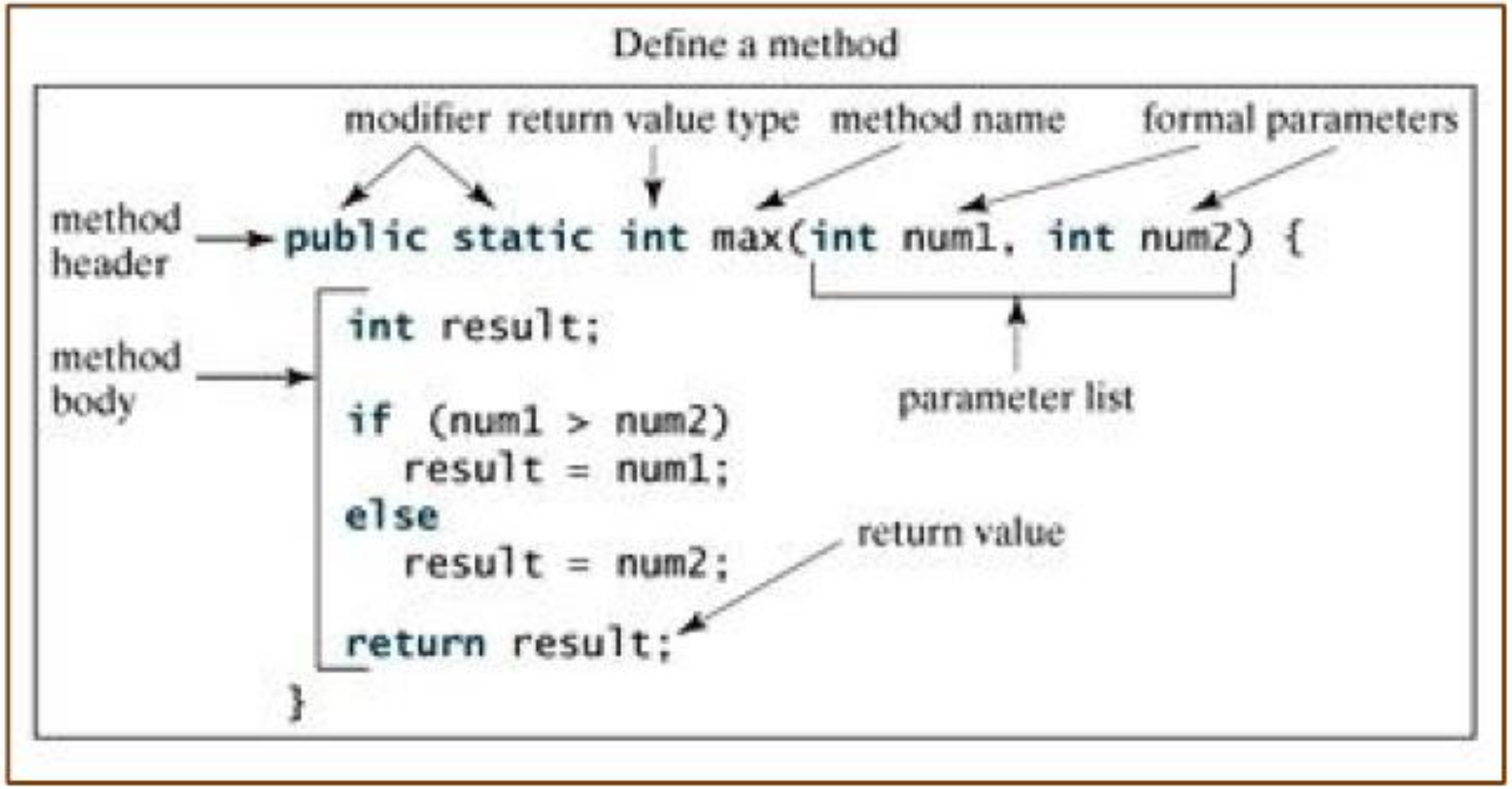
Reminder: Methods define an object's behavior

- Behavior – anything that happens when the object is created and tasks that can be performed

Defining a method

- 4 parts to a method: name, parameters, return type and the body
- Name and parameters = the **METHOD SIGNATURE**
- Two additional parts are the **modifier** and the **throws** keyword – more on this later
- When we have more than one method with the same name but different signatures (parameters) it is called **METHOD OVERLOADING**

Format of a method



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

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

   return result;
}
```

Method Creation (2)

Return Value – a primitive type, class, or value returned by the method

- Declare the method void to indicate no return value is expected

The relationship between Dogs and Bark – Create a dog.java class and DogTestDrive.java Application

Dog.java

Determine the size and display the dog bark

> 60 = Wooof! Wooof!

> 14 = Ruff! Ruff!

Everything else = Yip! Yip!

Dogtestdrive.java

Application to test the dog.java class

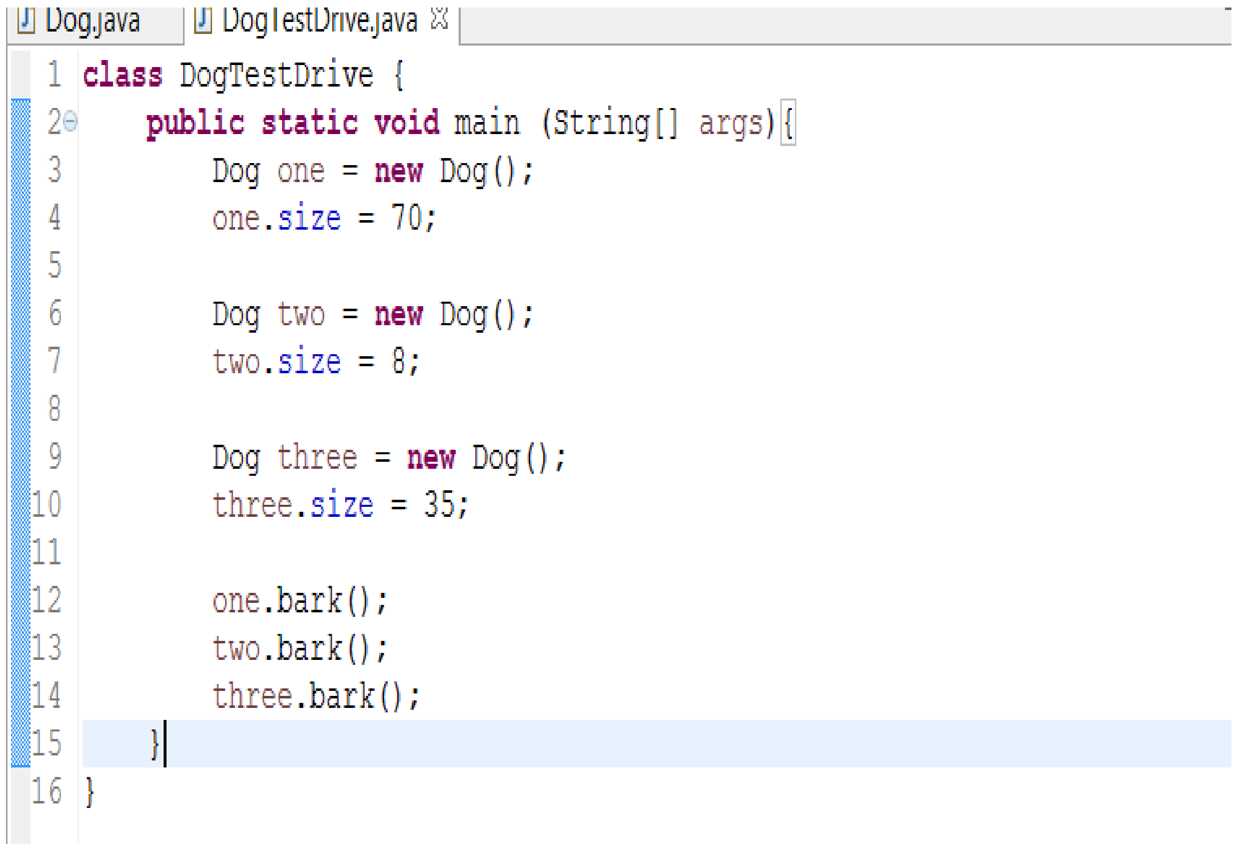
Create 3 dogs

Dog 1’s size is 70 lbs

Dog 2’s size is 8 lbs

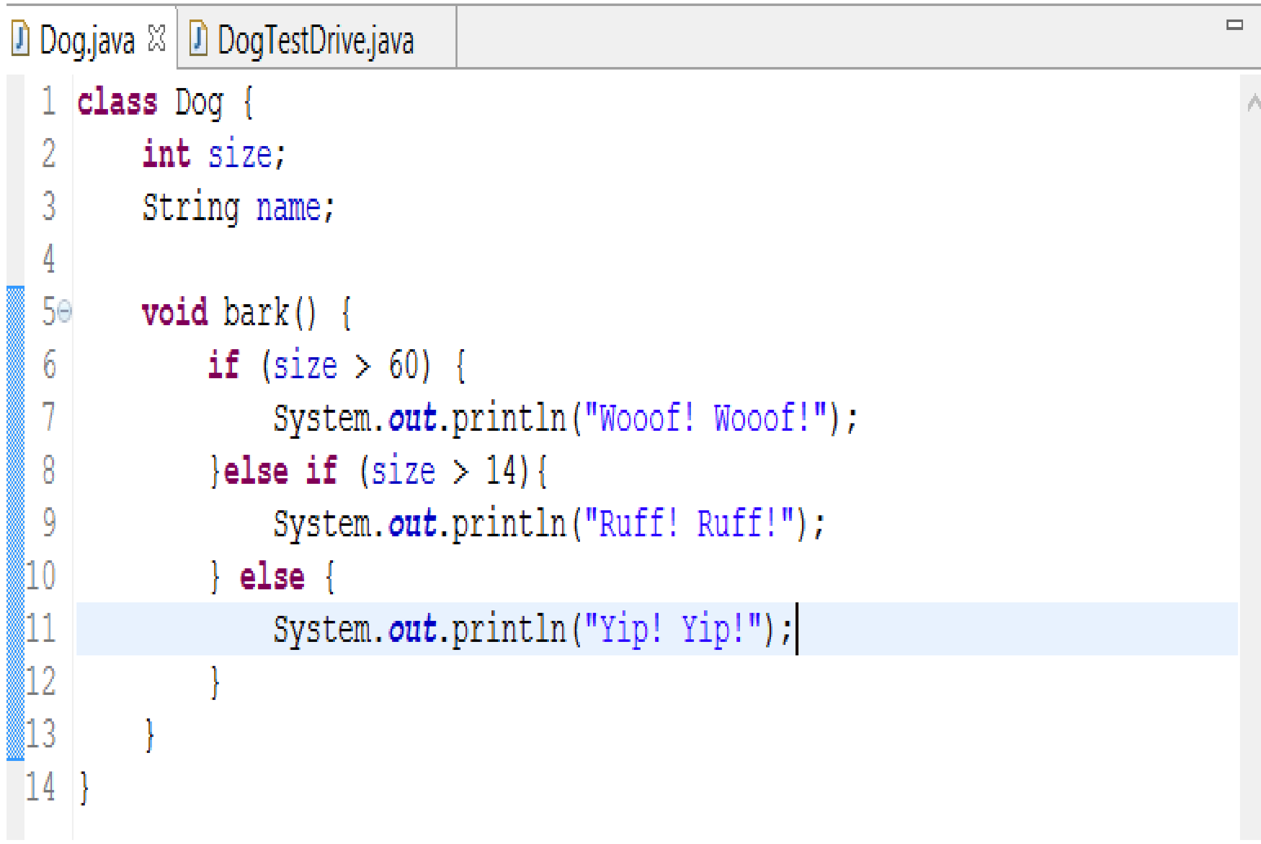
Dog 3’s size is 35 lbs

This is how a dog barks...



Dog.java DogTestDrive.java

```java
class DogTestDrive {
    public static void main (String[] args){
        Dog one = new Dog();
        one.size = 70;

        Dog two = new Dog();
        two.size = 8;

        Dog three = new Dog();
        three.size = 35;

        one.bark();
        two.bark();
        three.bark();
    }
}
```



Dog.java DogTestDrive.java

```java
class Dog {
    int size;
    String name;

    void bark() {
        if (size > 60) {
            System.out.println("Wooof! Wooof!");
        }else if (size > 14){
            System.out.println("Ruff! Ruff!");
        } else {
            System.out.println("Yip! Yip!");
        }
    }
}
```

Passing Values

We pass values into the methods (i.e.: dog.bark(3)) – To tell the Dog object how many times to bark

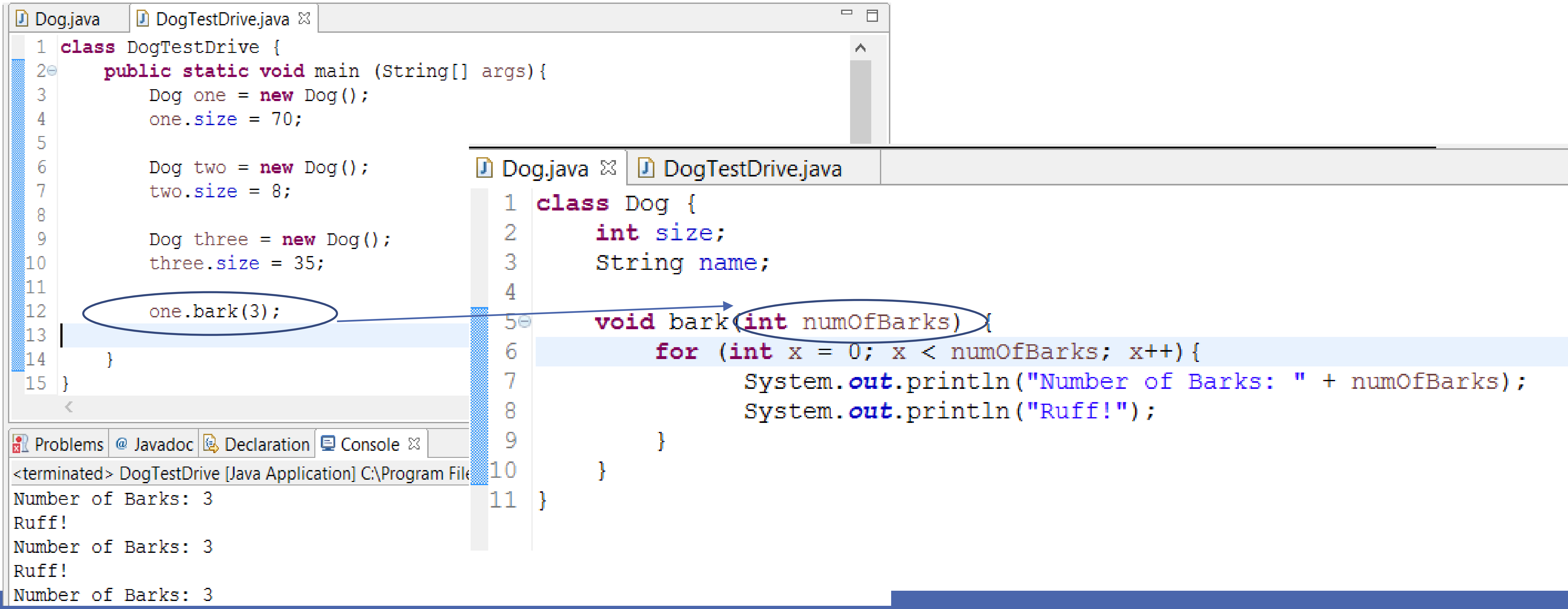
A method **uses** parameters and a caller **passes** arguments

Arguments land directly into the parameters of a method

- Parameters are local variables that is used inside the body of the method
- If a method takes a parameter, you must pass an argument of the expected type

Passing Values (2)

Let's refactor the Dog Class to match the following



```
class DogTestDrive {
    public static void main (String[] args){
        Dog one = new Dog();
        one.size = 70;

        Dog two = new Dog();
        two.size = 8;

        Dog three = new Dog();
        three.size = 35;

        one.bark(3);

    }
}
```

```
<terminated> DogTestDrive [Java Application] C:\Program File
Number of Barks: 3
Ruff!
Number of Barks: 3
Ruff!
Number of Barks: 3
```

```
class Dog {
    int size;
    String name;

    void bark(int numOfBarks) {
        for (int x = 0; x < numOfBarks; x++){
            System.out.println("Number of Barks: " + numOfBarks);
            System.out.println("Ruff!");
        }
    }
}
```

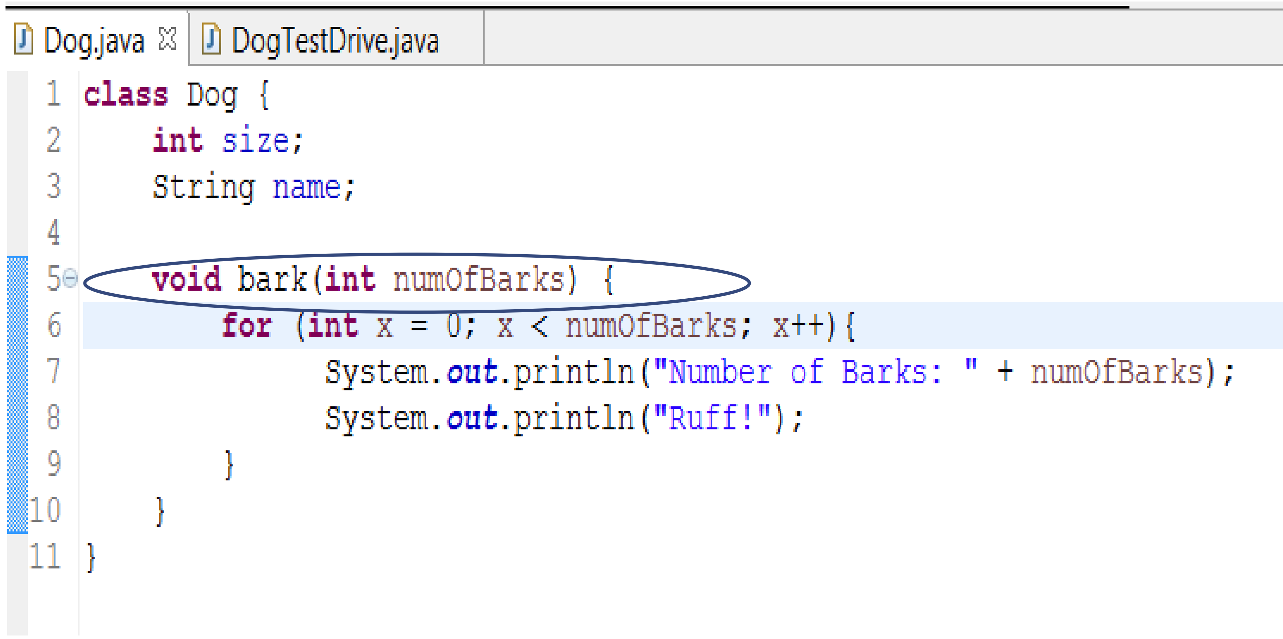
Getting Things Back from a Method

Methods can return values

- Every method is declared with a return type
- We've made our types a void type (don't send anything back)

We can declare a method to give something back to the caller:

int theSecret = life.giveSecret();
int giveSecret() {
        return 42;
}
both must be of the same type



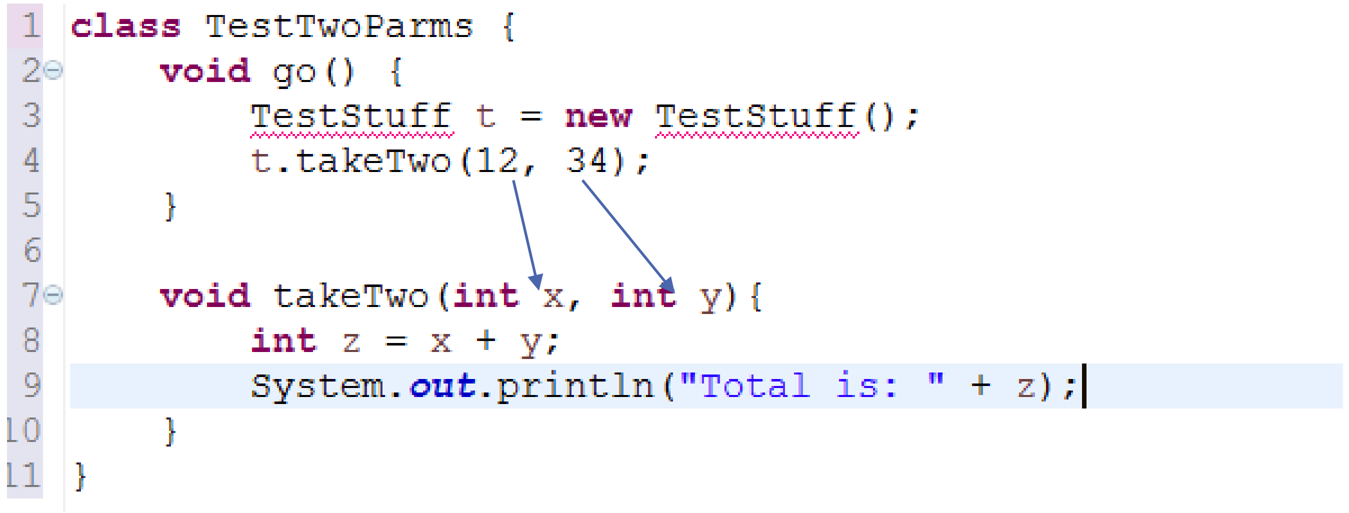
```
class Dog {
    int size;
    String name;

    void bark(int numOfBarks) {
        for (int x = 0; x < numOfBarks; x++){
              System.out.println("Number of Barks: " + numOfBarks);
              System.out.println("Ruff!");
        }
    }
}
```

Sending more than one thing

Ok to have multiple parameters in a method – just separate them with a comma

Arguments must be separated with a comma too and list the parameters in the order the method is expecting

Can use variables to pass in place of values as well



```
class TestTwoParms {
    void go() {
        TestStuff t = new TestStuff();
        t.takeTwo(12, 34);
    }

    void takeTwo(int x, int y){
        int z = x + y;
        System.out.println("Total is: " + z);
    }
}
```

Encapsulation – Hiding the data

This is a core OOP concept – the data should never be exposed to those who don't need to see it.

- This is called exposure – leaving the data to be reached with the dot operator – one.bark(3); -- this leaves the variable open to unauthorized modifications
- We hide the data with the public and private modifiers
  - Best Practice
    - Instance Variables = private
    - Getters and Setters = public

Encapsulation Exercise:

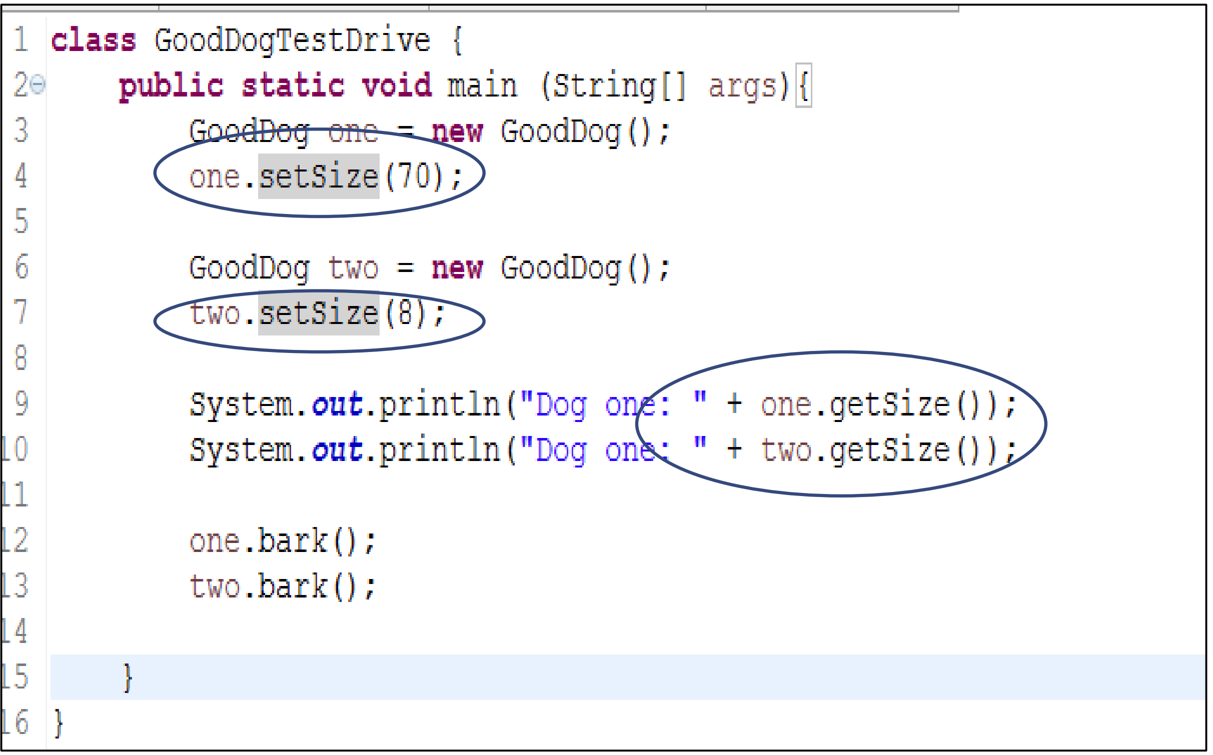
In this exercise, we will create the GoodDog.java class. The class has an attribute of **size** and methods of **getSize, setSize** and **bark()**. Since the size is an instance variable, let's make it **private**. Also, we will need to return the size back to a caller so be sure to give it a **return**.

We will then build a test application called GoodDogTestDrive.java to test the class.

To test, create two dog objects, one and two. Dog one's size is 70 and dog two's size is 8 – use the setSize() method. Print the size of each dog using the getSize() method. Then make the dogs bark using the bark() method.

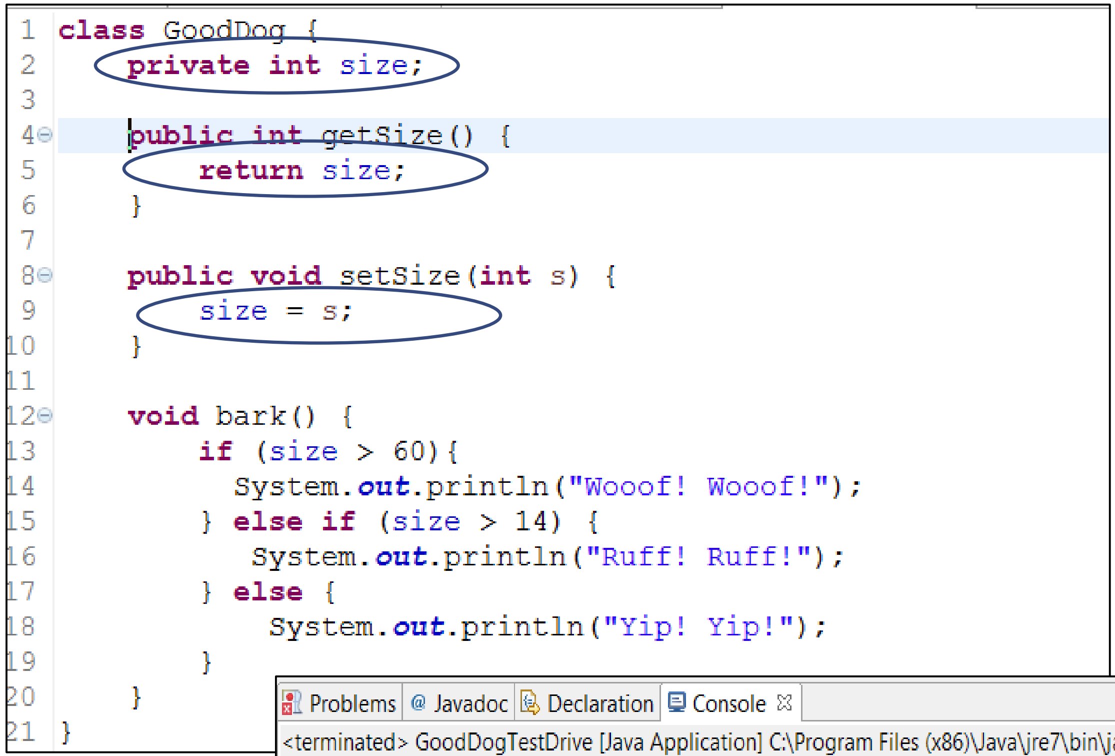
GoodDogTestDrive.java & GoodDog.java

GoodDogTestDrive.java

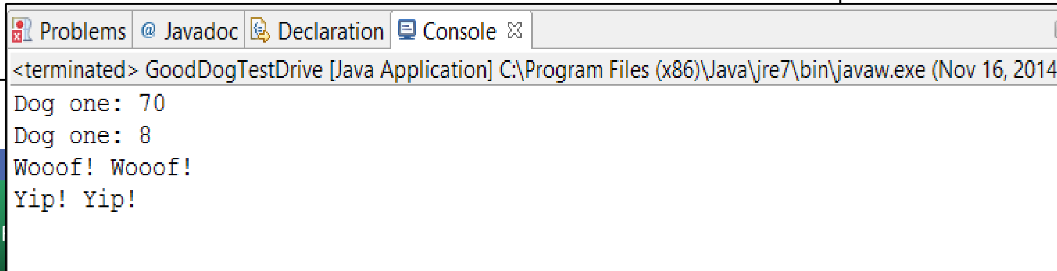


```
class GoodDogTestDrive {
    public static void main (String[] args){
        GoodDog one = new GoodDog();
        one.setSize(70);

        GoodDog two = new GoodDog();
        two.setSize(8);

        System.out.println("Dog one: " + one.getSize());
        System.out.println("Dog one: " + two.getSize());

        one.bark();
        two.bark();

    }
}
```

GoodDog.java



```
class GoodDog {
    private int size;

    public int getSize() {
        return size;
    }

    public void setSize(int s) {
        size = s;
    }

    void bark() {
        if (size > 60){
          System.out.println("Wooof! Wooof!");
        } else if (size > 14) {
           System.out.println("Ruff! Ruff!");
        } else {
             System.out.println("Yip! Yip!");
        }
    }
}
```



```
<terminated> GoodDogTestDrive [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (Nov 16, 2014
Dog one: 70
Dog one: 8
Wooof! Wooof!
Yip! Yip!
```

Creating Instance Variables

Reminder: Instance Variables are the things the object knows

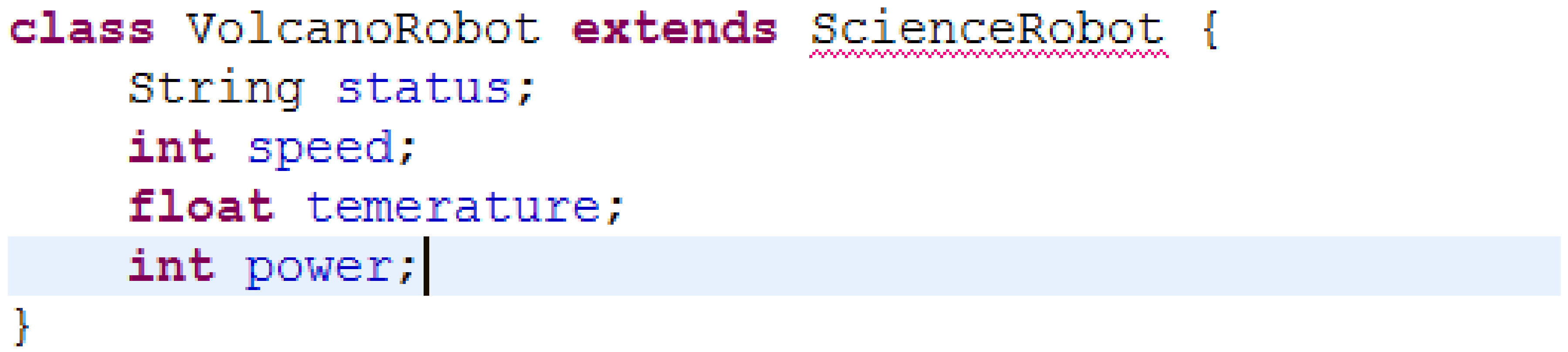
Instance variables always get a default value – if we don't assign a value to the variable or set it, there is still a value:

- Integers = 0
- Floats = 0.0
- Booleans = false
- References = null

Declared inside of a class but not inside of a method

- Local variables are declared inside of a method and must be initialized

For Example...



```
class VolcanoRobot extends ScienceRobot {
    String status;
    int speed;
    float temerature;
    int power;
}
```

In this snippet, VolcanoRobot inherits from ScienceRobot and there are 4 variables that are instance variables since they were not declared inside of a method

Class Variables

As noted previous, these variables apply to the whole class

Used for sharing information between different objects of the same class or for keeping track of common information

We use the static keyword to declare a class variable

```
static int SUM;
static final int MAXOBJECT = 10;
```

The names are capitalized to identify them as a class variable

Exercise 1 – RangeLister.java

Define a method, makeRanger() in a class called RangerList.java

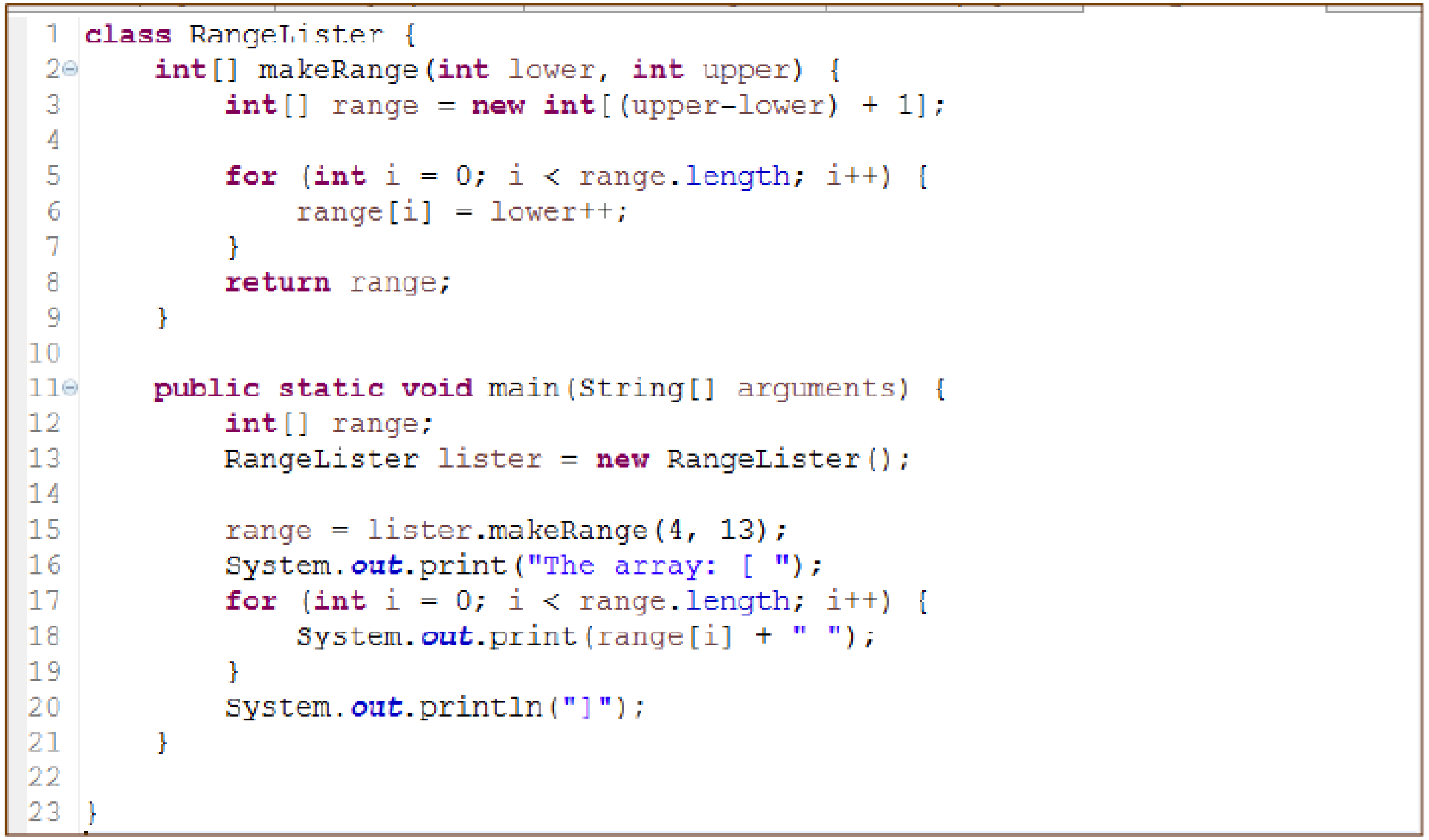
There are 2 to arguments – lower bounds and upper bounds. The method create an array of integers that are between the two bounds

Create this method within its class as shown. Test the application

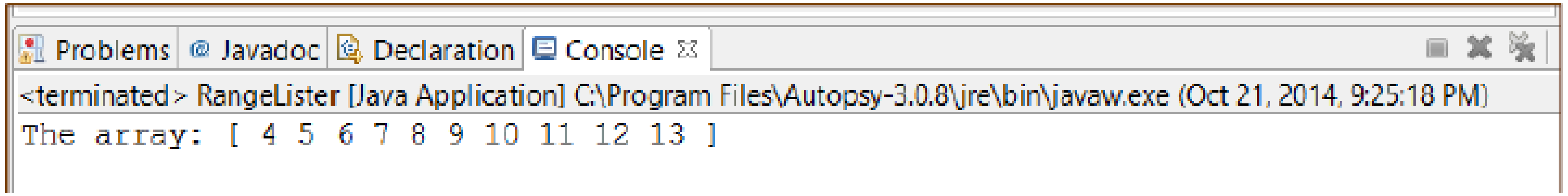
RangerLister.java

Line 15: call makeRange() with values 4, 13

Line 2: method makes an empty array using the arguments to define the elements and then uses a for loop (line 5) to fill the array with the values between 4 and 13



```
 1 class RangeLister {
 2     int[] makeRange(int lower, int upper) {
 3         int[] range = new int[(upper-lower) + 1];
 4 
 5         for (int i = 0; i < range.length; i++) {
 6             range[i] = lower++;
 7         }
 8         return range;
 9     }
10 
11     public static void main(String[] arguments) {
12         int[] range;
13         RangeLister lister = new RangeLister();
14 
15         range = lister.makeRange(4, 13);
16         System.out.print("The array: [ ");
17         for (int i = 0; i < range.length; i++) {
18             System.out.print(range[i] + " ");
19         }
20         System.out.println("]");
21     }
22 
23 }
```



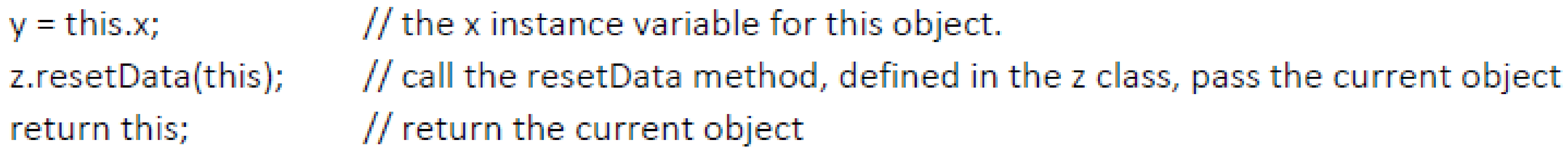
```
<terminated> RangeLister [Java Application] C:\Program Files\Autopsy-3.0.8\jre\bin\javaw.exe (Oct 21, 2014, 9:25:18 PM)
The array: [ 4 5 6 7 8 9 10 11 12 13 ]
```

this keyword

Used to when an object needs to refer to itself in order to use the instance variables it contains or to pass the current object as an argument to another method

Use *this* where we would use the object's name

Can be used anywhere in the same context of any object



```
y = this.x;             // the x instance variable for this object.
z.resetData(this);      // call the resetData method, defined in the z class, pass the current object
return this;            // return the current object
```

Independent Exercise – 2 hours – Page 63

A. Calculations

Using Luna, create a program called ***Investor.java*** that calculates how much a $1400 investment would be worth if it increased in value by 40% during the first year, lost $1500 in value the second year and increased by 12% in the third year

Test your code by running as a Java Application

B. Operators

Using Luna, write a program called ***Divider.java*** that displays two numbers and uses the / and % operators to display the result and remainder after they are divided. Use the \t character escape sequence to separate the result and remainder in your output

Test your code by running as a Java Application

Solutions can be found in Appendix A – Independent Exercise Day 2

Independent Exercise – 2 Hours

A. Birthday.java

Create a program called Birthday.java that turns a birthday in MM/DD/CCYY format into three individual strings Run as a Java application to test the program

B. Sample.java and Sampler.java

Create a class called Sample.java with instance variables for height, weight, and depth where each is an integer.

Create a Java application called Sampler.java that uses your new class (look back to VolcanoRobot for guidance), sets each of these values in an object and displays the values. Run Sampler.java to test your class

Solutions can be found in Appendix A – Independent Exercises Day 3

Independent Exercise – 2 Hours

A. Using countDays() method from the DayCounter.java application, create an application that displays every date in a given year in a single list from January 1 to December 31.

Name the Java file as YearDisplayer.java

Run as a Java Application

B. Create a class that takes words from the first 10 numbers (“one” – “ten”) and converts them into a single long integer. Use the switch Statement for the conversion and command-line arguments for the words.

Name the file WordNumber.java

Run as command line with your word selection (shown next)

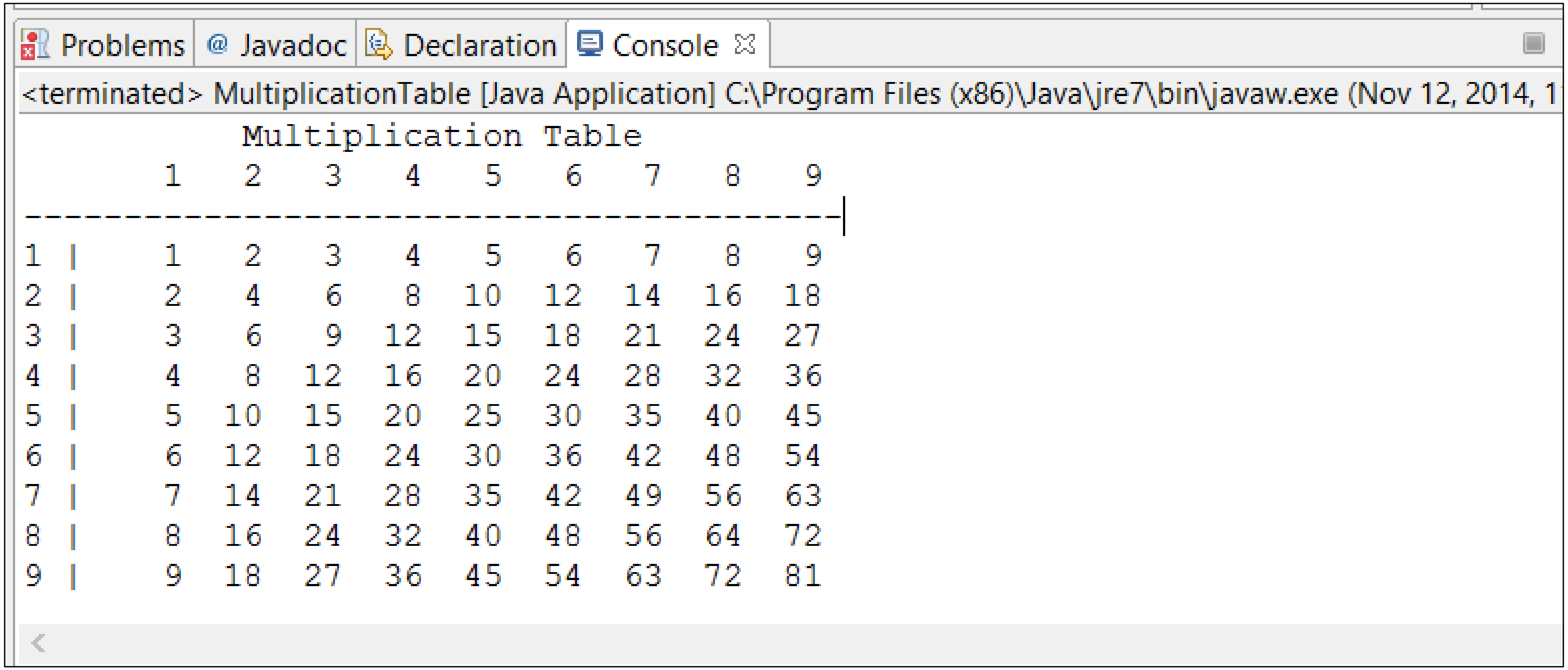
Bonus: Create a Multiplication Table

Create a MultiplicationTable.java application that is 9x9 and display it to the console.

Use a for loop to print the body of the table

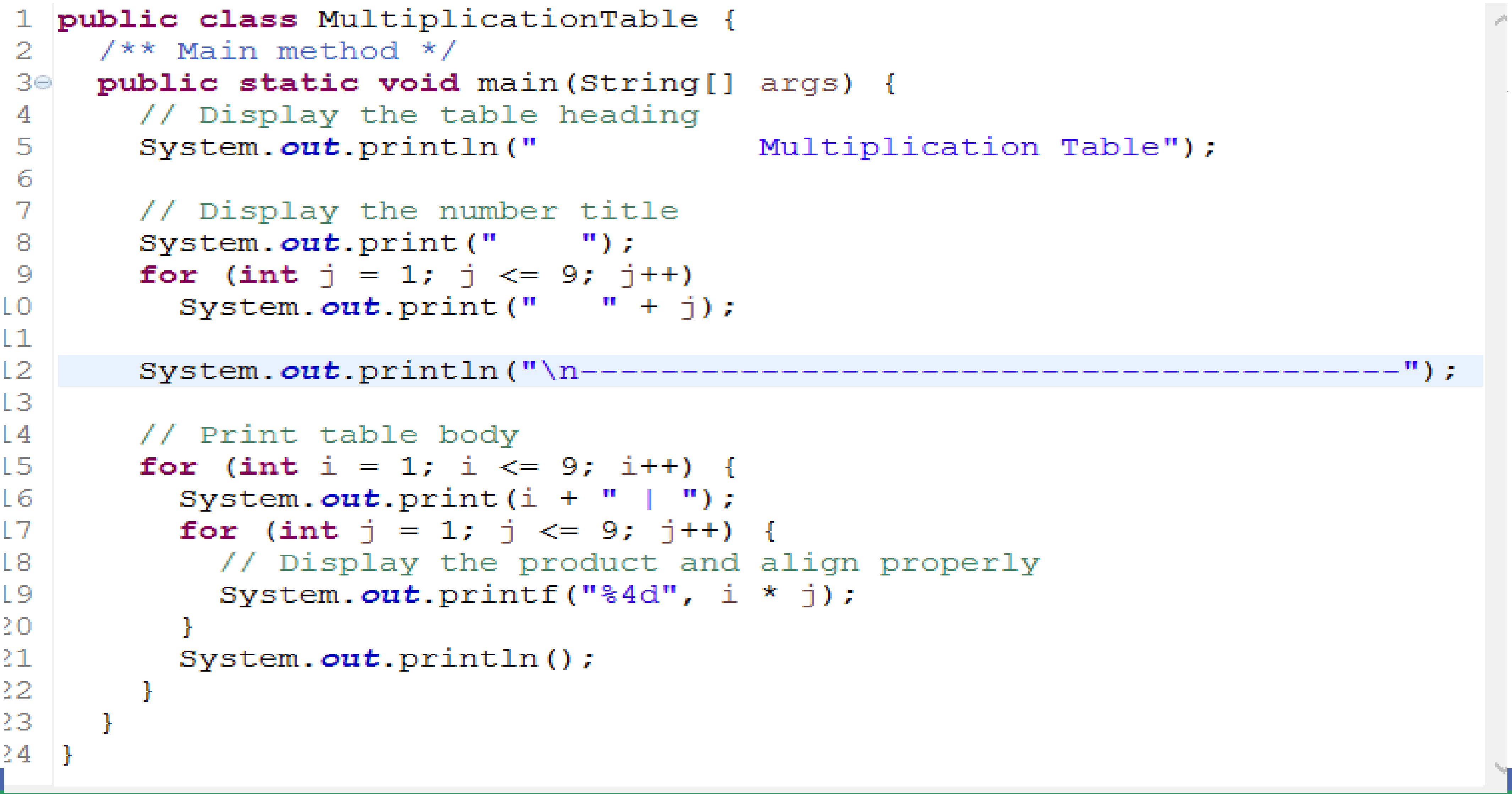
Run as a Java application.

MultiplicationTable.java - Output



```
<terminated> MultiplicationTable [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (Nov 12, 2014, 1
           Multiplication Table
       1   2   3   4   5   6   7   8   9
-----------------------------------------
1 |    1   2   3   4   5   6   7   8   9
2 |    2   4   6   8  10  12  14  16  18
3 |    3   6   9  12  15  18  21  24  27
4 |    4   8  12  16  20  24  28  32  36
5 |    5  10  15  20  25  30  35  40  45
6 |    6  12  18  24  30  36  42  48  54
7 |    7  14  21  28  35  42  49  56  63
8 |    8  16  24  32  40  48  56  64  72
9 |    9  18  27  36  45  54  63  72  81
```

Multiplication Table Solution



```java
public class MultiplicationTable {
  /** Main method */
  public static void main(String[] args) {
    // Display the table heading
    System.out.println("           Multiplication Table");

    // Display the number title
    System.out.print("    ");
    for (int j = 1; j <= 9; j++)
      System.out.print("   " + j);

    System.out.println("\n-----------------------------------------");

    // Print table body
    for (int i = 1; i <= 9; i++) {
      System.out.print(i + " | ");
      for (int j = 1; j <= 9; j++) {
        // Display the product and align properly
        System.out.printf("%4d", i * j);
      }
      System.out.println();
    }
  }
}
```