APPinc.
Making Technology Reachable

Java Programming

RITA M. BARRIOS, Ph.D.

PRES. TRAINING & EDUCATION

Day 2 Recap

Today we learned about statements and expressions – the smallest building block of any programming language. You will find these very similar across all programming languages

Statements create methods which lead to objects and classes

We've created variables and learned about assignment

We've covered much of the programming terminology as well as understanding data types and how they work

And concluding with the operators and order of operation

Week One

Objectives – By Day

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

Day 2- Other

- QA books will be available 1 week before we start
- AARP-Example in code
(based on the assumption you need to be 50 years old)

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

Day 3 - Objects

Java was built to take advantage of the OOP concepts

- Objects
  - Creation
  - Testing/Modifying associated classes
  - Instance variables
  - Calling methods
  - Conversion of an object from one class to the next
- Memory management
- Lots of Exercises
- Independent exercises

New Objects

New Java programs consist of a series of classes

- Classes create objects (called an instance of the class)
- Objects (instances) are self-contained elements of the program

String literals creates new instances of class String with the initialized value of that string

String is a class but can be assigned via the usage of a literal treating the literal as if it were a data type – not true with other classes

To use any other class, we must use the **new** operator

New Operators (2)

The **new** Operator

- Used to create a new object from a class (template)
- Syntax is as shown
- When there are no arguments, we are creating a very basic and simple object of the class
- Argument – value in the parenthesis – the value is passed to the class to create the initial instance variable that will be associated with the object
- The number of arguments is defined by the class itself by way of a constructor which tells us how many arguments the class can take.
  - Wrong number of argument will cause an error
- Let’s see this in action

```
String name = new String(“Dr. Barrios”);
URL address = new URL(http://appinc-us.com);
VolcanoRobot Robbie = new VolcanoRobot();
```

Exercise 1: TokenTester.java – Pg. 65

In this exercise, string tokens are used to analyze stock prices in a dataset – a group of related data. We create objects by using the keyword new in 2 different ways, then display each token the objects contain.

StringTokenizer is in the **java.util** package which divides a string into segments called tokens. To do the division, we must have a delimiter to tell tokenizer where to divide the string

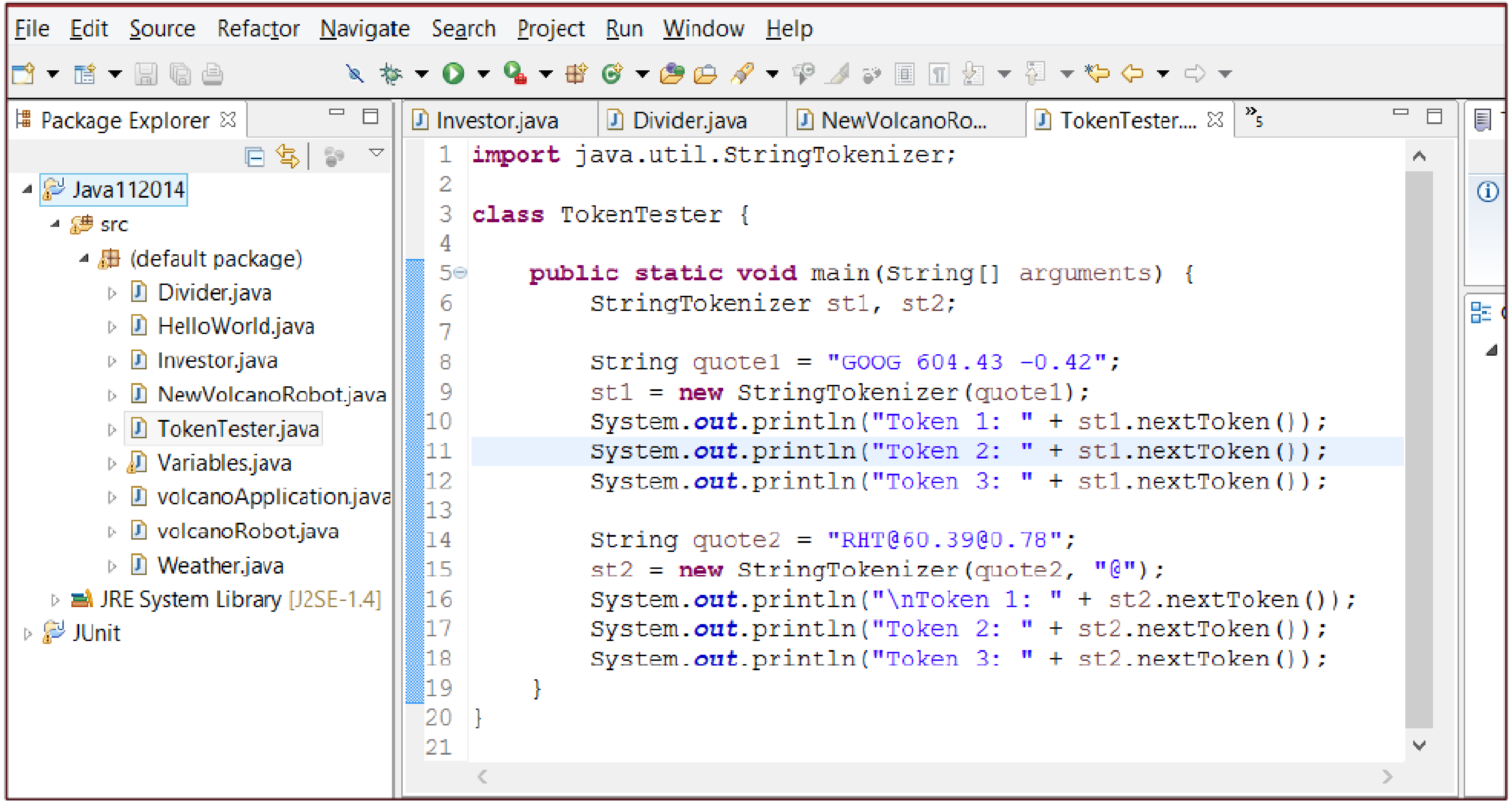
Constructors are:

- StringTokenizer(String str); StringTokenizer(String str, String delimiter)

We will have a class called **TokenTester.java --** code TokenTester.java as shown on the next slide.

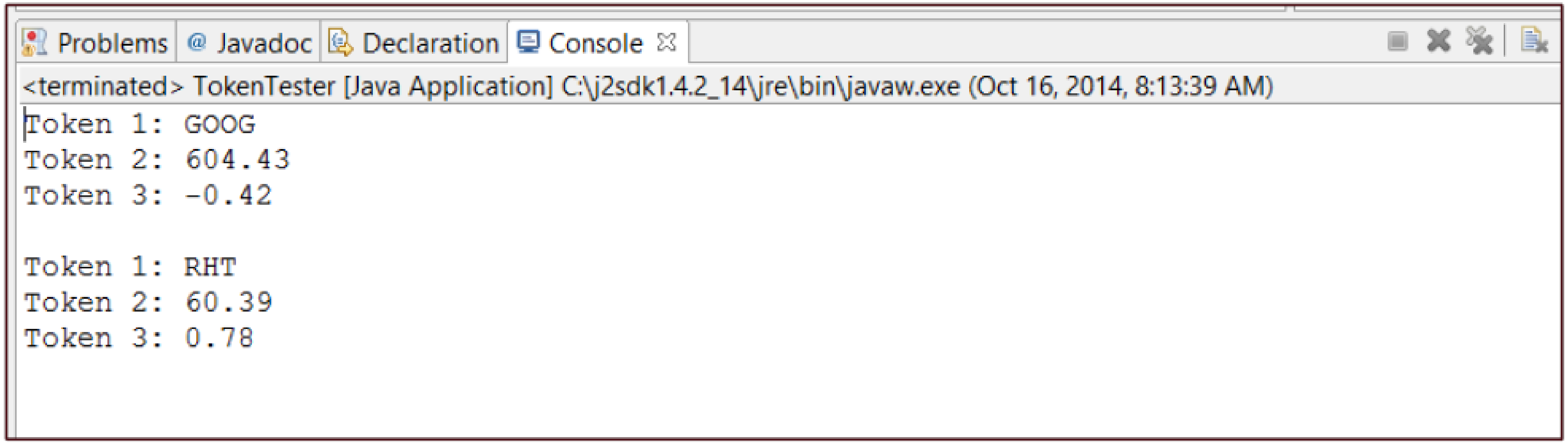
Test your application!

Exercise 1: TokenTester.java



```java
import java.util.StringTokenizer;

class TokenTester {

    public static void main(String[] arguments) {
        StringTokenizer st1, st2;

        String quote1 = "GOOG 604.43 -0.42";
        st1 = new StringTokenizer(quote1);
        System.out.println("Token 1: " + st1.nextToken());
        System.out.println("Token 2: " + st1.nextToken());
        System.out.println("Token 3: " + st1.nextToken());

        String quote2 = "RHT@60.39@0.78";
        st2 = new StringTokenizer(quote2, "@");
        System.out.println("\nToken 1: " + st2.nextToken());
        System.out.println("Token 2: " + st2.nextToken());
        System.out.println("Token 3: " + st2.nextToken());
    }
}
```

TokenTester.java – Output – Pg. 65



```
<terminated> TokenTester [Java Application] C:\j2sdk1.4.2_14\jre\bin\javaw.exe (Oct 16, 2014, 8:13:39 AM)
Token 1: GOOG
Token 2: 604.43
Token 3: -0.42

Token 1: RHT
Token 2: 60.39
Token 3: 0.78
```

Examine TokenTester.java

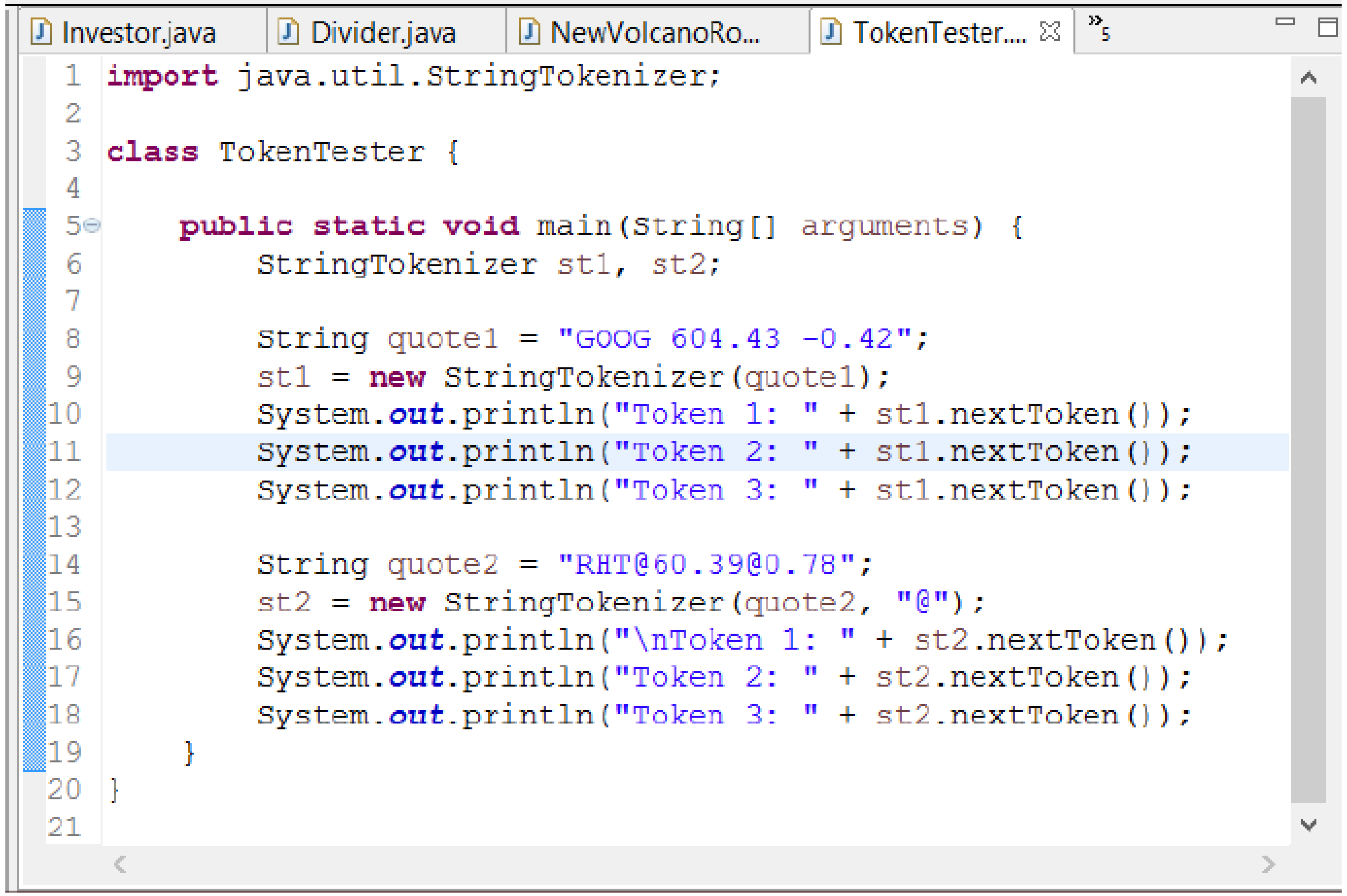
Line 1: import the StringTokenizer class from the java.util package

Line 6: Define 2 StringTokenizer object variables, st1 and st2

Line 8 and 14: Define and assign values to quote1 and quote2

Line 9 and 15: new StringTokenizer(argument) where argument is quote1 and quote2 respectively.

Note line 15: the second argument of @ is telling the tokenizer that @ should be used to delimit the string



```
 1 import java.util.StringTokenizer;
 2 
 3 class TokenTester {
 4 
 5     public static void main(String[] arguments) {
 6         StringTokenizer st1, st2;
 7 
 8         String quote1 = "GOOG 604.43 -0.42";
 9         st1 = new StringTokenizer(quote1);
10         System.out.println("Token 1: " + st1.nextToken());
11         System.out.println("Token 2: " + st1.nextToken());
12         System.out.println("Token 3: " + st1.nextToken());
13 
14         String quote2 = "RHT@60.39@0.78";
15         st2 = new StringTokenizer(quote2, "@");
16         System.out.println("\nToken 1: " + st2.nextToken());
17         System.out.println("Token 2: " + st2.nextToken());
18         System.out.println("Token 3: " + st2.nextToken());
19     }
20 }
21 
```

Constructors

When we use the **new** operator

- A new instance of the class is created
- Memory is allocated to it
- A special method in the class is called – the Constructor

Constructor

- Initialized the object and its variables
- Creates any other objects that the object itself needs
- Performs additional operations the object requires to prepare itself to be used
- A class may have several different constructors – each with different arguments
- The correct constructor is called depending on how many arguments we supply
- All constructors of a class have different arguments from each other
- If a class defines no constructors, an empty constructor is called with new

Memory Management

Dynamic and Automatic

Memory automatically **allocated** in the appropriate size when we create an object

Deallocations happen automatically when the object is killed (destroyed)

JVM will periodically look for unused objects and reclaim the memory – **Garbage Collection**

- Reduces **Memory Leaks** – failure of a program to release memory
- We have to be sure we are not holding onto an object beyond its useful life to avoid running out of memory

Gets and Sets

To **get** to the value of an instance variable, we use **dot notation**

- A form of addressing a name of a class or instance where there are two parts

object/className . variableName

If we have an object called customer that has a variable called orderTotal, using dot notation to refer to it would be

Data Type → float total = customer.orderTotal;

Local variable

Instance variable

Gets and Sets (2)

Setting the values of dot noted instance variables occurs the same as the local variable using the equal (=) operator

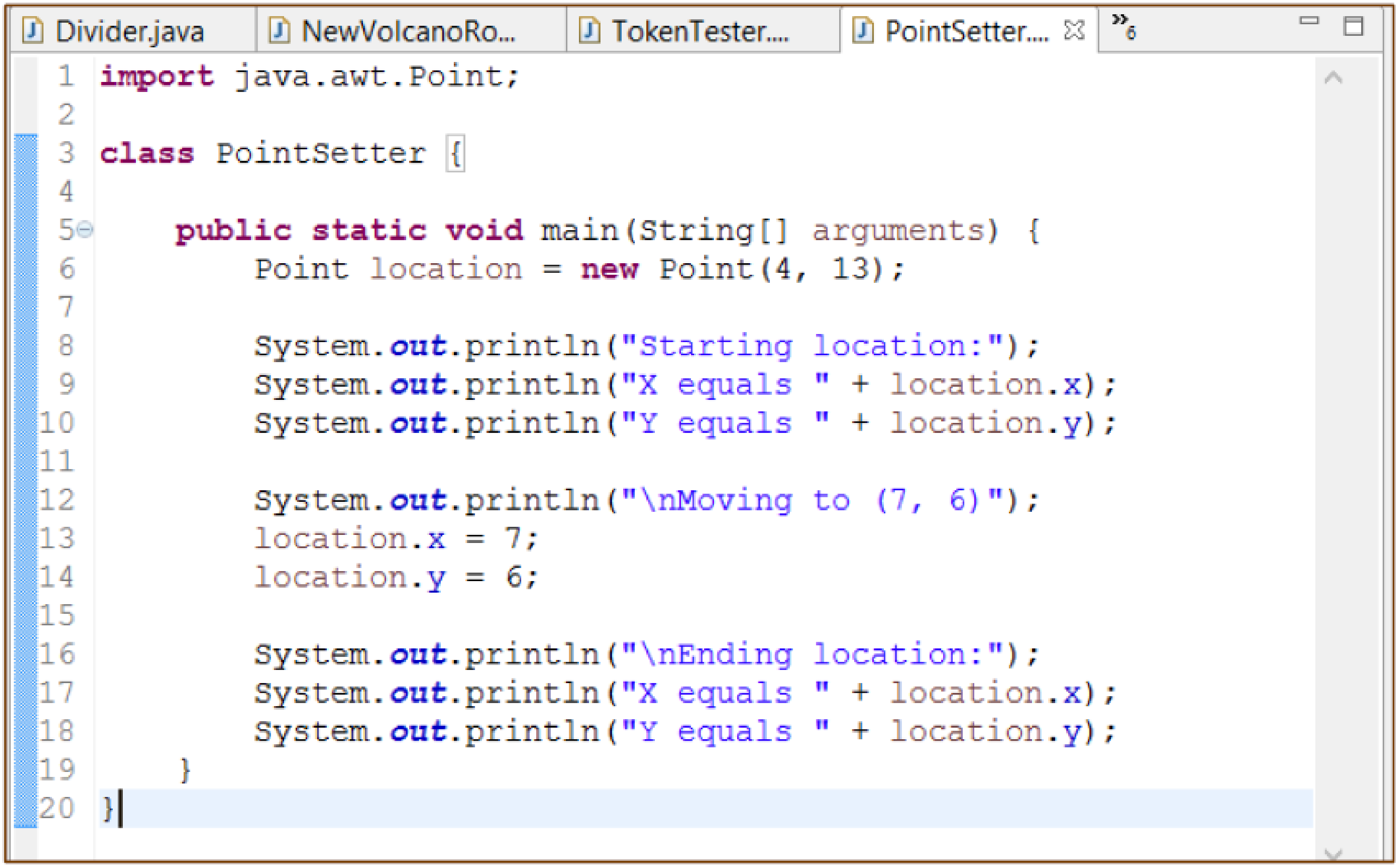
```
customer.layaway = true;
```

Exercise 2: PointSetter.java – Pg. 68

In this exercise we test and modify the instance variables in a **Point** object. Point is a class in the **java.awt** package that represents the x y points in a coordinate system

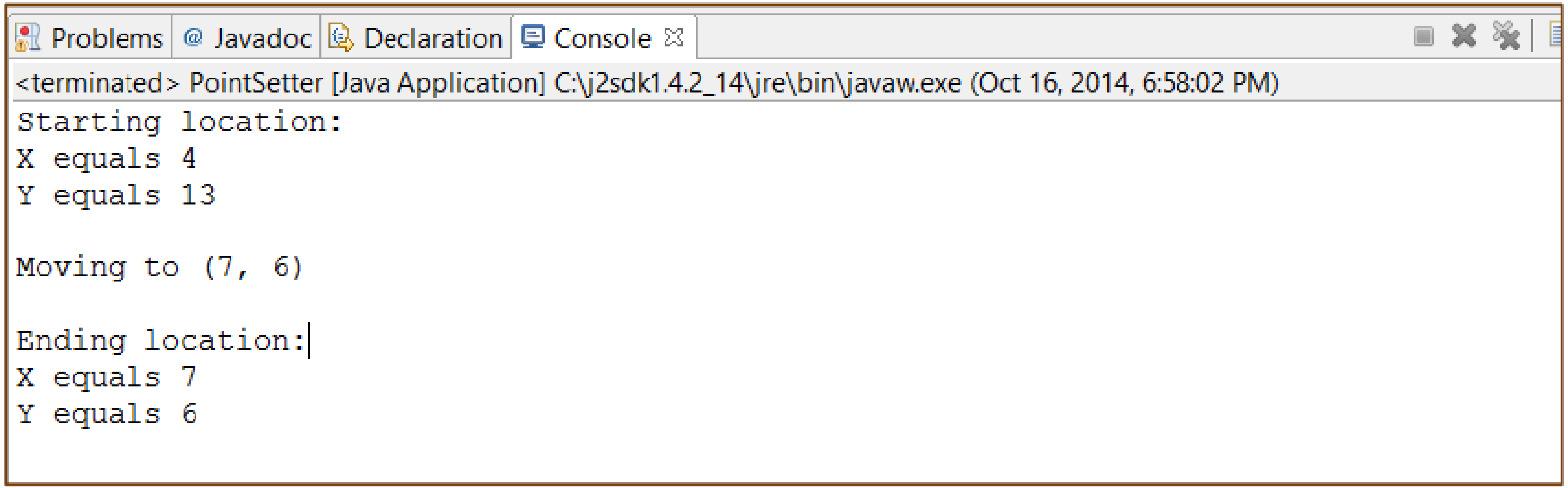
Using Luna code the module as shown. Once you get a clean compile, (no errors), run as a Java application ensuring you test all possible outcomes

Exercise 2: PointSetter.java – Pg. 68



```java
import java.awt.Point;

class PointSetter {

    public static void main(String[] arguments) {
        Point location = new Point(4, 13);

        System.out.println("Starting location:");
        System.out.println("X equals " + location.x);
        System.out.println("Y equals " + location.y);

        System.out.println("\nMoving to (7, 6)");
        location.x = 7;
        location.y = 6;

        System.out.println("\nEnding location:");
        System.out.println("X equals " + location.x);
        System.out.println("Y equals " + location.y);
    }
}
```

PointSetter.java – Output – Pg. 68



```
<terminated> PointSetter [Java Application] C:\j2sdk1.4.2_14\jre\bin\javaw.exe (Oct 16, 2014, 6:58:02 PM)
Starting location:
X equals 4
Y equals 13

Moving to (7, 6)

Ending location:
X equals 7
Y equals 6
```

Examine PointSetter.java

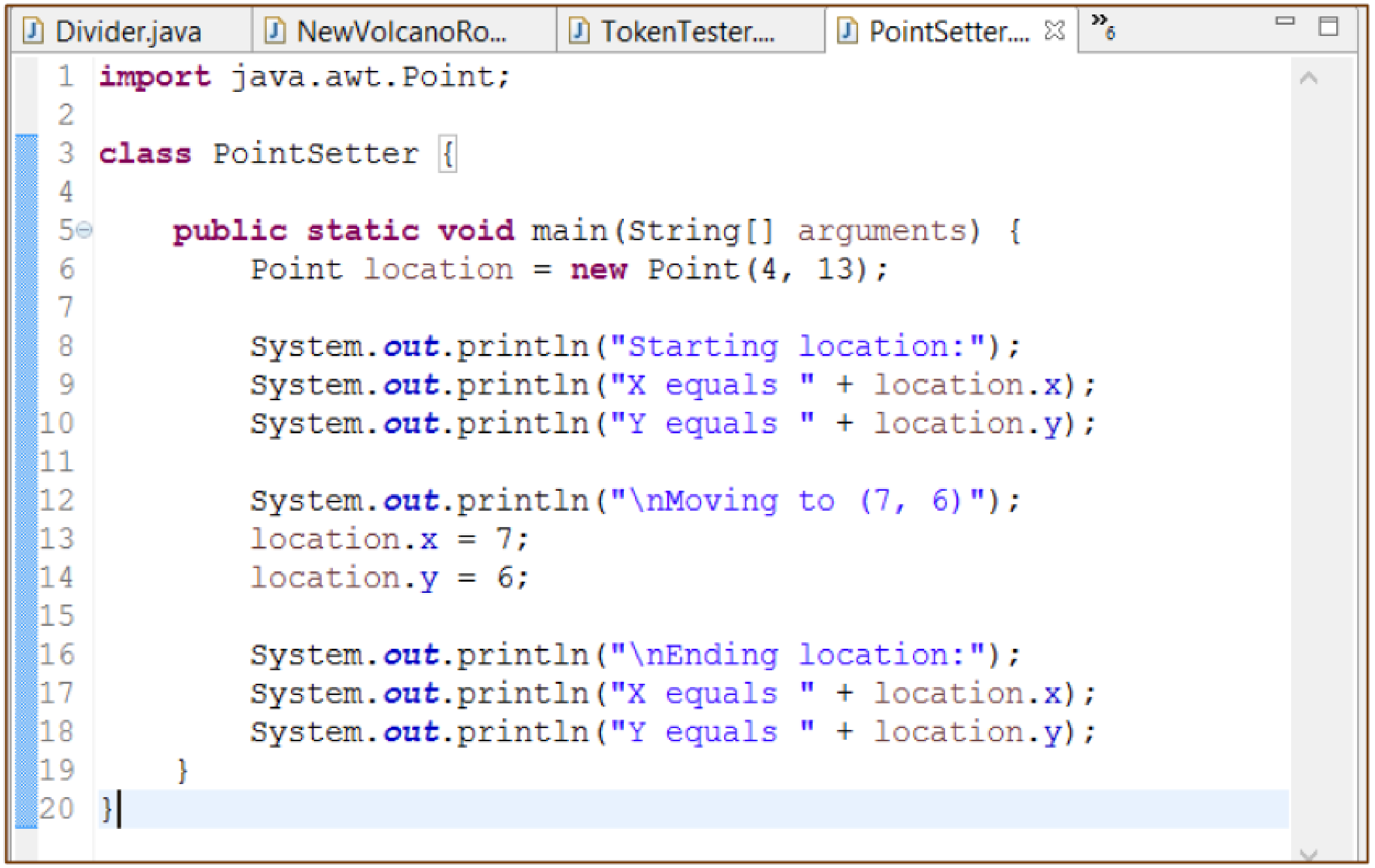
This is a very straight forward program

Line 6: Create an instance of Point where x = 4 and y = 13

Lines 8 – 10: print the values of x and y

Lines 13 – 14: x and y are modified

Lines 16 – 17: print the new values



```
 1 import java.awt.Point;
 2
 3 class PointSetter {
 4
 5     public static void main(String[] arguments) {
 6         Point location = new Point(4, 13);
 7
 8         System.out.println("Starting location:");
 9         System.out.println("X equals " + location.x);
10         System.out.println("Y equals " + location.y);
11
12         System.out.println("\nMoving to (7, 6)");
13         location.x = 7;
14         location.y = 6;
15
16         System.out.println("\nEnding location:");
17         System.out.println("X equals " + location.x);
18         System.out.println("Y equals " + location.y);
19     }
20 }
```

Class Variables

CLASS VARIABLE – variables defined and stored in the class itself

- Applies to the class and all of its instances
- Each instance gets a new copy of the variables defined in the class
- Changes to an instance's variables does not impact the other instances created with the class – it is self contained
- Changes in the class variable will impact all instances since the class will only get one set when it is loaded
- Declared with the keyword **static** before the variable name
  - Every object of the class will hold that declared value for the variable

Class Variables (2)

Each instance of FamilyMember will have its own values for name and age

surname is changed therefore **ALL** instances of FamilyMember will have the surname of Smith

We use dot notation to access the Class Variables – it is good form to refer to the class so we know what type of variable we are working with

```
class FamilyMember {
        static String surname = "Smith";
        String name;
        Int age;
}
```

```
FamilyMember  dad = new FamilyMember();
System.out.println("family surname is : " + dad.surname);
System.out.println("family surname is : " + FamilyName.surname);
```

Calling Methods

We use dot notation when calling methods too

```
object.methodName();
customer.addToCart(itemNumber, price, quantity);
```

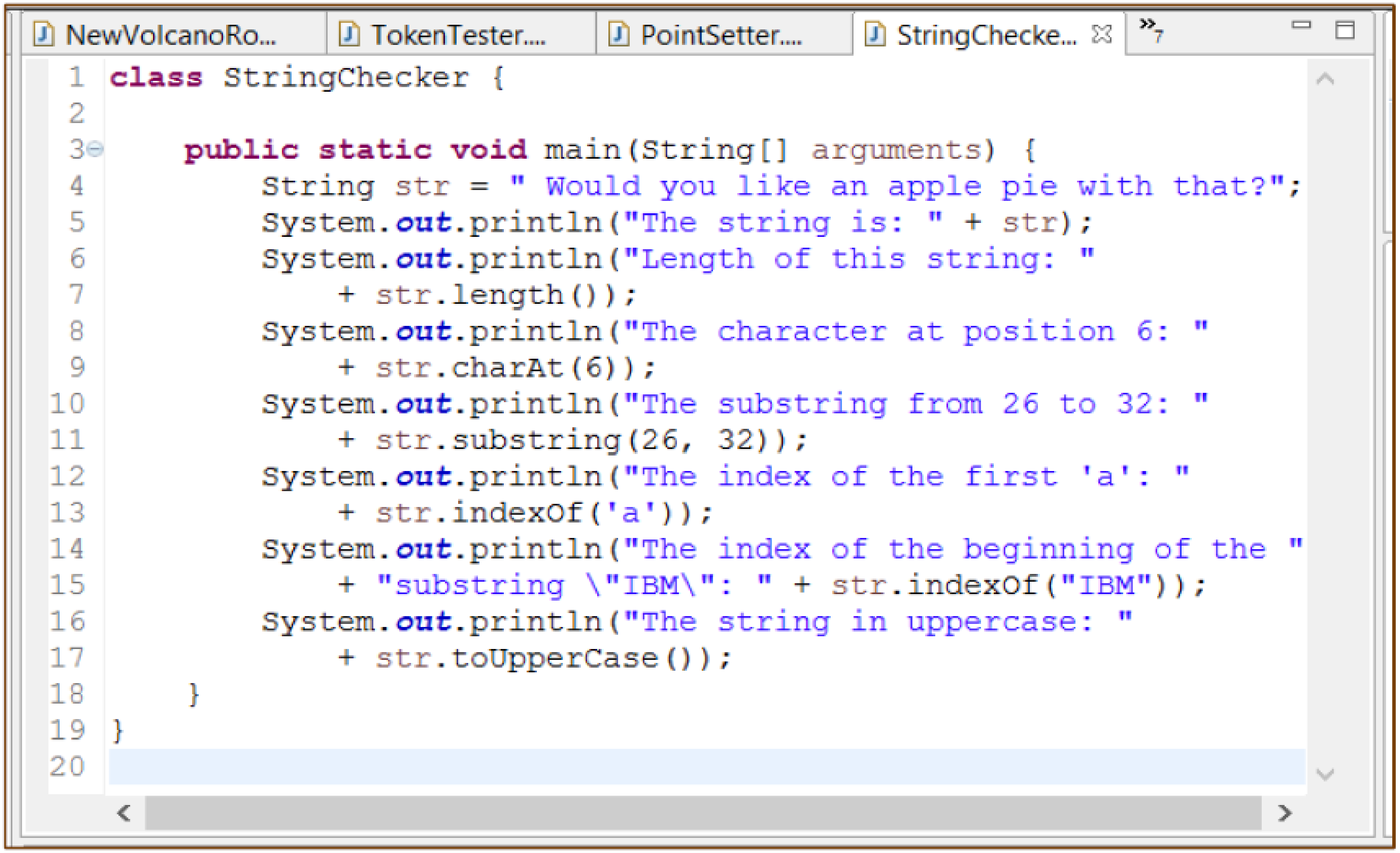
Always use () even if there are no arguments to pass

Exercise 3: StringChecker.java – Pg. 70

In this exercise we call methods that are defined in the String class which include tests and modifications

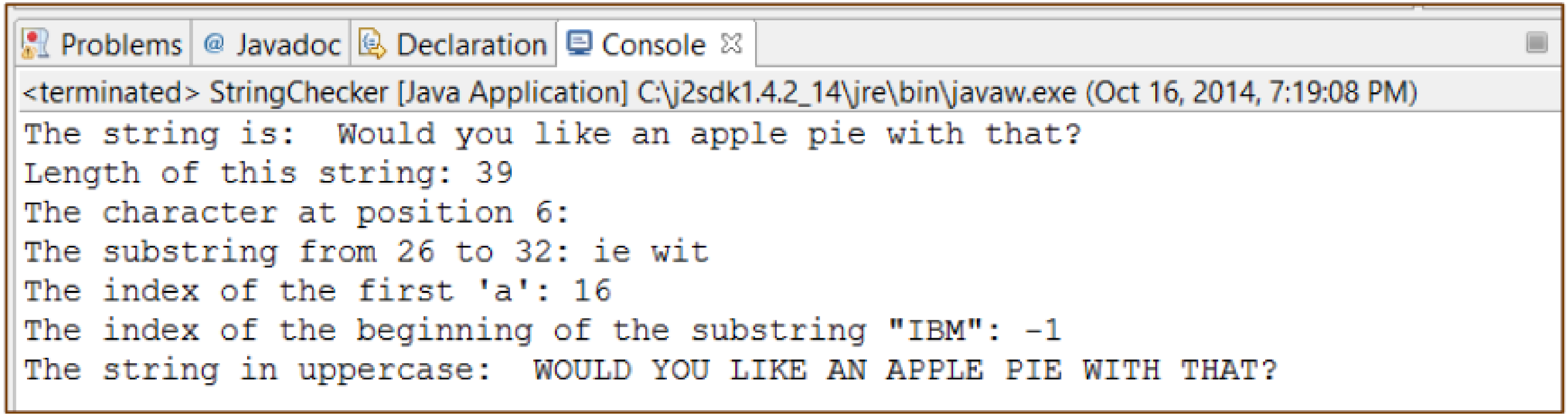
Using Luna, create a Java file called StringChecker.java using the following code. Once we get a clean compile (saved with no errors), run as a Java application

Exercise 3: StringChecker.java – Pg. 70



```java
class StringChecker {

    public static void main(String[] arguments) {
        String str = " Would you like an apple pie with that?";
        System.out.println("The string is: " + str);
        System.out.println("Length of this string: "
            + str.length());
        System.out.println("The character at position 6: "
            + str.charAt(6));
        System.out.println("The substring from 26 to 32: "
            + str.substring(26, 32));
        System.out.println("The index of the first 'a': "
            + str.indexOf('a'));
        System.out.println("The index of the beginning of the "
            + "substring \"IBM\": " + str.indexOf("IBM"));
        System.out.println("The string in uppercase: "
            + str.toUpperCase());
    }
}
```

StringChecker.java Output Pg. 70



```
<terminated> StringChecker [Java Application] C:\j2sdk1.4.2_14\jre\bin\javaw.exe (Oct 16, 2014, 7:19:08 PM)
The string is:  Would you like an apple pie with that?
Length of this string: 39
The character at position 6:
The substring from 26 to 32: ie wit
The index of the first 'a': 16
The index of the beginning of the substring "IBM": -1
The string in uppercase:  WOULD YOU LIKE AN APPLE PIE WITH THAT?
```

Examine StringChecker.java

Line 4: create a new instance of String

Line 5: prints the value of str (line 4)

Line 7: calls length() to return the length of the string

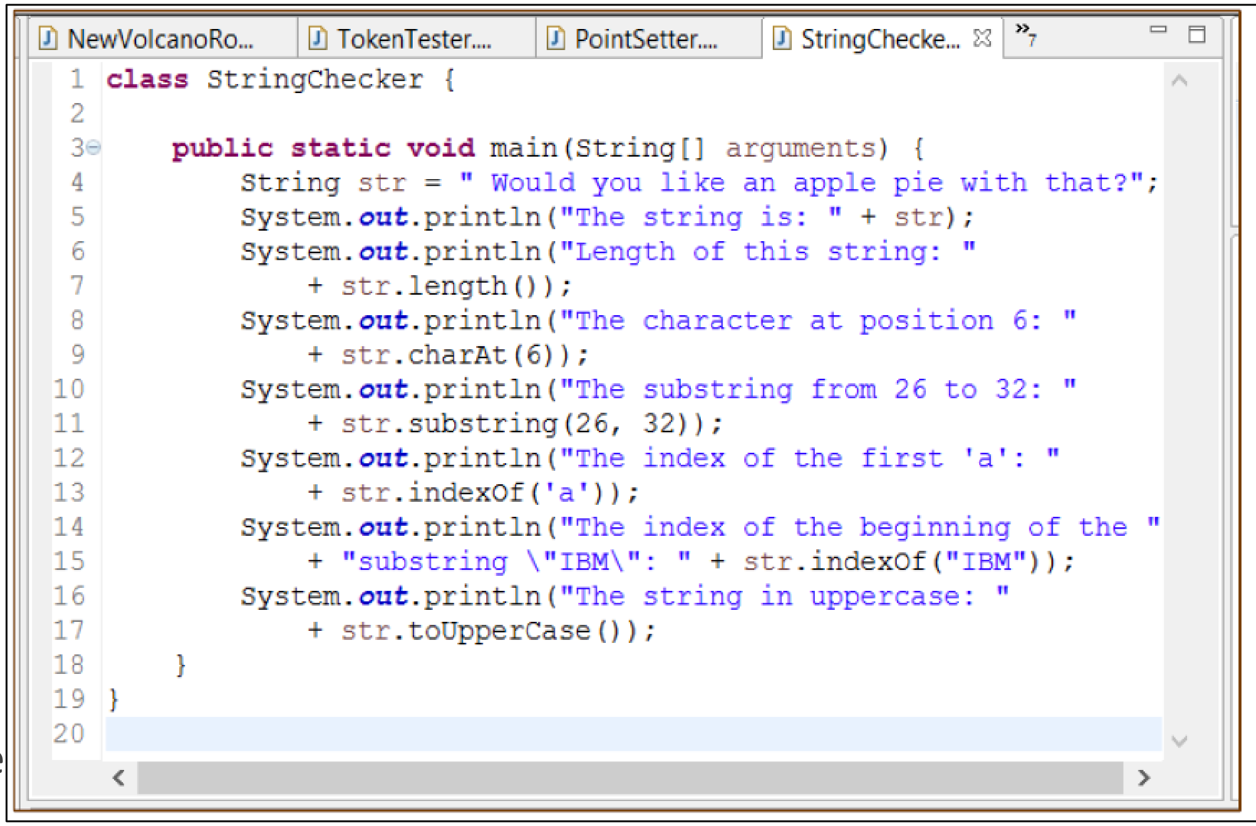
Line 9: calls charAt() to return the character value at the given position

Line 11: calls substr() to take two integers as a range and then return the substring with in the range – substring() can be called with only one argument which is the start of the substring. The entire remainder will be returned

Line 13: calls indexOf() to return the position of the first instance of the given character which is an ‘a’ – note the use of the single quotes

Line 15: a different use of indexOf()

Line 17: toUpperCase() to return a copy of the string in upper case



```java
1  class StringChecker {
2
3      public static void main(String[] arguments) {
4          String str = " Would you like an apple pie with that?";
5          System.out.println("The string is: " + str);
6          System.out.println("Length of this string: "
7              + str.length());
8          System.out.println("The character at position 6: "
9              + str.charAt(6));
10         System.out.println("The substring from 26 to 32: "
11             + str.substring(26, 32));
12         System.out.println("The index of the first 'a': "
13             + str.indexOf('a'));
14         System.out.println("The index of the beginning of the "
15             + "substring \"IBM\": " + str.indexOf("IBM"));
16         System.out.println("The string in uppercase: "
17             + str.toUpperCase());
18     }
19 }
20
```

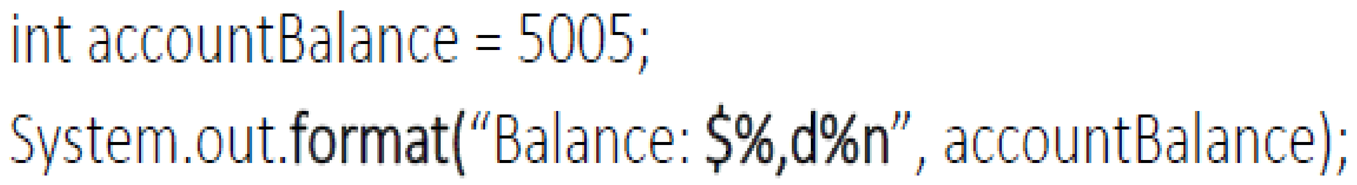
Formatting Strings

System.out.formt() method formats strings with characters such as $ and commas

Has 2 arguments: the output format template and the string

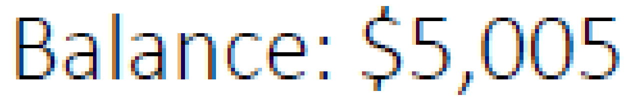
Formatting begins with the % followed by one or more Flags:

- %,d - display a decimal with commas dividing each group into 3 digits
- %n - display a new line



```
int accountBalance = 5005;
System.out.format("Balance: $%,d%n", accountBalance);
```

Output:



```
Balance: $5,005
```

Nesting Method Calls

Methods can return
- A reference to an object
- A primitive
- No value

StringChecker –
- all methods called the String objects str to return the values that are displayed
- charAt() returned a character at the specified point – this can be stored for later use

```
String label = “From: “;
String upper = label.toUpperCase();
```

String object upper contains the value returned by calling label.toUpperCase() which is FROM

Nesting Methods (2)

When a method returns an objects, as shown previously, we can call on the object’s methods which allows us to nest methods as we would variables as in: customer.cancelOrder.fileComplaint();

- fileComplaint() is defined in the object returned by cancelOrder() of the customer object
- System.out. Is a class variable that contains and instance of the PrintStream class (java.io.package) which has a println() method therefore we can call it to print to the console

Class Methods

Apply to the class as a whole

Used for general utility methods that might not operate directly on an object of that class

String contains a method of valueOf() which can take on one of many types of arguments

- Returns a new instance of String containing the argument's string value

Dot notation is used: use the instance of the class or the class itself on the left with the method on the right

Object References

REFERENCE – an address that indicates where an object’s variables and methods are stored

When we assign an object to a variable or pass an object to a method as an argument, we are not using the object directly – we are using a reference to that object

Called passing by reference (vs. passing by value)

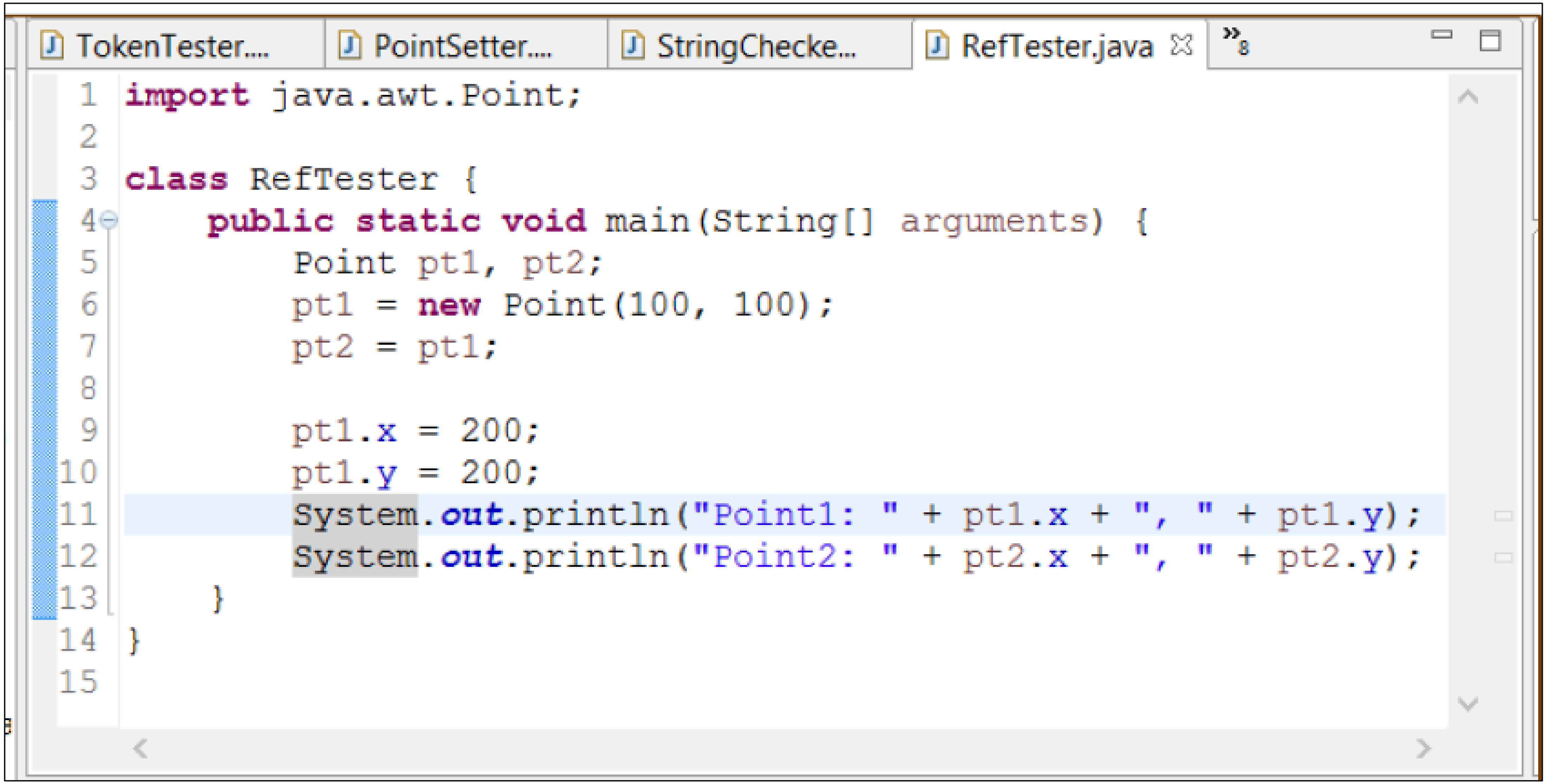
The next exercise shows how this works.

Exercise 4: RefTester.java – Pg. 72

In this exercise we are going to work with references to pass variables/objects to a method.

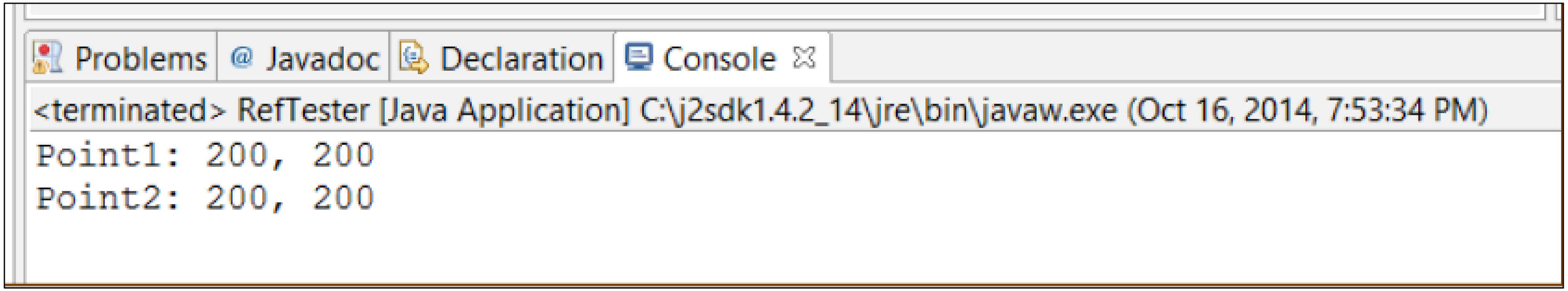
Using Luna, create a Java file called RefTester.java. Once we get a clean compile, (save with no errors), run as a Java application

RefTester.java – Pg. 72



```
import java.awt.Point;

class RefTester {
    public static void main(String[] arguments) {
        Point pt1, pt2;
        pt1 = new Point(100, 100);
        pt2 = pt1;

        pt1.x = 200;
        pt1.y = 200;
        System.out.println("Point1: " + pt1.x + ", " + pt1.y);
        System.out.println("Point2: " + pt2.x + ", " + pt2.y);
    }
}
```

RefTester.java - Output



```
<terminated> RefTester [Java Application] C:\j2sdk1.4.2_14\jre\bin\javaw.exe (Oct 16, 2014, 7:53:34 PM)
Point1: 200, 200
Point2: 200, 200
```

Examine RefTester.java

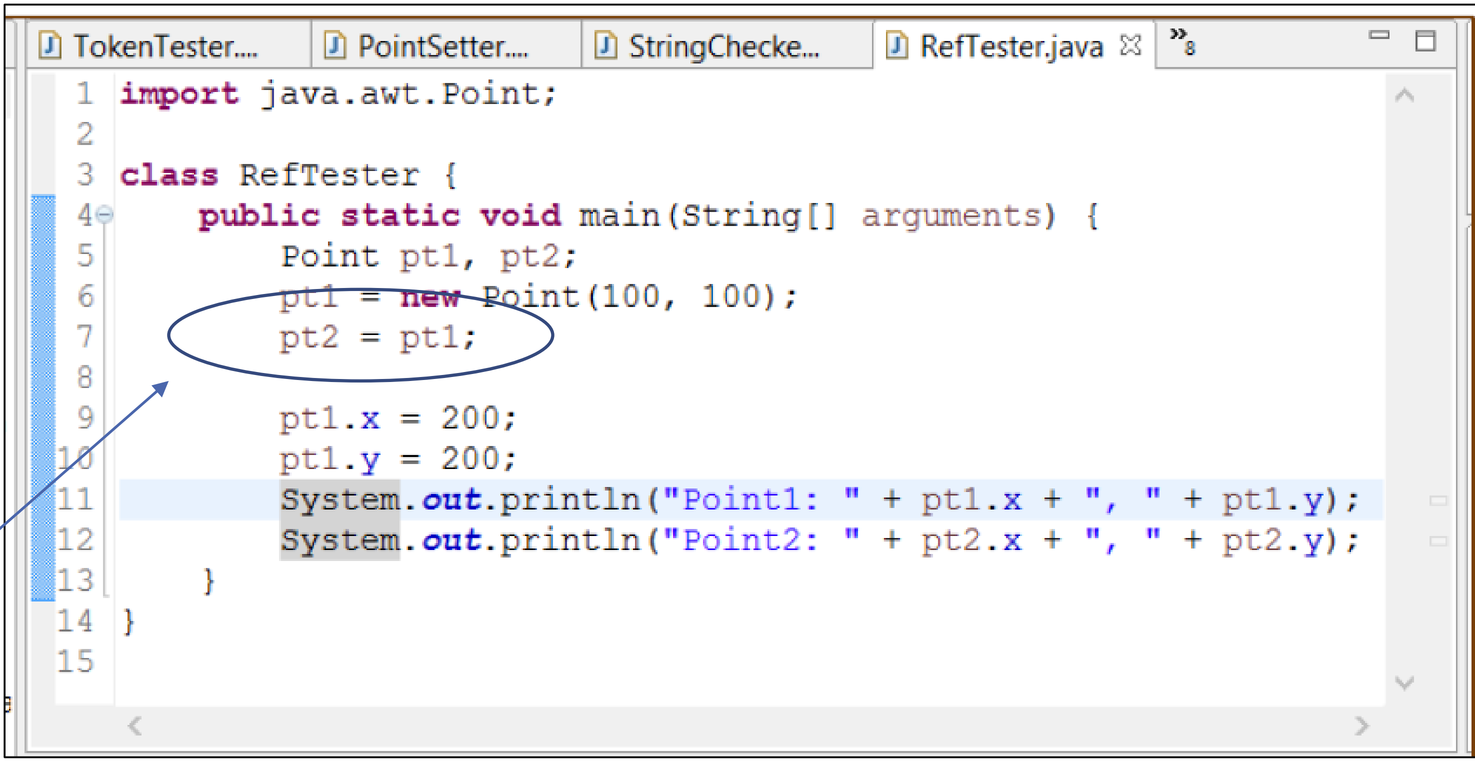
Line 5 – two Point variables are created

Line 6 – a new Point object is assigned to pt1

Line 7 – the value of pt1 is assigned to pt2

Line 9 – 12: These are the important lines

- x and y of pt1 are both set to 200 and then displayed
  - Note that values of pt2 are also changed automatically
    - Line 7 creates a reference from pt2 to pt1 so if we change pt1, pt2 also gets it – the reference!



```
 1 import java.awt.Point;
 2 
 3 class RefTester {
 4     public static void main(String[] arguments) {
 5         Point pt1, pt2;
 6         pt1 = new Point(100, 100);
 7         pt2 = pt1;
 8 
 9         pt1.x = 200;
10         pt1.y = 200;
11         System.out.println("Point1: " + pt1.x + ", " + pt1.y);
12         System.out.println("Point2: " + pt2.x + ", " + pt2.y);
13     }
14 }
15 
```

Object Casting and Primitives

As we’ve seen, methods and constructors require very specific things to take on a form and will not accept anything else – we must use variables of the correct data type

We **CAST** value into the correct format

Most common

- Casting between primitive – int to float, float to double
- Casting from an object of a class to an object of another class – Object to String
- Primitive to object and then extracting primitives values
- Boolean can never be cast
- Destination must be as large as source to avoid truncation
- int can go to character but cannot be used further to perform mathematics

Object Comparisons – Values and Classes

One of the most common functions

The operators we've used previously will not work on objects with the exception of == and !=

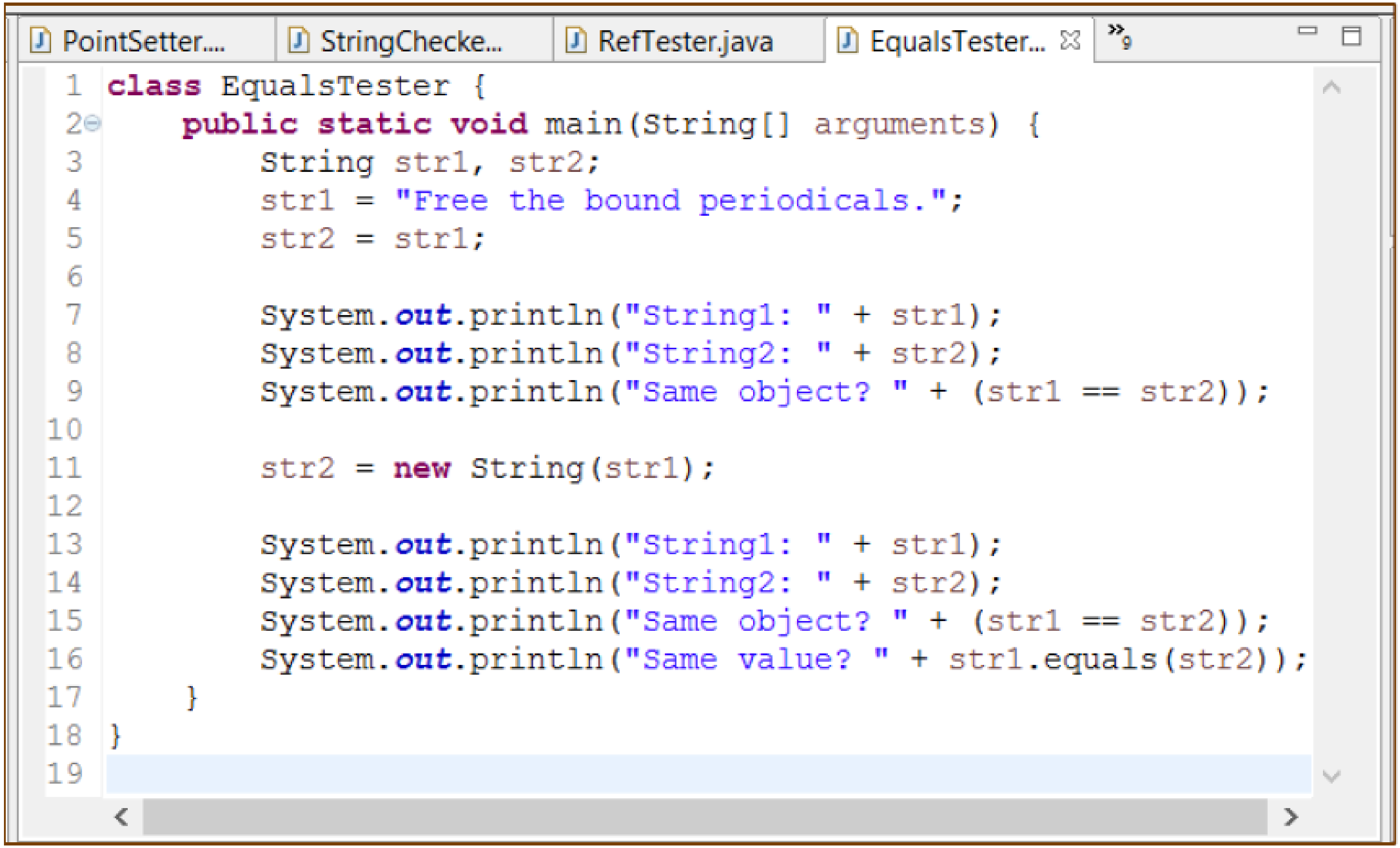
- These do not operate how you expect them to with objects
  - They do not compare value of the objects but whether both sides of the operator refer to the same object
- To compare objects, we implement special methods and call those methods
- We use equals() in our next exercise to determine whether two String objects have matching values

Exercise 5: EqualsTester.java

In this exercise we look at comparing objects by calling the equals() method to determine if two String objects have matching values. The method tests each character in the string and returns true if they have the same value

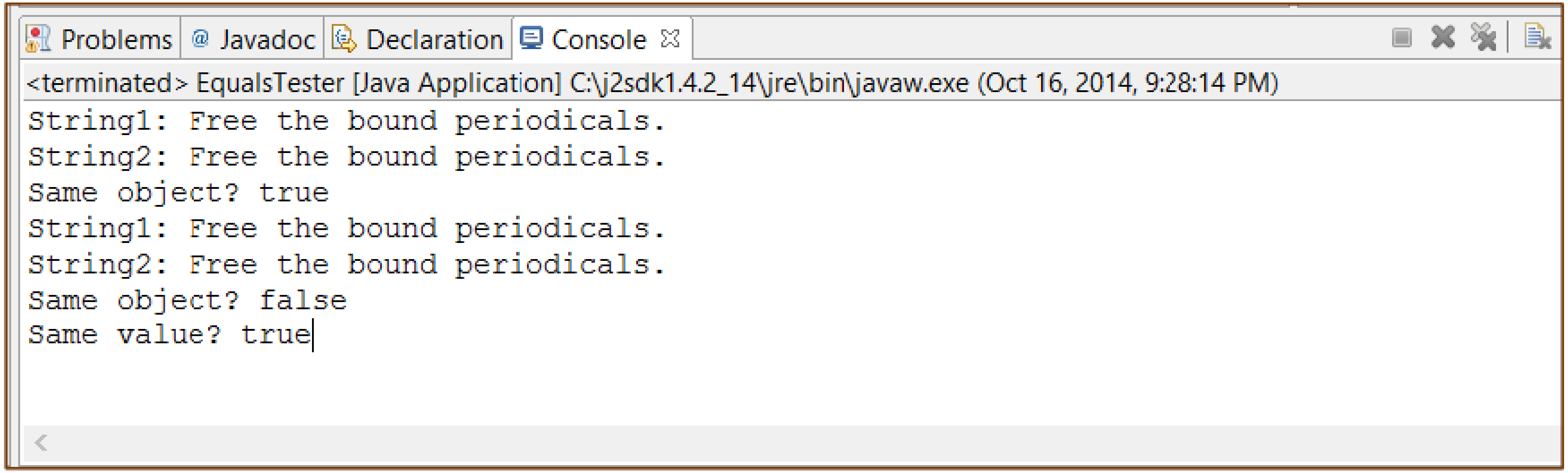
Using Luna, create a Java file called EqualsTester.java with the following code. The code can be found on page 75

Please test each condition of your program as a Java application



```java
class EqualsTester {
    public static void main(String[] arguments) {
        String str1, str2;
        str1 = "Free the bound periodicals.";
        str2 = str1;

        System.out.println("String1: " + str1);
        System.out.println("String2: " + str2);
        System.out.println("Same object? " + (str1 == str2));

        str2 = new String(str1);

        System.out.println("String1: " + str1);
        System.out.println("String2: " + str2);
        System.out.println("Same object? " + (str1 == str2));
        System.out.println("Same value? " + str1.equals(str2));
    }
}
```

EqualsTester.java - Output



```
<terminated> EqualsTester [Java Application] C:\j2sdk1.4.2_14\jre\bin\javaw.exe (Oct 16, 2014, 9:28:14 PM)
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? true
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? false
Same value? true
```

Examine EqualsTester.java

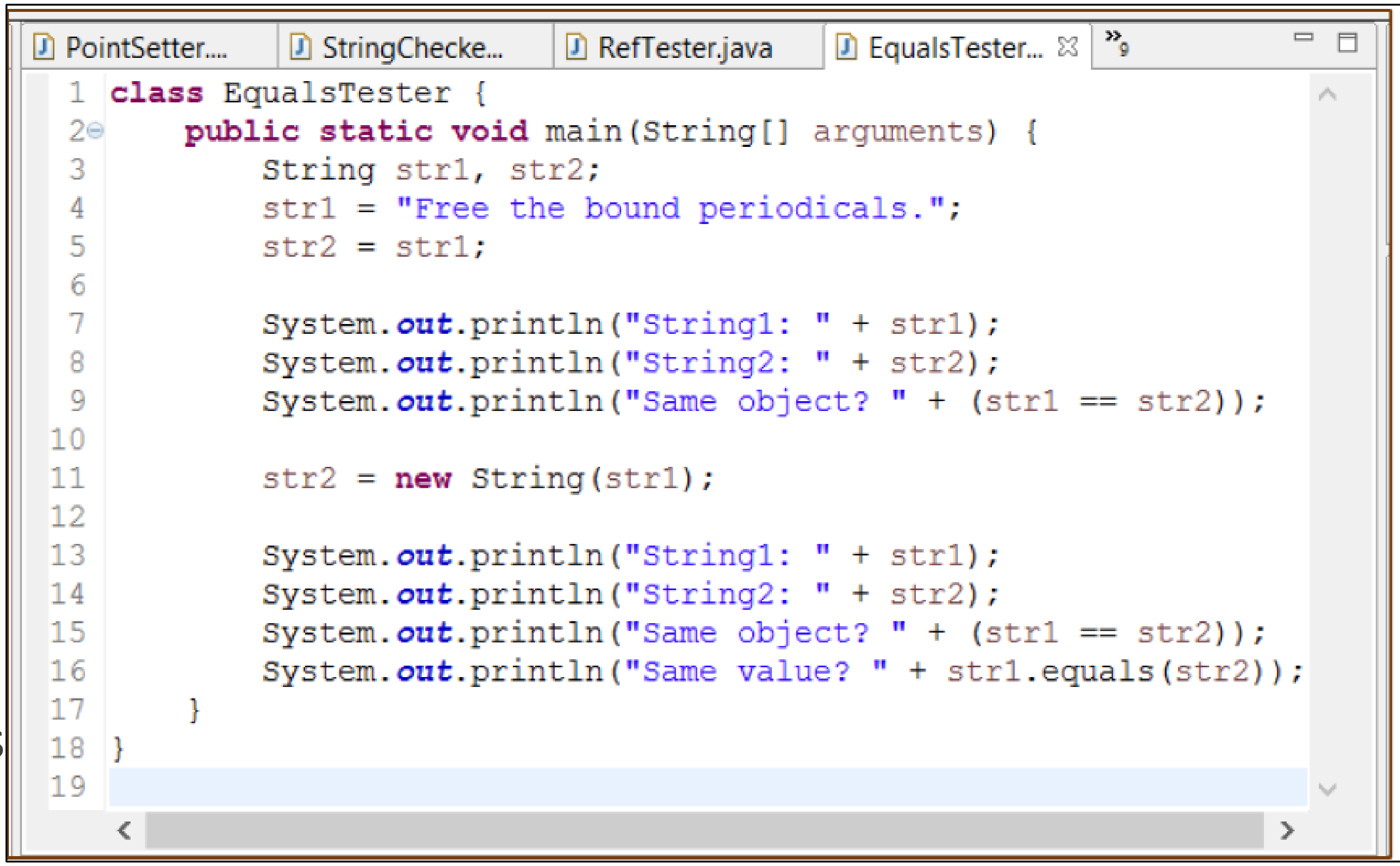
Lines 3 – 5: Declare two variables (str1 and str2), assign the literal to str1 and then assign the same value to str2 to create the reference to the same object

Line 9 – equality test

Line 11 – create a new String object with the same value as str1 and assign str2 to the new String object – creating two different String objects with the same name

Line 15 – equality test with == which returns false since they do not have the same value in memory

Line 16 – using equality() the returned value is true since it is comparing character by character



```
1  class EqualsTester {
2      public static void main(String[] arguments) {
3          String str1, str2;
4          str1 = "Free the bound periodicals.";
5          str2 = str1;
6  
7          System.out.println("String1: " + str1);
8          System.out.println("String2: " + str2);
9          System.out.println("Same object? " + (str1 == str2));
10 
11         str2 = new String(str1);
12 
13         System.out.println("String1: " + str1);
14         System.out.println("String2: " + str2);
15         System.out.println("Same object? " + (str1 == str2));
16         System.out.println("Same value? " + str1.equals(str2));
17     }
18 }
19 
```

Determine the Class of an Object

key.getClass().getName() method to find out the class

getClass() is defined in the Object class – can be called in ALL objects
- Returns a Class object that represents the object's class
- getName() method returns a string containing the name of the class

instanceof – Operator - comparison expression with a reference to an object on the left and a class name on the right
- Returns a boolean
  - True = object is an instance of the named class or subclass of the class
  - False = object is not an instance of

```
boolean check1 = "Texas" insanceof String; ← Returns true
Point pt = new Point(10, 10);
Boolean check2 = pt instanceof String;  ← Returns false
```

Questions?

Open Discussion

Day 3 Recap

Today we learned to work with objects

- Creating
- Reading their values
- Changing their values
- Calling their methods

We talked about casting

- From one class to another
- Primitive data types

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

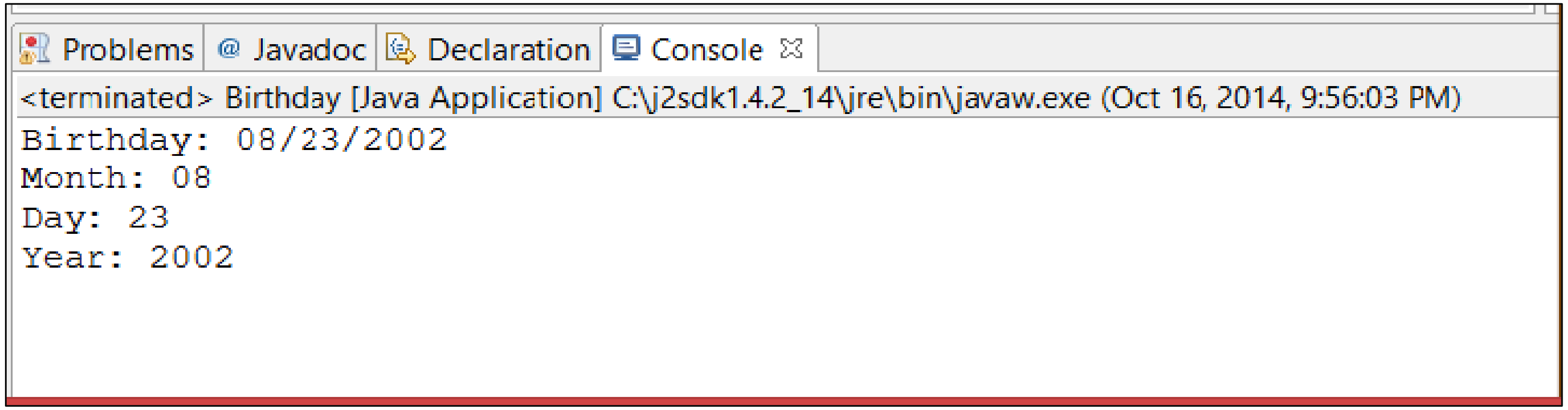
Create a class called Sample.java with instance variables for height, weight, and depth where each is an integer.

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

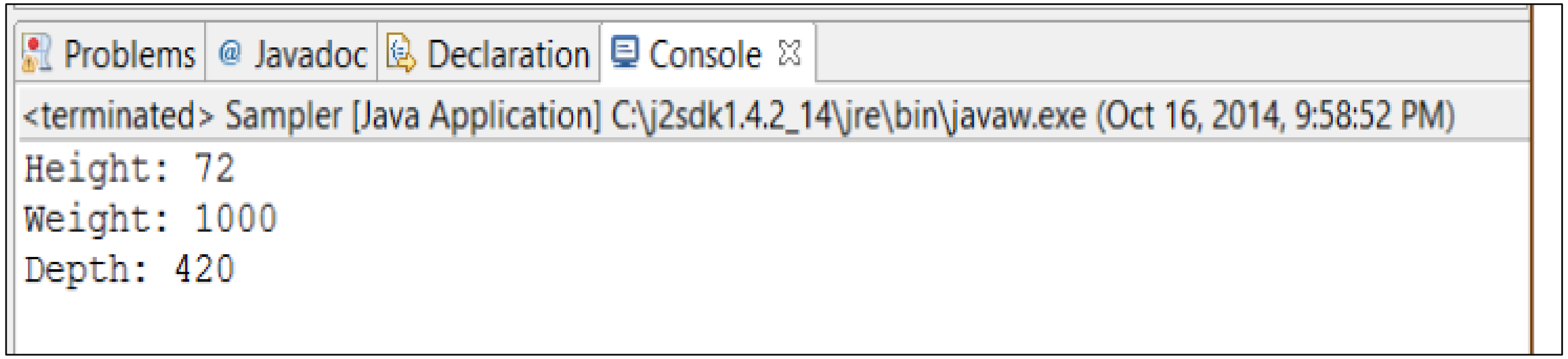
Independent Exercise Output - Pg. 293 -294

Birthday.java

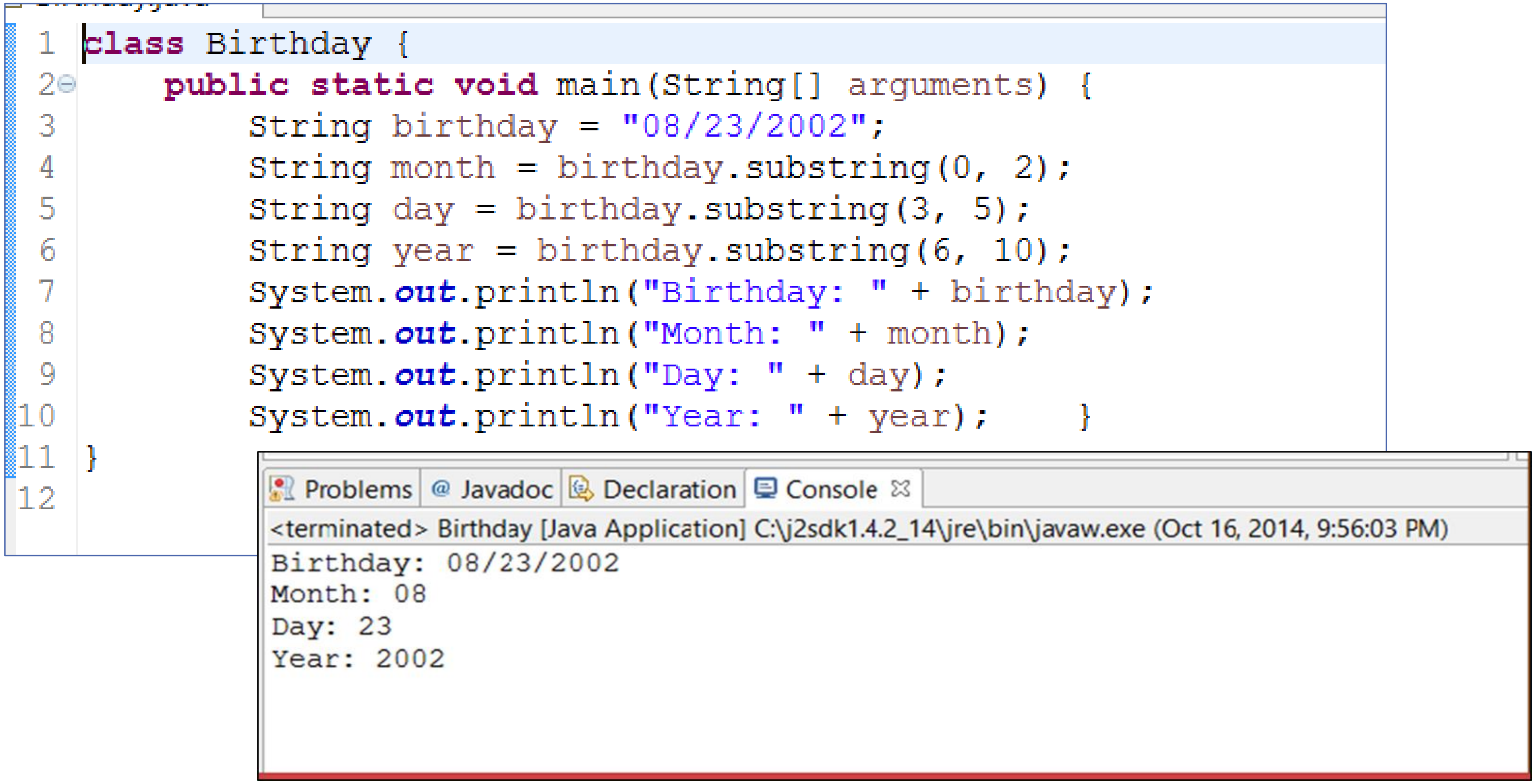


```
<terminated> Birthday [Java Application] C:\j2sdk1.4.2_14\jre\bin\javaw.exe (Oct 16, 2014, 9:56:03 PM)
Birthday: 08/23/2002
Month: 08
Day: 23
Year: 2002
```

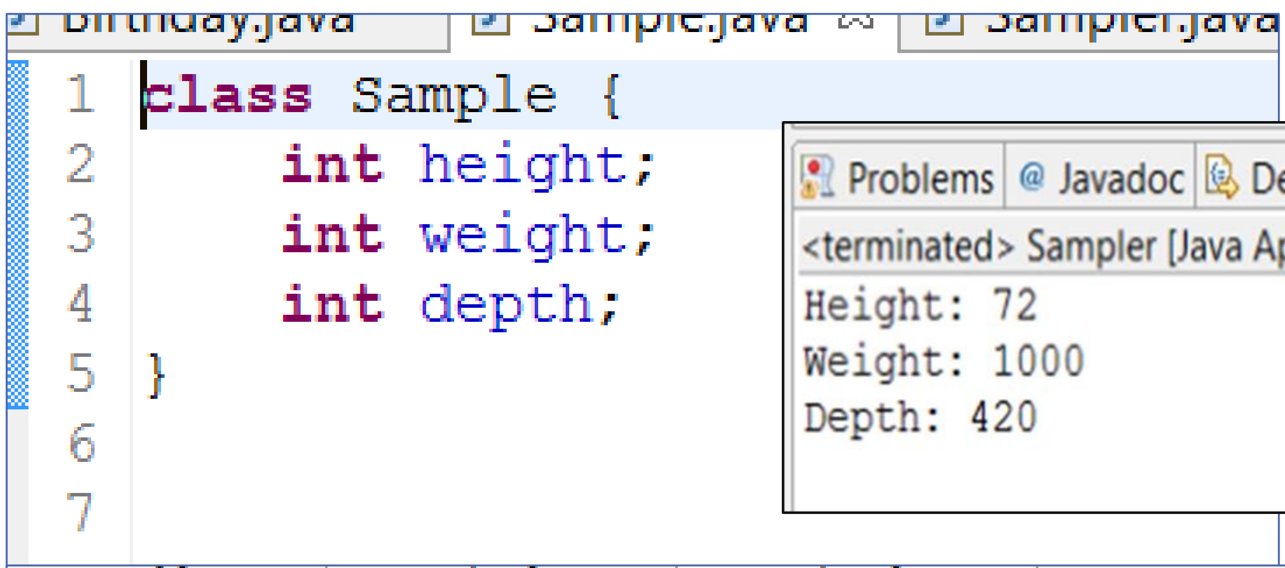
Sampler.java



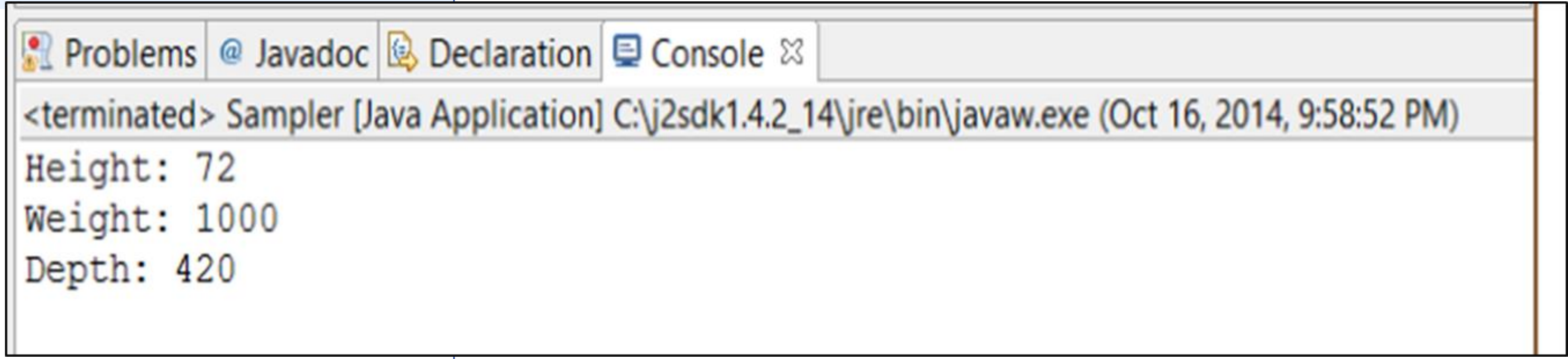
```
<terminated> Sampler [Java Application] C:\j2sdk1.4.2_14\jre\bin\javaw.exe (Oct 16, 2014, 9:58:52 PM)
Height: 72
Weight: 1000
Depth: 420
```


```java
class Birthday {
    public static void main(String[] arguments) {
        String birthday = "08/23/2002";
        String month = birthday.substring(0, 2);
        String day = birthday.substring(3, 5);
        String year = birthday.substring(6, 10);
        System.out.println("Birthday: " + birthday);
        System.out.println("Month: " + month);
        System.out.println("Day: " + day);
        System.out.println("Year: " + year);     }
}
```

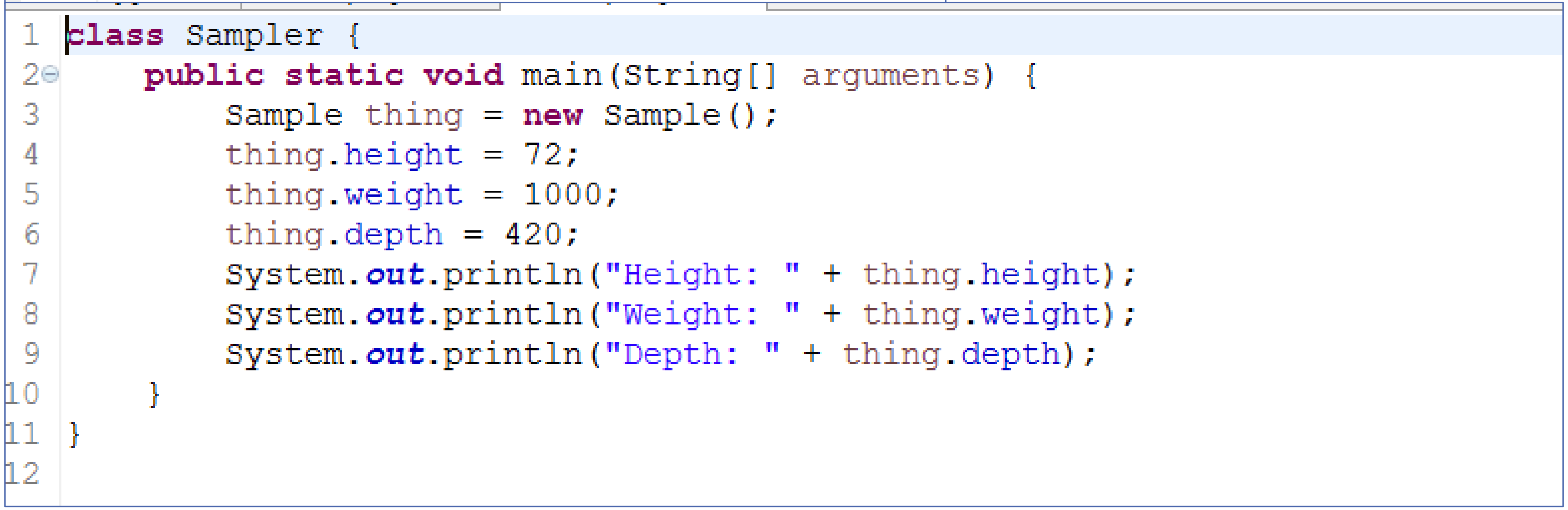
```
<terminated> Birthday [Java Application] C:\j2sdk1.4.2_14\jre\bin\javaw.exe (Oct 16, 2014, 9:56:03 PM)
Birthday: 08/23/2002
Month: 08
Day: 23
Year: 2002
```



```java
class Sample {
    int height;
    int weight;
    int depth;
}
```



```
<terminated> Sampler [Java Application] C:\j2sdk1.4.2_14\jre\bin\javaw.exe (Oct 16, 2014, 9:58:52 PM)
Height: 72
Weight: 1000
Depth: 420
```



```java
class Sampler {
    public static void main(String[] arguments) {
        Sample thing = new Sample();
        thing.height = 72;
        thing.weight = 1000;
        thing.depth = 420;
        System.out.println("Height: " + thing.height);
        System.out.println("Weight: " + thing.weight);
        System.out.println("Depth: " + thing.depth);
    }
}
```