09 - Wrapper Classes and Tokening Strings

ICSI 201
Introduction to Computer Science

Qi Wang
Department of Computer Science
University at Albany
State University of New York

Course Materials

- All course materials are **copyrighted** and can be used by the students who are enrolled in the class only.
- Posting/sharing the course materials for other uses without permission is prohibited.

Outline

- Introduction to Wrapper Classes
- Wrapper Classes
  - Numeric Data Types and the Character Class
- Tokenizing Strings:
  - The *StringTokenizer* Class

Introduction to Wrapper Classes

Introduction to Wrapper Classes

- Java provides 8 primitive data types.
- They are called “primitive” because they are not created from classes.
- Java provides wrapper classes for all of the primitive data types.
- A *wrapper class* is a class that is “wrapped around” a primitive data type.
- The wrapper classes are part of `java.lang` so to use them, there is no `import` statement required.

Wrapper Classes

- Wrapper classes allow you to create objects to represent a primitive.
- Wrapper classes are immutable, which means that once you create an object, you cannot change the object’s value.
- To get the value stored in an object you must call a method.
- Wrapper classes provide static methods that are very useful.

Wrapper Classes for the Numeric Data Types

Numeric Data Type Wrappers

- Java provides wrapper classes for all of the primitive data types.
- The numeric primitive wrapper classes are:

Wrapper Class	Numeric Primitive Type It Applies To
`Byte`	`byte`
`Double`	`double`
`Float`	`float`
`Integer`	`int`
`Long`	`long`
`Short`	`Short`

Creating a Wrapper Object

- To create objects from these wrapper classes, you can pass a value to the constructor:

```
Integer number = new Integer(7);
```

- You can also assign a primitive value to a wrapper class object:

```
Integer number;
number = 7;
```

The Parse Methods

- Any `String` containing a number, such as “127.89”, can be converted to a numeric data type.
- Each of the numeric wrapper classes has a static method that converts a string to a number.
  - The `Integer` class has a method that converts a `String` to an `int`,
  - The `Double` class has a method that converts a `String` to a `double`,
  - etc.
- These methods are known as *parse methods* because their names begin with the word “parse.”

The Parse Methods

```
// Store 1 in bVar.
byte bVar = Byte.parseByte("1");
// Store 2599 in iVar.
int iVar = Integer.parseInt("2599");
// Store 10 in sVar.
short sVar = Short.parseShort("10");
// Store 15908 in lVar.
long lVar = Long.parseLong("15908");
// Store 12.3 in fVar.
float fVar = Float.parseFloat("12.3");
// Store 7945.6 in dVar.
double dVar = Double.parseDouble("7945.6");
```

- The parse methods all throw a `NumberFormatException` if the `String` object does not represent a numeric value.

The `toString` Methods

- Each of the numeric wrapper classes has a static `toString` method that converts a number to a string.
- The method accepts the number as its argument and returns a string representation of that number.

```
int i = 12;
double d = 14.95;
String str1 = Integer.toString(i);
String str2 = Double.toString(d);
```

MIN_VALUE and MAX_VALUE

- The numeric wrapper classes each have a set of static final variables:
  - MIN_VALUE
  - MAX_VALUE.
- These variables hold the minimum and maximum values for a particular data type.

```
System.out.println("The minimum value for an "
                + "int is "
                + Integer.MIN_VALUE);
System.out.println("The maximum value for an "
                + "int is "
                + Integer.MAX_VALUE);
```

Autoboxing and Unboxing

- You can declare a wrapper class variable and assign a value:

```
Integer number;
number = 7;
```

- You may think this is an error, but because number is a wrapper class variable, *autoboxing* occurs.

- *Unboxing* does the opposite with wrapper class variables:

```
Integer myInt = 5;          // Autoboxes the value 5
int primitiveNumber;
primitiveNumber = myInt;  // unboxing
```

Autoboxing and Unboxing

- You rarely need to declare numeric wrapper class objects, but they can be useful when you need to work with primitives in a context where primitives are not permitted.
- Recall the `ArrayList` class, which works only with objects.

```
ArrayList<int> list =
    new ArrayList<int>();      // Error!
ArrayList<Integer> list =
    new ArrayList<Integer>(); // OK!
```

- Autoboxing and unboxing allow you to conveniently use `ArrayList`s with primitives.

Wrapper Classes for primitive *char* Type

The Character Class

Character Testing and Conversion With The `Character` Class

- The `Character` class allows a char data type to be *wrapped* in an object.

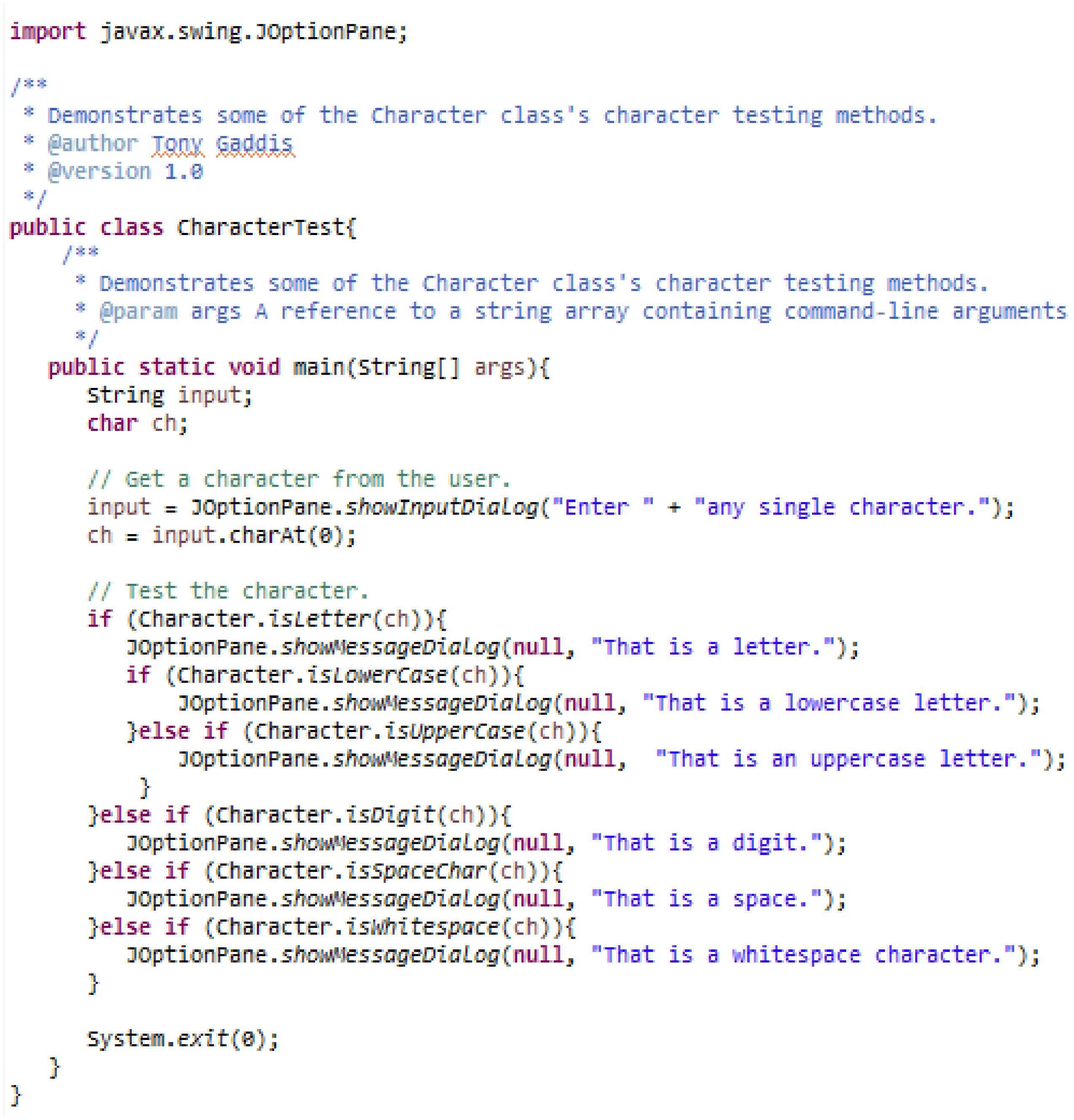
Wrapper Class	Primitive Type It Applies To
`Character`	`char`

- The `Character` class provides methods that allow easy testing, processing, and conversion of character data.

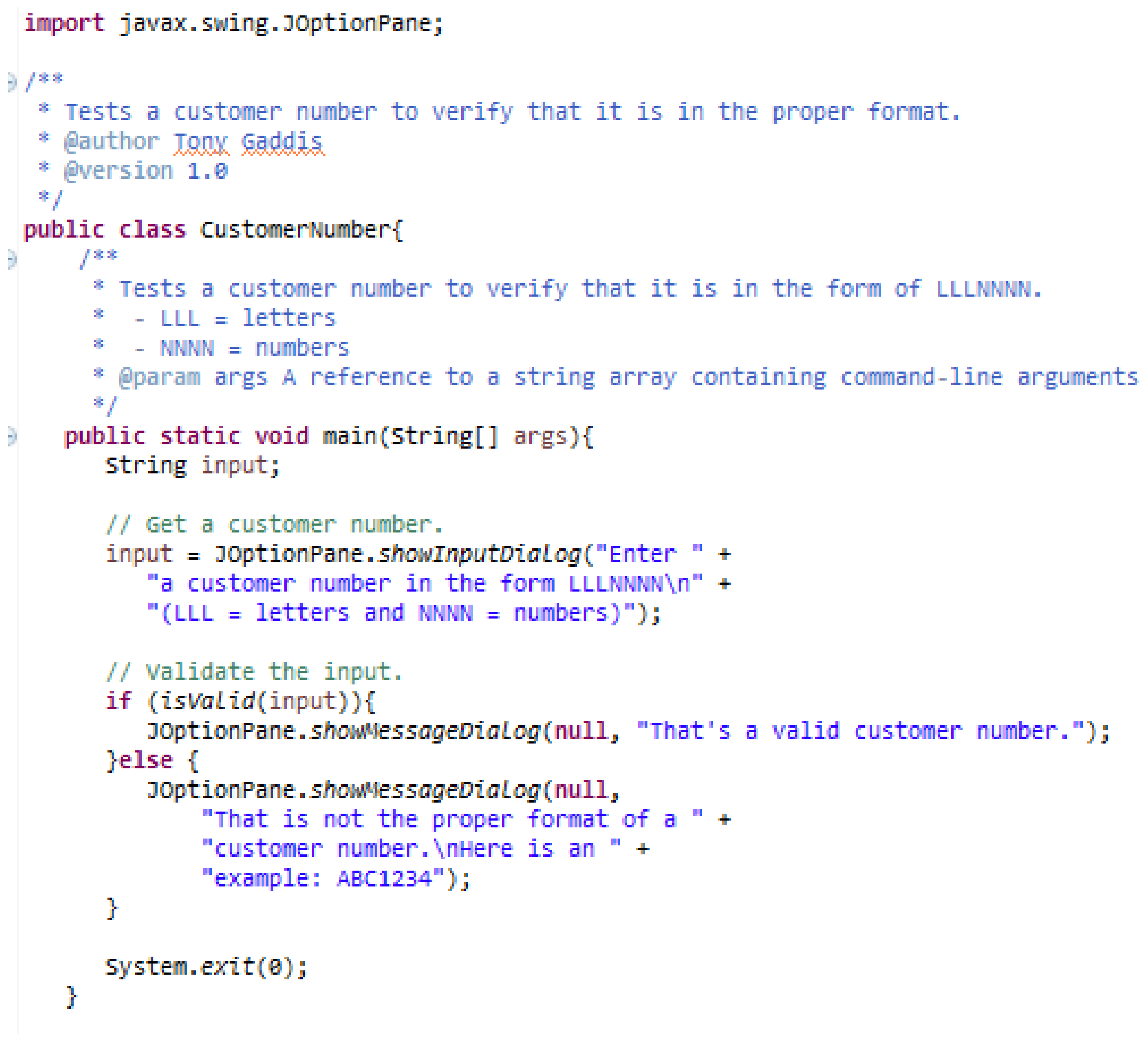
The `Character` Class

Method	Description
`boolean isDigit( char ch)`	Returns true if the argument passed into *ch* is a digit from 0 through 9. Otherwise returns false.
`boolean isLetter( char ch)`	Returns true if the argument passed into *ch* is an alphabetic letter. Otherwise returns false.
`boolean isLetterOrDigit( char ch)`	Returns true if the character passed into *ch* contains a digit (0 through 9) or an alphabetic letter. Otherwise returns false.
`boolean isLowerCase( char ch)`	Returns true if the argument passed into *ch* is a lowercase letter. Otherwise returns false.
`boolean isUpperCase( char ch)`	Returns true if the argument passed into *ch* is an uppercase letter. Otherwise returns false.
`boolean isSpaceChar( char ch)`	Returns true if the argument passed into *ch* is a space character. Otherwise returns false.

- See CharacterTest.java,CustomerNumber.java.



```java
import javax.swing.JOptionPane;

/**
 * Demonstrates some of the Character class's character testing methods.
 * @author Tony Gaddis
 * @version 1.0
 */
public class CharacterTest{
    /**
     * Demonstrates some of the Character class's character testing methods.
     * @param args A reference to a string array containing command-line arguments
     */
   public static void main(String[] args){
      String input;
      char ch;

      // Get a character from the user.
      input = JOptionPane.showInputDialog("Enter " + "any single character.");
      ch = input.charAt(0);

      // Test the character.
      if (Character.isLetter(ch)){
         JOptionPane.showMessageDialog(null, "That is a letter.");
         if (Character.isLowerCase(ch)){
             JOptionPane.showMessageDialog(null, "That is a lowercase letter.");
         }else if (Character.isUpperCase(ch)){
             JOptionPane.showMessageDialog(null,  "That is an uppercase letter.");
          }
      }else if (Character.isDigit(ch)){
         JOptionPane.showMessageDialog(null, "That is a digit.");
      }else if (Character.isSpaceChar(ch)){
         JOptionPane.showMessageDialog(null, "That is a space.");
      }else if (Character.isWhitespace(ch)){
         JOptionPane.showMessageDialog(null, "That is a whitespace character.");
      }

      System.exit(0);
   }
}
```



```
import javax.swing.JOptionPane;

/**
 * Tests a customer number to verify that it is in the proper format.
 * @author Tony Gaddis
 * @version 1.0
 */
public class CustomerNumber{
    /**
     * Tests a customer number to verify that it is in the form of LLLNNNN.
     *  - LLL = letters
     *  - NNNN = numbers
     * @param args A reference to a string array containing command-line arguments
     */
   public static void main(String[] args){
      String input;

      // Get a customer number.
      input = JOptionPane.showInputDialog("Enter " +
         "a customer number in the form LLLNNNN\n" +
         "(LLL = letters and NNNN = numbers)");

      // Validate the input.
      if (isValid(input)){
         JOptionPane.showMessageDialog(null, "That's a valid customer number.");
      }else {
         JOptionPane.showMessageDialog(null,
             "That is not the proper format of a " +
             "customer number.\nHere is an " +
             "example: ABC1234");
      }

      System.exit(0);
   }
```

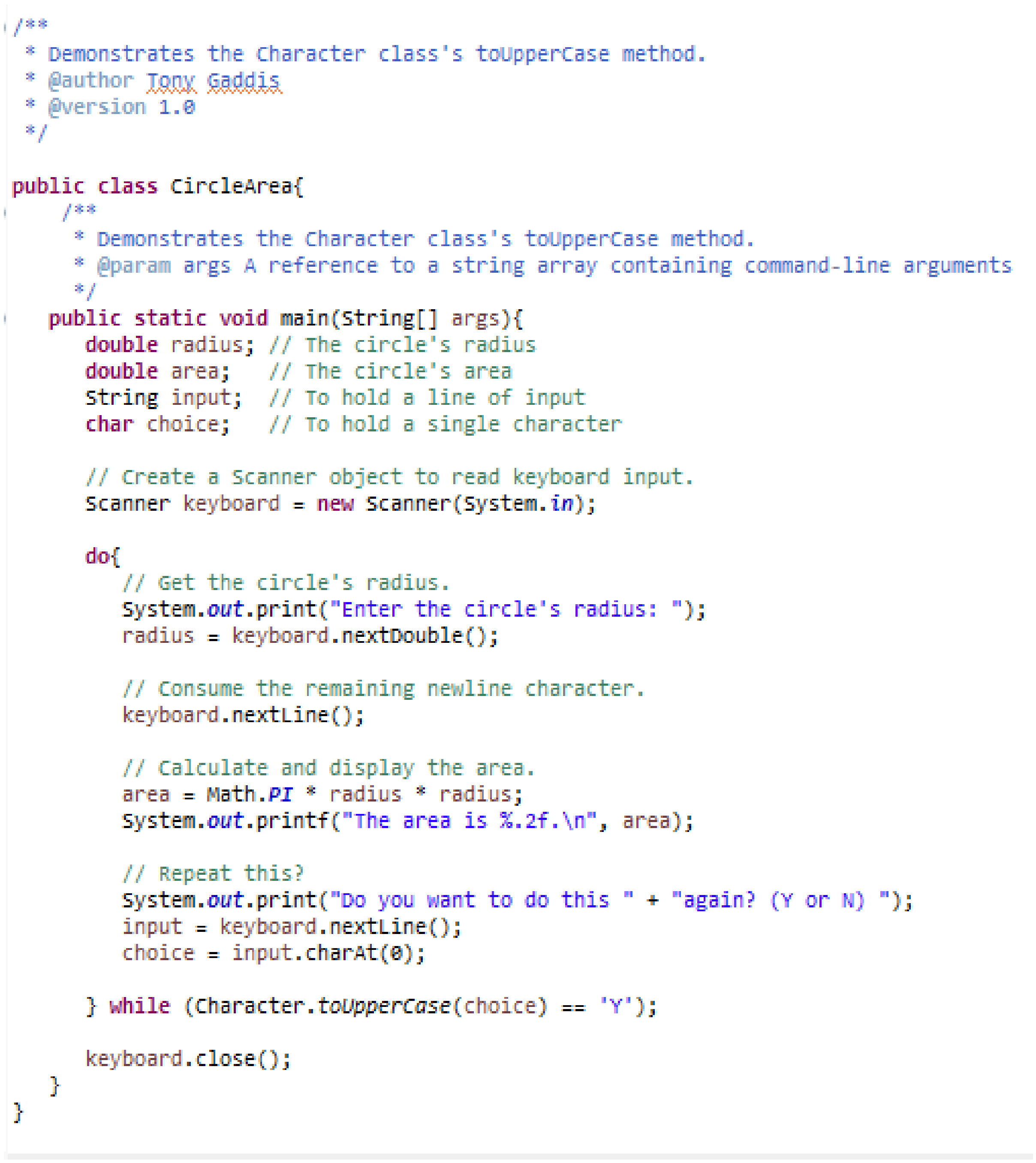
```java
   /**
    * Determines whether a String is a valid customer number. If so, it
    * returns true.
    * @param custNumber The String to test
    * @return true if valid, otherwise false
    */
   private static boolean isValid(String custNumber) {
      boolean goodSoFar = true;
      int i = 0;

      // Test the length.
      if (custNumber.length() != 7)
         goodSoFar = false;

      // Test the first three characters for letters.
      while (goodSoFar && i < 3){
         if (!Character.isLetter(custNumber.charAt(i)))
            goodSoFar = false;
         i++;
      }

      // Test the last four characters for digits.
      while (goodSoFar && i < 7){
         if (!Character.isDigit(custNumber.charAt(i)))
            goodSoFar = false;
         i++;
      }

      return goodSoFar;
   }
}
```

Character Testing and Conversion With The `Character` Class

- The `Character` class provides two methods that will change the case of a character.

Method	Description
`char toLowerCase(char ch)`	Returns the lowercase equivalent of the argument passed to *ch*.
`char toUpperCase(char ch)`	Returns the uppercase equivalent of the argument passed to *ch*.

- See CircleArea.java.



```java
/**
 * Demonstrates the Character class's toUpperCase method.
 * @author Tony Gaddis
 * @version 1.0
 */

public class CircleArea{
    /**
     * Demonstrates the Character class's toUpperCase method.
     * @param args A reference to a string array containing command-line arguments
     */
   public static void main(String[] args){
      double radius; // The circle's radius
      double area;   // The circle's area
      String input;  // To hold a line of input
      char choice;   // To hold a single character

      // Create a Scanner object to read keyboard input.
      Scanner keyboard = new Scanner(System.in);

      do{
         // Get the circle's radius.
         System.out.print("Enter the circle's radius: ");
         radius = keyboard.nextDouble();

         // Consume the remaining newline character.
         keyboard.nextLine();

         // Calculate and display the area.
         area = Math.PI * radius * radius;
         System.out.printf("The area is %.2f.\n", area);

         // Repeat this?
         System.out.print("Do you want to do this " + "again? (Y or N) ");
         input = keyboard.nextLine();
         choice = input.charAt(0);

      } while (Character.toUpperCase(choice) == 'Y');

      keyboard.close();
   }
}
```

Tokenizing Strings

Tokenizing Strings

- Use `StringTokenizer` object to tokenize a string.
- Constructor:

```
public StringTokenizer(String str,
     String delim, boolean returnDelims)
```

Parameters:

`str` - a string to be parsed.

`delim` - the delimiters.

`returnDelims*` - flag indicating whether to return the delimiters as tokens.

* this parameter is optional.

Tokenizing Strings

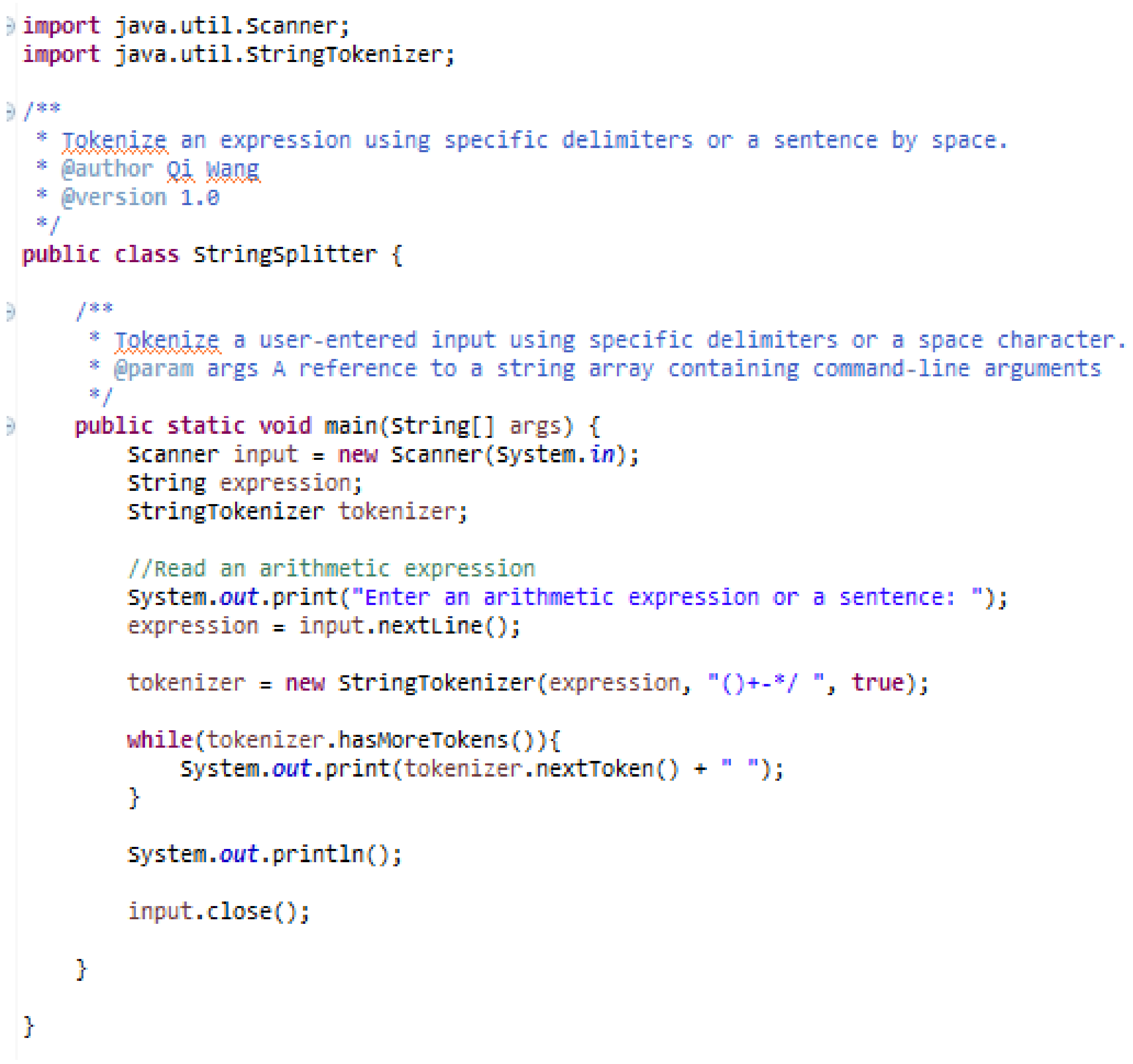
- If the `returnDelims` flag is true, then the delimiter characters are also returned as tokens. Each delimiter is returned as a string of length one.
- If the `returnDelims` flag is false, the delimiter characters are skipped and only serve as separators between tokens.

Tokenizing Strings

- For example,

```
 //Separate tokens with a space character, the
 //delimiter,  which is not returned as a token.
 StringTokenizer tokenizer = new
   StringTokenizer(expression, " ");


  //Separate tokens with delimters +,-,*,/,(,), or a
 //space character, and return the delimiter as a
 //token( a string of length one).
 StringTokenizer tokenizer = new
   StringTokenizer(expression, "+-*/() ", true);
```

- See StringSplitter.java.



```java
import java.util.Scanner;
import java.util.StringTokenizer;

/**
 * Tokenize an expression using specific delimiters or a sentence by space.
 * @author Qi Wang
 * @version 1.0
 */
public class StringSplitter {

    /**
     * Tokenize a user-entered input using specific delimiters or a space character.
     * @param args A reference to a string array containing command-line arguments
     */
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String expression;
        StringTokenizer tokenizer;

        //Read an arithmetic expression
        System.out.print("Enter an arithmetic expression or a sentence: ");
        expression = input.nextLine();

        tokenizer = new StringTokenizer(expression, "()+-*/ ", true);

        while(tokenizer.hasMoreTokens()){
            System.out.print(tokenizer.nextToken() + " ");
        }

        System.out.println();

        input.close();

    }

}
```

Summary

- Introduction to Wrapper Classes
- Wrapper Classes
  - Numeric Data Types and the Character Class
- Tokenizing Strings:
  - The *StringTokenizer* Class