Chapter 9: File IO

CS 2070

April 1, 2018

Introduction to Wrapper Classes

Character Testing and Conversion with the Character Class

More String Methods

The StringBuilder Class

Tokenizing Strings

Wrapper Classes for the Numeric Data Types

Focus on Problem Solving: The TestScoreReader Class

Introduction to Wrapper Classes

Introduction to Wrapper Classes

- Java provides 8 primitive data types. (Name them.)
- 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.

Primitive Wrapper Classes

Notice that a few of these wrapper classes (Integer and Character) are full words compared to their primitive counterparts.

- byte becomes Byte
- short becomes Short
- int becomes Integer
- long becomes Long
- float becomes Float
- double becomes Double
- boolean becomes Boolean
- char becomes Character

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

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

Character Class Static Methods

- boolean isDigit(char ch): Returns true if ch is 0 to 9
- boolean isLetter(char ch): Returns true if ch is a-z or A-Z.
- boolean isLetterOrDigit(char ch): Returns true if ch is 0-9, a-z, or A-Z.
- boolean isLowerCase(char ch): Returns true if ch is a-z.
- boolean isUpperCase(char ch): Returns true if ch is A-Z.
- boolean isSpaceChar(char ch): Returns true if ch is a space, tab, newline, carriage return, or the vertical tab.

More String Methods

The Character Class

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

- char toLowerCase(char ch): Returns the lower case equivalent of ch (or returns ch if not possible)
- char toUpperCase(char ch): Returns the upper case equivalent of ch (or returns ch if not possible)

Substrings

- The String class provides several methods that search for a string inside of a string.
- A substring is a string that is part of another string.

Some of the substring searching methods provided by the String class:

- boolean startsWith(String str)
- boolean endsWith(String str)
- boolean regionMatches(int start, String str, int start2, int n)
- boolean regionMatches(boolean ignoreCase, int start, String str, int start2, int n)

Searching Strings

The startsWith method determines whether a string begins with a specified substring.

```
String str = "Four score and seven years ago";
if (str.startsWith("Four"))
  System.out.println("The string starts with Four.");
else
  System.out.println("The string does not start with Four.'
```

- str.startsWith("Four") returns true because str does begin with "Four".
- startsWith is a case sensitive comparison.

Searching Strings

The endsWith method determines whether a string ends with a specified substring.

```
String str = "Four score and seven years ago";
if (str.endsWith("ago"))
  System.out.println("The string ends with ago.");
else
  System.out.println("The string does not end with ago.");
```

The endsWith method also performs a case sensitive comparison.

Searching Strings

- The String class provides methods that will if specified regions of two strings match.
  - regionMatches(int start, String str, int start2, int n)
    - returns true if the specified regions match or false if they don't
    - Case sensitive comparison
  - regionMatches(boolean ignoreCase, int start, String str, int start2, int n)
    - If ignoreCase is true, it performs case insensitive comparison

Searching Strings

The String class also provides methods that will locate the position of a substring.

- indexOf
  - returns the first location of a substring or character in the calling String Object.
- lastIndexOf
  - returns the last location of a substring or character in the calling String Object.

Searching Strings

```
String str = "Four score and seven years ago";
int first, last;
first = str.indexOf('r');
last = str.lastIndexOf('r');
System.out.println("The letter r first appears at "
                    + "position " + first);
System.out.println("The letter r last appears at "
                   + "position " + last);
```

Searching Strings

```
String str = "and a one and a two and a three";
int position;
System.out.println("The word and appears at the "
                   + "following locations.");

position = str.indexOf("and");
while (position != -1) {
  System.out.println(position);
  position = str.indexOf("and", position + 1);
}
```

Extracting Substrings

The String class provides methods to extract substrings in a String object. The substring method returns a substring beginning at a start location and an optional ending location.

```
String fullName = "Cynthia Susan Smith";
String lastName = fullName.substring(14);
System.out.println("The full name is "
                   + fullName);
System.out.println("The last name is "
                   + lastName);
```

This should display that the last name is “Smith”.

Returning Modified Strings

The String class provides methods to return modified String objects.

- concat: Returns a String object that is the concatenation of two String objects.
- replace: Returns a String object with all occurrences of one character being replaced by another character.
- trim: Returns a String object with all leading and trailing whitespace characters removed.

The valueOf Methods

- The String class provides several overloaded valueOf methods.
- They return a String object representation of
  - a primitive value or
  - a character array.

Code.

```
String.valueOf(true) will return "true".
String.valueOf(5.0) will return "5.0".
String.valueOf(‘C') will return "C".
```

The valueOf Methods

```
boolean b = true;
char [] letters = { 'a', 'b', 'c', 'd', 'e' };
double d = 2.4981567;
int i = 7;
System.out.println(String.valueOf(b)); // True
System.out.println(String.valueOf(letters)); // abcde
System.out.println(String.valueOf(letters, 1, 3)); // bcd
System.out.println(String.valueOf(d)); // 2.4981567
System.out.println(String.valueOf(i)); // 7
```

The StringBuilder Class

The StringBuilder Class

- The StringBuilder class is similar to the String class.
- However, you may change the contents of StringBuilder objects.
  - You can change specific characters,
  - insert characters,
  - delete characters, and
  - perform other operations.
- A StringBuilder object will grow or shrink in size, as needed, to accommodate the changes.

StringBuilder Constructors

- StringBuilder()
  - This constructor gives the object enough storage space to hold 16 characters.
- StringBuilder(int length)
  - This constructor gives the object enough storage space to hold length characters.
- StringBuilder(String str)
  - This constructor initializes the object with the string in str.
  - The object will have at least enough storage space to hold the string in str.

Other StringBuilder Methods

The String and StringBuilder also have common methods:

```
char charAt(int position)
void getChars(int start, int end,
          char[] array, int arrayStart)
int indexOf(String str)
int indexOf(String str, int start)
int lastIndexOf(String str)
int lastIndexOf(String str, int start)
int length()
String substring(int start)
String substring(int start, int end)
```

Appending to a StringBuilder Object

- The StringBuilder class has several overloaded versions of a method named append.
- They append a string representation of their argument to the calling object's current contents.
- The general form of the append method is:
  object.append(item);
  - where object is an instance of the StringBuilder class and item is:
    - a primitive literal or variable.
    - a char array, or
    - a String literal or object.

Appending to a StringBuilder Object

After the append method is called, a string representation of item will be appended to object's contents.

```
StringBuilder str = new StringBuilder();

str.append("We sold ");
str.append(12);
str.append(" doughnuts for $");
str.append(15.95);

System.out.println(str);
```

This code will produce the following output:

```
We sold 12 doughnuts for $15.95
```

Appending to a StringBuilder Object

- The StringBuilder class also has several overloaded versions of a method named insert
- These methods accept two arguments:
  - an int that specifies the position to begin insertion, and
  - the value to be inserted.
- The value to be inserted may be
  - a primitive literal or variable.
  - a char array, or
  - a String literal or object.

Appending to a StringBuilder Object

The general form of a typical call to the insert method.

```
object.insert(start, item);
```

where object is an instance of the StringBuilder class, start is the insertion location, and item is:

- a primitive literal or variable.
- a char array, or
- a String literal or object.

Replacing a Substring in a StringBuilder Object

- The StringBuilder class has a replace method that replaces a specified substring with a string.

The general form of a call to the method:

```
object.replace(start, end, str);
```

- start is an int that specifies the starting position of a substring in the calling object, and
- end is an int that specifies the ending position of the substring. (The starting position is included in the substring, but the ending position is not.)
- The str parameter is a String object.

After the method executes, the substring will be replaced with str.

Replacing a Substring in a StringBuilder Object

The replace method in this code replaces the word “Chicago” with “New York”.

```
StringBuilder str = new StringBuilder(
          "We moved from Chicago to Atlanta.");
str.replace(14, 21, "New York");
System.out.println(str);
```

The code will produce the following output:

```
We moved from New York to Atlanta.
```

Other StringBuilder Methods

The StringBuilder class also provides methods to set and delete characters in an object.

```
StringBuilder str = new StringBuilder(
                          "I ate 100 blueberries!");
// Display the StringBuilder object.
System.out.println(str);
// Delete the '0'.
str.deleteCharAt(8);
// Delete "blue".
str.delete(9, 13);
// Display the StringBuilder object.
System.out.println(str);
// Change the '1' to '5'
str.setCharAt(6, '5');
// Display the StringBuilder object.
System.out.println(str);
```

Other StringBuilder Methods

- The toString method
- You can call a StringBuilder's toString method to convert that StringBuilder object to a regular String

Tokenizing Strings

Tokenizing Strings

- Use the String class's split method
- Tokenizes a String object and returns an array of String objects
- Each array element is one token.

Code.

```
// Create a String to tokenize.
String str = "one two three four";
// Get the tokens from the string.
String[] tokens = str.split(" ");
// Display each token.
for (String s : tokens)
   System.out.println(s);
```

Wrapper Classes for the Numeric Data Types

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

- Recall from Chapter 2, we converted String input (from JOptionPane) into numbers. 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);
```

toBinaryString, toHexString, and toOctalString Methods

The Integer and Long classes have three additional methods: toBinaryString, toHexString, and toOctalString

```
int number = 14;
System.out.println(Integer.toBinaryString(number)); // 111(
System.out.println(Integer.toHexString(number));    // e
System.out.println(Integer.toOctalString(number));  // 16
```

MIN_VALUE and MAX_VALUE

The numeric wrapper classes each have a set of static final variables MIN_VALUE and 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 nay think this is an error, but because number is a wrapper class variable, autoboxing occurs.

Autoboxing and Unboxing

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.

Code.

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

Autoboxing and unboxing allow you to conveniently use ArrayLists with primitives.

Focus on Problem Solving: The TestScoreReader Class

Problem Solving

Dr. Harrison keeps student scores in an Excel file. This can be exported as a comma separated text file. Each student's data will be on one line. We want to write a Java program that will find the average for each student. (The number of students changes each year.)