Fundamental Programming Structures in Java: Strings

In this class we will cover how to:

- Manipulate characters
- Declare a String object
- Compare String values
- Use other String methods
- Convert Strings to numbers
- Use the StringBuffer class
- Use of the StringTokenizer class

Manipulating Characters

- Character class- Contains standard methods for testing the values of characters, such as letters or digits
  - Primitive data type char is used to hold any single character
  - The Character class is defined in java.lang.Object
  - Automatically imported into every program you write
  - What is the difference between char and Character?

Table 7-1 Common methods of the Character class

Method	Description
isUpperCase()	Tests if character is uppercase
toUpperCase()	Changes a lowercase character to uppercase
isLowerCase()	Tests if character is lowercase
toLowerCase()	Changes an uppercase character to lowercase
isDigit()	Returns `true` if the argument is a digit (0-9) and `false` otherwise
isLetter()	Returns `true` if the argument is a letter and `false` otherwise
isLetterOrDigit()	Returns `true` if the argument is a letter or digit and `false` otherwise
isWhitespace()	Returns `true` if the argument is whitespace and `false` otherwise. (This includes the space, tab, newline, carriage return, and form feed.)

Example of Using Character Class

- Source code:
  - public class CharacterUse {
    public static void main(String[] args) {
    char c = 'c';
    System.out.println("uppercase is " +
    Character.toUpperCase(c));
    }
    }
- Result:
  - uppercase is C

Declaring a String Object

String variable

- An object of the class String
  - The class String is defined in java.lang.String and is automatically imported into every program
- The string itself is distinct from the variable you use to refer to it
- Create a String object by using the keyword new and the String constructor method
  - String aGreeting = new String(“Hello”);
- What does it mean when someone says Strings are *immutable*?

Strings are Immutable

- Immutable- Strings and other objects that can’t be changed
  - String is a class; each created String is a class object
  - Strings are never actually changed; instead new Strings are created and String variables hold the new addresses.
  - A part of the Java system called the garbage collector will eventually discard the unused strings
- So, what does this mean for Java programmers?

Changing String Values

- Strings have no methods to let you change an existing character in a string.
  - To change a string you have to reassign the variable to a new string.
  - example:
    - String greeting = “Hello”;
      greeting = greeting.substr(0,4) + “!”;
    - Result: greeting is now “Hell!”

Comparing String Values

- Because String variables hold memory addresses, you cannot make a simple comparison
- The String class provides a number of methods
  - equals() method
  - equalsIgnoreCase() method
  - compareTo() method

Comparing String Values

- equals() method- Evaluates the contents of two String objects to determine if they are equivalent
  - This method returns true if the two String objects have identical contents
  - Can take either a variable String object or a literal string as its argument
    - s1.equals(s2);
    - s1.equals(“Hello”);

Comparing String Values

- equalsIgnoreCase() method- Similar to the equals() method
  - Ignores case when determining if two Strings are equivalent
  - Useful when users type responses to prompts in your program

Comparing String Values

- compareTo() method- Used to compare two Strings
  - It provides additional information to the user in the form of an integer value
  - Returns zero only if the two Strings hold the same value
  - If there is any difference between the Strings, a negative number is returned if the calling object is “less than” the argument
  - A positive number is returned if the calling object is “more than” the argument
  - Strings are considered “less than” or “more than” each other based on their Unicode values

Using Other String Methods

- There are additional String methods available in the String class
  - toUpperCase() and toLowerCase() convert any String to its uppercase or lowercase equivalent
  - substr() returns a portion of a String
    - e.g. String s = “Hello”;
      String s2 = s.substring(1,4);
      What is the value of s2?
  - What are some other String methods?
  - See the API documentation for a complete list of String methods

Using Other String Methods

- indexOf() method determines whether a specific character occurs within a String
  - If it does contain that character, the method returns the position of the character
  - The first position of a String is zero, not one
  - The return value is -1 if the character is not in the String

Using Other Methods

- charAt() method requires an integer argument which indicates the position of the character that the method returns
- endsWith() method and the startsWith() method each take a String argument and return true or false if a string object does or does not end or start with the specified argument
- replace() method allows you to replace all occurrences of some character within a String
- toString() method converts any primitive type to a String

Concatenation

- Concatenation- Joining strings

- Examples:
  - “45” + “36” = “4536”
  - String name = “Cartman”;
    System.out.println(“Name is “ + name);

Converting Strings to Numbers

To convert a String to a number

- Use the Integer class
  - Part of java.lang
- parseInt() method is part of the Integer class and takes a String argument and returns its integer value
  - Use the integer value as any other integer
  - Useful for taking in integer parameters
  - e.g.
```
String s = args[0];
int aNumber = Integer.parseInt(s);
aNumber++;
```
- To convert a String object to a double value you must use the Double class

Learning about the StringBuffer Class

- A String class limitation is that the value of a String is fixed after the string is created
- To circumvent these limitations you can use the StringBuffer class

StringBuffer

- StringBuffer is an alternative to the String class
- Part of the java.lang package
- Must use the keyword new and provide an initializing value between parentheses
- A StringBuffer object contains a memory block called a buffer which might not contain a string

StringBuffer

- The length of the String may not be the same as the length of the buffer
- The length of the buffer is referred to as the capacity of the StringBuffer object
- You can change the length of a String in a StringBuffer object with the setLength() method
- When the StringBuffer object's length is longer than the String it holds, the extra characters contain '\u0000'
- If you use the setLength() method to specify a length shorter than its String, the string is truncated

StringBuffer

- Using StringBuffer objects provide more flexibility than String objects because you can insert or append new contents into a StringBuffer

StringBuffer Constructors

- The StringBuffer class provides 3 constructors
  - public StringBuffer() constructs a StringBuffer with no characters and a default size of 16 characters
  - public StringBuffer(int length) constructs a StringBuffer with no characters and a capacity specified by length
  - public StringBuffer(String s) contains the same characters as those stored in the String object s

```
public class EventStringBuffer
{
  public static void main(String[] args)
  {
  StringBuffer eventString =
        new StringBuffer("Event Handlers Incorporated");
     int aCapacity = eventString.capacity();
     System.out.println("The capacity is " + aCapacity);

     StringBuffer philosophyString = null;
     philosophyString =
        new StringBuffer("Dedicated to making your event a most memorable one");
     int bCapacity = philosophyString.capacity();
     System.out.println("The capacity is " + bCapacity);

     eventString.setLength(50);
     System.out.println("The eventString is " + eventString + "end");

     philosophyString.setLength(30)
        System.out.println("The philosophyString is " + philosophyString);
  }
}
```

Figure 7-9 EventStringBuffer program

StringBuffer Methods

- append() method- Lets you add characters to the end of a StringBuffer object
- insert() method- Lets you add characters at a specified location within a StringBuffer object
- setCharAt() method- Use to alter just one character in a StringBuffer

classTest1.java | Java1.java

```
1
2  import java.util.Enumeration;
6
7  public class Java1 {
8
9      public static void main(String[] args) {
10
11         StringBuffer newBuffer = new StringBuffer("Hello");
12         System.out.println(newBuffer);
13         newBuffer.append(" World");
14         System.out.println(newBuffer);
15
16     }
17 }
```

Problems | Javadoc | Declaration | Console

<terminated> Java1 [Java Application] C:\Program Files (x86)\Java\jre1.8.0_60\bin\javaw.exe (Feb 1, 2018, 10:09:22 AM)

```
Hello
Hello World
```

StringBuffer Methods

- charAt() method extracts characters from a StringBuffer object
  - Accepts an argument that is the offset of the character position from the beginning of a String

StringTokenizer

- **Used to break up a string based on a certain "token"**
  - e.g. take the string "Homer, Marge, Bart, Lisa, Maggie"
    You can use the StringTokenizer to break up the string based on the comma to get five separate strings
    "Homer" "Marge" "Bart" "Lisa" and "Maggie"
- **Easier than using the "if" statement**
- **Example:**
  - ```
    String testString = "one-two-three";
    StringTokenizer st = new
    StringTokenizer(testString,"-");
    while (st.hasMoreTokens()) {
            String nextName = st.nextToken();
            System.out.println(nextName);
    }
    ```
  - Result:
    one
    two
    ```