Characters and Strings

Introduction to Programming and Computational Problem Solving

CSE 11

Lecture 4
Announcements

• Assignment 1 is due today, 11:59 PM
• Assignment 2 will be released today
  – Due April 17, 11:59 PM
Characters and strings

• Character data type (i.e., char)
• Comparing and testing characters
• String data type (i.e., String)
• Simple string methods (e.g., number of characters in a string)
• Reading a character and string from the console
Data types

• Java is a strongly typed language
  – Programmers must explicitly identify the type of every variable, method, and object
Java characters use Unicode, a 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world’s diverse languages.

- Unicode takes two bytes, preceded by \u, expressed in four hexadecimal numbers that run from \u0000 to \uFFFF
  - Unicode can represent 65536 characters
# Common and special characters

<table>
<thead>
<tr>
<th>Characters</th>
<th>Code Value in Decimal</th>
<th>Unicode Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'0' to '9'</td>
<td>48 to 57</td>
<td>\u0030 to \u0039</td>
</tr>
<tr>
<td>'A' to 'Z'</td>
<td>65 to 90</td>
<td>\u0041 to \u005A</td>
</tr>
<tr>
<td>'a' to 'z'</td>
<td>97 to 122</td>
<td>\u0061 to \u007A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Name</th>
<th>Unicode Code</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>Backspace</td>
<td>\u0008</td>
<td>8</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
<td>\u0009</td>
<td>9</td>
</tr>
<tr>
<td>\n</td>
<td>Linefeed</td>
<td>\u000A</td>
<td>10</td>
</tr>
<tr>
<td>\f</td>
<td>Formfeed</td>
<td>\u000C</td>
<td>12</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage Return</td>
<td>\u000D</td>
<td>13</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
<td>\u005C</td>
<td>92</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double Quote</td>
<td>\u0022</td>
<td>34</td>
</tr>
</tbody>
</table>
Comparing and testing characters

if (ch >= 'A' && ch <= 'Z')
    System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
    System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
    System.out.println(ch + " is a numeric character");

Relational and logical operators will be covered next lecture
Comparing and testing characters

• The Character class
  – Java 8 API documentation
    • https://docs.oracle.com/javase/8/docs/api/java/lang/Character.html
  – Java 11 API documentation
    • https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Character.html

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isDigit(ch)</td>
<td>Returns true if the specified character is a digit.</td>
</tr>
<tr>
<td>isLetter(ch)</td>
<td>Returns true if the specified character is a letter.</td>
</tr>
<tr>
<td>isLetterOrDigit(ch)</td>
<td>Returns true if the specified character is a letter or digit.</td>
</tr>
<tr>
<td>isLowerCase(ch)</td>
<td>Returns true if the specified character is a lowercase letter.</td>
</tr>
<tr>
<td>isUpperCase(ch)</td>
<td>Returns true if the specified character is an uppercase letter.</td>
</tr>
<tr>
<td>toLowerCase(ch)</td>
<td>Returns the lowercase of the specified character.</td>
</tr>
<tr>
<td>toUpperCase(ch)</td>
<td>Returns the uppercase of the specified character.</td>
</tr>
</tbody>
</table>
Casting between char and numeric data types

```java
int i = 'a'; // Same as int i = (int)'a';

char c = 97; // Same as char c = (char)97;
```
String type

• The char type only represents one character
• To represent a string of characters, use the String type
• String is a predefined class in the Java library (just like the System class and Scanner class)
  – Java 8 API documentation
    • https://docs.oracle.com/javase/8/docs/api/java/lang/String.html
  – Java 11 API documentation
    • https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html
String type

• String is a predefined class in the Java library
  String message = "Welcome to Java";

• The String type is not a primitive type; it is known as a reference type
  – Any Java class can be used as a reference type for a variable
# Simple String methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length()</td>
<td>Returns the number of characters in this string.</td>
</tr>
<tr>
<td>charAt(index)</td>
<td>Returns the character at the specified index from this string.</td>
</tr>
<tr>
<td>concat(s1)</td>
<td>Returns a new string that concatenates this string with string s1.</td>
</tr>
<tr>
<td>toUpperCase()</td>
<td>Returns a new string with all letters in uppercase.</td>
</tr>
<tr>
<td>toLowerCase()</td>
<td>Returns a new string with all letters in lowercase.</td>
</tr>
<tr>
<td>trim()</td>
<td>Returns a new string with whitespace characters trimmed on both sides.</td>
</tr>
</tbody>
</table>

- These methods can only be invoked from a specific string instance
  - These methods are called instance methods
Instance methods vs static methods

• These methods can only be invoked from a specific string instance
  – These methods are called instance methods
  – The syntax to invoke an instance method is
    referenceVariable.methodName(arguments)

• A non-instance method is called a static method
  – A static method can be invoked without using an object
    (i.e., they are not tied to a specific object instance)
  – The syntax to invoke a static method is
    ClassName.methodName(arguments)
  – For example, all the methods defined in the Math class are
    static methods

Methods will be covered next week
Getting characters from a string

String message = "Welcome to Java";
System.out.println("The first character in message is " + message.charAt(0));

Indices
message

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Welcome to Java

message.charAt(0) message.length() is 15 message.charAt(14)
String concatenation

String s3 = s1.concat(s2); // These two are equivalent
String s3 = s1 + s2;       // equivalent

// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
Reading a string from the console

Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
Reading a character from the console

Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
Explicit import and implicit Import

• At top of source file

```java
import java.util.Scanner; // Explicit Import

import java.util.*; // Implicit import
```
# Comparing strings

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals(s1)</td>
<td>Returns true if this string is equal to string s1.</td>
</tr>
<tr>
<td>equalsIgnoreCase(s1)</td>
<td>Returns true if this string is equal to string s1; it is case insensitive.</td>
</tr>
<tr>
<td>compareTo(s1)</td>
<td>Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1.</td>
</tr>
<tr>
<td>compareToIgnoreCase(s1)</td>
<td>Same as compareTo except that the comparison is case insensitive.</td>
</tr>
<tr>
<td>startsWith(prefix)</td>
<td>Returns true if this string starts with the specified prefix.</td>
</tr>
<tr>
<td>endsWith(suffix)</td>
<td>Returns true if this string ends with the specified suffix.</td>
</tr>
</tbody>
</table>
Substrings

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>substring(beginIndex)</code></td>
<td>Returns this string’s substring that begins with the character at the specified <code>beginIndex</code> and extends to the end of the string.</td>
</tr>
<tr>
<td><code>substring(beginIndex, endIndex)</code></td>
<td>Returns this string’s substring that begins at the specified <code>beginIndex</code> and extends to the character at index <code>endIndex - 1</code>. Note that the character at <code>endIndex</code> is not part of the substring.</td>
</tr>
</tbody>
</table>

Indices Message

```
W e l c o m e t o J a v a
```

message.substring(0, 11) message.substring(11)
Finding a character or a substring in a string

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>indexOf(ch)</td>
<td>Returns the index of the first occurrence of ch in the string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>indexOf(ch, fromIndex)</td>
<td>Returns the index of the first occurrence of ch after fromIndex in the string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>indexOf(s)</td>
<td>Returns the index of the first occurrence of string s in this string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>indexOf(s, fromIndex)</td>
<td>Returns the index of the first occurrence of string s in this string after fromIndex. Returns -1 if not matched.</td>
</tr>
<tr>
<td>lastIndexOf(ch)</td>
<td>Returns the index of the last occurrence of ch in the string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>lastIndexOf(ch, fromIndex)</td>
<td>Returns the index of the last occurrence of ch before fromIndex in this string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>lastIndexOf(s)</td>
<td>Returns the index of the last occurrence of string s. Returns -1 if not matched.</td>
</tr>
<tr>
<td>lastIndexOf(s, fromIndex)</td>
<td>Returns the index of the last occurrence of string s before fromIndex. Returns -1 if not matched.</td>
</tr>
</tbody>
</table>
Finding a character or a substring in a string

```java
int k = s.indexOf(' ');  
String firstName = s.substring(0, k);  
String lastName = s.substring(k + 1);
```
Conversion between strings and numbers

• String to number
  int intValue =
      Integer.parseInt(intString);
double doubleValue =
      Double.parseDouble(doubleString);

• Number to string
  String s = number + "";
Formatting output

• Use the printf statement
  `System.out.printf(format, items);`

• Where `format` is a string that may consist of substrings and format specifiers
  – A format specifier specifies how an item should be displayed
  – Each specifier begins with a percent sign
  – An item may be a numeric value, character, Boolean value, or a string
# Common specifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Output Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>%b</td>
<td>a boolean value</td>
<td>true or false</td>
</tr>
<tr>
<td>%c</td>
<td>a character</td>
<td>'a'</td>
</tr>
<tr>
<td>%d</td>
<td>a decimal integer</td>
<td>200</td>
</tr>
<tr>
<td>%f</td>
<td>a floating-point number</td>
<td>45.460000</td>
</tr>
<tr>
<td>%e</td>
<td>a number in standard scientific notation</td>
<td>4.5560000e+01</td>
</tr>
<tr>
<td>%s</td>
<td>a string</td>
<td>&quot;Java is cool&quot;</td>
</tr>
</tbody>
</table>

```java
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

**display**

```java
count is 5 and amount is 45.560000
```
Next Lecture

• Selections