Character and String Processing

CSX3002/ITX2001 Object-Oriented Concepts and Programming
CS4402 Selected Topic in Object-Oriented Concepts
IT2371 Object-Oriented Programming

Kwankamol Nongpong, Ph.D.
Assumption University

Character Fundamentals (1)

- Characters are internally represented as 16-bit Unicode (integers).
  - Each value represent a unique symbol (ASCII standard)
    - **`'A'`** has ASCII value of 65; **`'Z'`** has ASCII value of 90
    - **`'a'`** has ASCII value of 97; **`'z'`** has ASCII value of 122
    - **`'0'`** has ASCII value of 48; **`'9'`** has ASCII value of 57
    - There are many other symbols ('+', '-', '!', '\n', '\t', etc.)
    - Some are unprintable control characters ('\0' (null), '\a' (DEL), 12 (FF) ...
  - Character constants are written with **single quotes**
    - **('A', '$', etc)**

Character Fundamentals (2)

- Since characters are internally 16-bit integers:
  - `int num = ‘7’ - ‘0’;`
    - What is the value of num?
  - `char symbol = 65;`
    - What is the value of symbol?
  - `char symbol = ‘a’ + 10;`
    - What is the value of symbol?

Character-Handling Library

- java.lang.Character - library of Java character type
- Methods to perform tests and manipulations on characters
- Pass character as argument (most methods)

Character Handling Methods (1)

- Character-related methods
  - `static boolean isDigit(char ch)`
  - `static boolean isLetter(char ch)`
  - `static boolean isLetterOrDigit(char ch)`
  - `static boolean isLowerCase(char ch)`
  - `static boolean isUpperCase(char ch)`
  - `static boolean isWhitespace(char ch)`
  - `static char toLowerCase(char ch)`
  - `static char toUpperCase(char ch)`
  - and more…

Character-Handling Methods (2)

- Upcoming example
  - **`isLowerCase`**
    - Returns **`true`** if lowercase letter (a-z)
  - **`isUpperCase`**
    - Returns **`true`** if uppercase letter (A-Z)
  - **`toLowerCase`**
    - If passed uppercase letter, returns lowercase letter
      - **`A`** to **`a`**
    - Otherwise, returns original argument
  - **`toUpperCase`**
    - As above, but turns lowercase letter to uppercase
      - **`a`** to **`A`**

Example 1

- CharacterProcessing1.java

```
According to Character.isDigit:
8 is a digit
# is not a digit

According to Character.isLetter:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to
Character.isLetterOrDigit:
A is a letter or a digit
8 is a letter or a digit
# is not a letter or a digit
```

Example 2:

- CharacterProcessing2.java

```
According to Character.isLowerCase:
p is a lowercase letter
P is not a lowercase letter
5 is not a lowercase letter
! is not a lowercase letter

According to Character.isUpperCase:
D is an uppercase letter
d is not an uppercase letter
8 is not an uppercase letter
$ is not an uppercase letter

u converted to uppercase is U
7 converted to uppercase is 7
$ converted to uppercase is $
L converted to lowercase is l
k converted to lowercase is k
```

Fundamental of Strings in Java (1)

- String
  - Collection of characters
  - Can include anything that can be a character
    - Letters
    - Digits
    - Special symbols
  - String literal (string constants)
    - Enclosed in double quotes, for example:
      - `"I like Java"`

Fundamental of Strings in Java (2)

- The String class represents character strings.
- All string literals in Java programs are implemented as instances of this class.
  - For instance, "abc“
- Strings are constant.
  - Their values cannot be changed after they are created.
  - String buffers support mutable strings. (later)

String Declaration and Initialization

- String str = "abc";
- char data[] = {'a', 'b', 'c'};
- String str = new String(data);
- String str = new String("abc");

String-Handling Library

- We can manually manipulate strings as we know how strings in Java is internally represented.
- Alternatively, we can take advantage of standard functions that come with the language.
- String handling library **`java.lang.String`** provides functions to
  - Manipulate string data
  - Compare strings
  - Tokenize strings (separate strings into logical pieces)

String Methods

Method	Description
`length()`	Returns the length of this string
`charAt(int index)`	Returns the char value at the specified index.
`isEmpty()`	Returns true if length() = 0

String Manipulating Methods

Method	Description
`concat(String s)`	Concatenates the specified string to the end of this string.
`startsWith(String prefix)`	Tests if this string starts with the specified prefix.
`endsWith(String suffix)`	Tests if this string ends with the specified suffix.
`contains(CharSeq seq)`	Returns true if and only if this string contains the specified sequence of char values.
`substring(int beginIndex)`	Returns a new string that is a substring of this string.
`substring(int beginIndex, int endIndex)`	Returns a new string that is a substring of this string.

String Manipulating Methods

Method	Description
`replace(char oldChar, char newChar)`	Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.
`trim()`	Returns a copy of the string, with leading and trailing whitespace omitted.

String Comparison Methods

Method	Description
`compareTo(String str)`	Zero - equal lexicographically Positive – greater than the parameter Negative – less than the parameter
`compareToIgnoreCase(String str)`	Similar to compareTo() but the cases of all characters are ignored.
`==`	Checks if the two strings are the same object.
`equals(Object anObject)`	Checks if the contents are the same.

What about the operator **==** ?

Example

- StringProcessing.java

String Tokenization

- Breaking strings into tokens, separated by delimiting characters
- Tokens are usually logical units, such as words (separated by spaces)
- **`"This is my string"`** has 4 word tokens (separated by spaces)

Example

- StringTokenization.java