ICS 102 Lab04: Characters and Strings

**Objectives:**

* Learning how to use:
  + Character methods
  + String methods

**1. Characters**

Java uses the primitive 16-bit data type **char** to store characters. Java uses Unicode to represent characters. The range of **char** is 0 to 65535

A character constant is enclosed in single quotes. Examples:

'A' '&' '+' '=' 'k'

**1. 1 Escape Sequences:**

A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler.

An escape sequence is used to signal an alternative interpretation of a character or a series of characters.

The following table shows some Java escape sequences:

|  |  |
| --- | --- |
| Escape Sequence | Description |
| \t | Insert a tab in the text at this point. |
| \n | Insert a newline in the text at this point. |
| \' | Insert a single quote character in the text at this point. |
| \" | Insert a double quote character in the text at this point. |
| \\ | Insert a backslash character in the text at this point. |
| \uxxxx | Hexadecimal UNICODE character (xxxx)  Example: '\u0001' |

**Examples:**

|  |  |
| --- | --- |
| Java Statement | Output |
| System.out.println("He said \"Hello!\" to me."); | He said "Hello!" to me. |
| System.out.println("Ahmad\'s book"); | Ahmad's book |
| System.out.println("Ahmad's book"); | Ahmad's book |
| System.out.println("two\nlines"); | two  lines |

Even though **char**s are not integers, in many cases you can operate on them as if they were integers. This allows you to add two characters together, or to increment the value of a character variable, provided the result is in the allowed range for char data type i.e. 0 to 6535. For example, consider the following program fragment:

**char ch1, ch2, ch3, ch4;**

**ch1 = 88; // Unicode for X is 88**

**ch2 = 'Y';**

**++ch2; // Note ch2 = ch2 + 1; is not valid, it must be:** ch = (char) ((int)ch + 1);

**ch3 = 'B';**

**int code = ch3;**

**ch4 = (char)(ch3 + 1);**

**System.out.println("ch1 = " + ch1);**

**System.out.println("ch2 = " + ch2);**

**System.out.println("ch3 = " + ch3);**

**System.out.println("ch4 = " + ch4);**

**System.out.println("Unicode for " + ch3 + " is " + code);**

This program fragment displays the following output:

ch1 = X

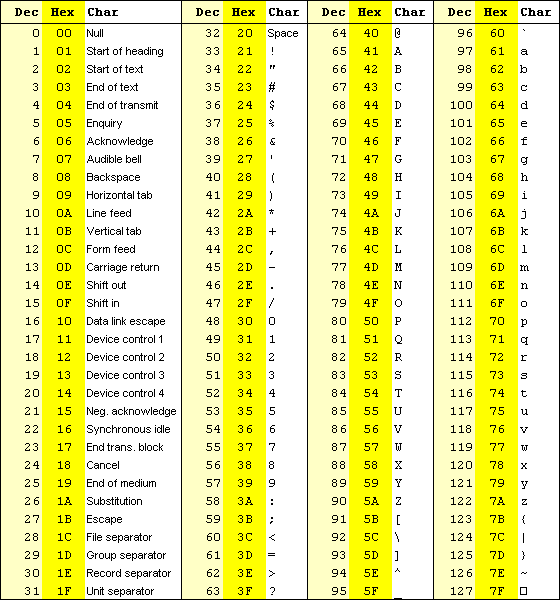
ch2 = Z

ch3 = B

ch4 = C

Unicode for B is 66

Note: The first 128 characters of Unicode correspond to the standard ASCII characters:



**1. 2 Comparison of characters**

Since characters have numeric Unicode codes ranging from 0 to 65535, they may be compared using the arithmetic comparison operators **==** (equal), **!=** (not equal), **>**, **>=**, **<**, and **<=**. The result of such a comparison is the boolean value **true** or **false**.

**Note:** ' ' < '0' < '1' < . . . < '9' < 'A' < 'B' < . . . < 'Z' < 'a' < 'b' < . . . < 'z'

Example: The code:

**boolean flag1, flag2;**

**char ch1 = 'R', ch2;**

**ch2 = ch1;**

**flag1 = 'b' < 'B'; flag2 = 'M' < 'a';**

**System.out.println("flag1 = " + flag1 + ", flag2 = " + flag2);**

**System.out.println(ch1 == ch2);**

Outputs:

**flag1 = false, flag2 = true**

**true**

**1.3 The java.lang Character wrapper class**

This class provides several methods for determining a character's category (lowercase letter, digit, etc.) and for converting characters from uppercase to lowercase and vice versa.

|  |  |  |
| --- | --- | --- |
| Method | Comment | Example |
| static boolean [**isLetter**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Character.html#isLetter(char))(char ch) | Determines if the specified character is a letter. | 'a', 'G', '\u00DF' |
| static boolean [**isLowerCase**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Character.html#isLowerCase(char))(char ch) | Determines if the specified character is a lowercase character. | 'a', 'j', 'w', '\u00F6' |
| static boolean [**isUpperCase**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Character.html#isUpperCase(char))(char ch) | Determines if the specified character is an uppercase character. | 'F', 'Q', 'E', '\u00DE' |
| static boolean [**isDigit**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Character.html#isDigit(char))(char ch) | Determines if the specified character is a digit. | '0', '4', '7' |
| static boolean [**isLetterOrDigit**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Character.html#isLetterOrDigit(char))(char ch) | Determines if the specified character is a letter or digit. | 'm', 'T', '6', '9' |
| static boolean [**isSpaceChar**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Character.html#isSpaceChar(char))(char ch) | Determines if the specified character is a Unicode space character. | **' '** |
| static boolen **isWhitespace(char ch)** | Determines if the specified character is a Unicode whitespace character. | **' ',** '\n', '\t', '\r' |
| static boolean [**isISOControl**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Character.html#isISOControl(char))(char ch) | Determines if the specified character is an ISO control character. | '\n', '\t', '\r' |
| static char [**toLowerCase**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Character.html#toLowerCase(char))(char ch) | Returns the lowercase of the character argument if possible. |  |
| static char [**toUpperCase**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Character.html#toUpperCase(char))(char ch) | Returns the uppercase of the character argument if possible. |  |

**Examples:**

**boolean test1= Character.isDigit('K'); // test1 = false**

**boolean test2 = Character.isLowerCase('b'); // test2 = true**

**char ch1 = Character.toLowerCase('H'); // ch1 = 'h'**

**char ch2 = Character.toUpperCase('j'); // ch2 = 'J'**

**2. Strings**

Strings are a sequence of characters. In the Java programming language, strings are objects.

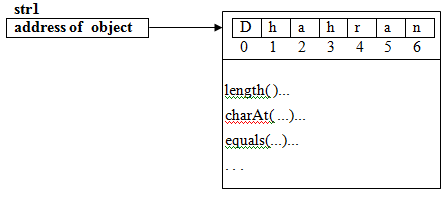
The Java platform provides the [String](http://docs.oracle.com/javase/7/docs/api/java/lang/String.html) class, in the **java.lang** package, to create and manipulate strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

**2.1 String Declarations and object creation**

There are many ways of creating String objects. Two of the ways are using the **new** operator and using a String literal. These are depicted in the following examples:

|  |  |  |
| --- | --- | --- |
| Statement | Effect | Comment |
| String str1 = **new** String(“Dhahran”);  Or  String str1;  str1 = **new** String(“Dhahran”); | Creates a new String object with the value **Dhahran** in heap memory and assigns the starting address of the object to the reference variable **str1.** The object is created even if there is already another object with the value **Dhahran** |  |
| String str1 = “Dhahran”;  Or  String str1;  str1 = “Dhahran”; | The JVM checks an area of heap memory called String pool. If the String pool contains an object with the value **Dhahran**, no new object is created; the starting address of the object is assigned to the reference variable **str1**. If there is no such object, a new object is created in the pool and its starting address is assigned to **str1** | This way of creating an object without using the **new** operator is only applicable to String |

In a String object, the characters are indexed starting from 0 as shown below:

****

|  |  |
| --- | --- |
| Statements | Effect |
| String s1 = “Hello”;  String s2 = “Hello”;  String s3 = “Hello”;  String s4 = new String(“Hello”);  String s5 = new String(“Hello”); |  |

String objects are *immutable*, which means that once created, their values cannot be changed; however, references to String objects may be changed. In Java, the **StringBuffer** class supports mutable strings (i.e. strings that can be modified)

.

|  |  |
| --- | --- |
| Statement | Effect |
| **String** str1 = **new String**("Dhahran"); |  |
| **String** str2 = **new String**("Jubail"); |  |
| str1 = str2; |  |

Note: Automatic garbage collection will remove unreferenced objects from the heap memory.

**2. 2 String Concatenation**

The String concatenation operator + is used to create a new String object that is obtained by joining a String to another String, or to an object, or to a primitive value. If one of the two operands in a concatenation operation is not a String, the + operator will convert it to a String before concatenating.

**Note:** The concatenation operator +, has the same priority and associativity as the addition, +, and subtraction, - , operators.

**Examples:**

|  |  |
| --- | --- |
| Java Statements | Final value of str3 |
| String str1 = "Problem"; String str2 = "Solving"; String str3 = str1 + str2; | "ProblemSolving" |
| String str3 = "Hello" + " " +              "there"; | "Hello there" |
| String str3 = "Ahmad"; str3 += " Muhsin"; | "Ahmad Muhsin" |
| String str1 = "ICS "; int z = 102; String str3 = str1 + z; | "ICS 102" |
| String str3 = 5402 + " " +              "Athman Street"; | "5402 Athman Street" |

|  |  |
| --- | --- |
| String str3;  str3 = 12 + 14 + " = " + 12 + 14; | 26 = 1214 |
| String str3 = "It is "; boolean y = true; str3 += y; | "It is true" |
| String str3 = "" + 'a' + 'b' ; | "ab" |

The **String** class also includes a method for concatenating two strings:

**string1.concat(string2)**

This returns a new string that contains the value of **string1** with the value of **string2** added to it at the end.

You can also use the **concat( )** method with string literals, as in:

* "The course title is ".concat("Introduction to Programming")
* "to".concat("get").concat("there")

**2.3 String Comparisons**

There are six ways to compare two initialized Strings, referenced by **str1** and **str2**:

|  |  |
| --- | --- |
| Comparison | Effect |
| str1.equals(str2) | Returns **true** if the contents of the String objects referenced by **str1** and **str2** are equal; otherwise it returns **false**. |
| str1.equalsIgnoreCase(str2) | Similar to **equals** but the string contents are compared without case sensitivity. |
| str1.compareTo(str2) | Compares a String to another String and returns an **int** less than zero, zero, or greater than zero indicating, respectively, whether **this** String (the String referenced by **str1**) is smaller, equal to or larger than the other String.  For example:  **"cat".compareTo("dog")** returns a negative value;  **"cat".compareTo("cat")** returns 0;  **"cat".compareTo("ant")** returns a positive value. |
| str1.compareToIgnoreCase(str2) | Similar to **campareTo** but the strings are compared without case sensitivity. |
| str1 **==** str2 | **true** if **str1** and **str2** refer to the same String object; otherwise it returns **false**.  **Note:** If **str1 == str2** is **true** then **str1.equals(str2)** is also **true** |
| str1 **!=** str2 | **true** if **str1** and **str2** DO NOT refer to the same String object; otherwise, **false**.  **Note:** If **str1 != str2** is **true** then **str1.equals(str2)** may be **true** or it may be **false** |

**String Comparison Example1:**

|  |  |
| --- | --- |
| Statements | Effect |
| String s1 = “Hello”;  String s2 = “Hello”;  String s3 = “Hello”;  String s4 = new String(“Hello”);  String s5 = new String(“Hello”); |  |

boolean flag1, flag2, flag3, flag4;

flag1 = s1 **==** s2; // flag1 = true

flag2 = s1.equals(s2); // flag2 = true

flag3 = s4 **==** s5; // flag3 = false

flag4 = s4.equals(s5); // flag4 = true

**String Comparison Example2:**

String s = "a";  
 s += "b"; // String "ab" not created in String pool  
 String r = "ab";  
 System.out.println(s + " **==** " + r + " is " + (s **==** r));

String s1 = "Hello";  
 String s2 = "Hello";  
 System.out.println(s1 + " **==** " + s2 + " is " + (s1 **==** s2));

Output:

ab **==** ab is false

Hello **==** Hello is true

**String Comparison Example3:**

String s1, s2, s3, s4;

int c1, c2, c3;

s1 = "A big house";

s2 = "A big mouse";

c1 = s1.compareTo(s2);

// String referenced by s1 is less than that referenced by s2 because

// 'h' < 'm' 🡪 c1 < 0

s3 = "a";

s4 = "A man, a plan, a canal; Panama!";

c2 = s3.compareTo(s4);

// String referenced by s3 is greater than that referenced by s4 because // 'a' > 'A' 🡪 c2 > 0

c3 = "Riyadh".compareTo("Riyadh is a big city");

// c3 < 0 because the first 6 characters of the strings match, and the

// length of Riyadh is smaller

**2. 4 Other String methods**

The **String** class includes methods for examining individual characters of strings, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase.

**2.4.1 String length method**

|  |  |  |
| --- | --- | --- |
| Method | Description | Example using:  **String str1 = "Orange juice ice";** |
| int [**length**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#length())() | Returns the length of this string. | int strLen = str1.length( );  // strLen = 16; |

**2.4.2 String index methods**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| **'O'** | **'r'** | **'a'** | **'n'** | **'g'** | **'e'** | **'**  **'** | **'j'** | **'u'** | **'i'** | **'c'** | **'e'** | **' '** | **'i** **'** | **'c'** | **'e'** |

|  |  |  |
| --- | --- | --- |
| Method | Description | Example using:  **String str1 = "Orange juice ice";** |
| char [**charAt**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#charAt(int))(int index) | Returns the char value at the specified index. (Error if **index** is not valid) | **char ch = str1.charAt(1);**  // ch = 'r' |
| int [**indexOf**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#indexOf(int))(char ch) | Returns the index within this string of the first occurrence of the specified character (-1 if not found) | **int k = str1.indexOf('e');**  // k = 5 |
| int [**indexOf**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#indexOf(java.lang.String))([String](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html) str) | Returns the index within this string of the start of the first occurrence of **str** (-1 if not found). | **int k = str1.indexOf("ice");**  // k = 9 |
| int [**lastIndexOf**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#lastIndexOf(int))(char ch) | Returns the index within this string of the last occurrence of **ch** (-1 if not found). | **int k = str1.lastIndexOf('e');**  // k = 15 |
| int [**lastIndexOf**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#lastIndexOf(java.lang.String))([String](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html) str) | Returns the index within this string of the start of the last occurrence of **str** (-1 if not found). | **int k = str2.lastIndexOf("ice");**  // k = 13 |

**2.4.3 String substring methods**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| **'O'** | **'r'** | **'a'** | **'n'** | **'g'** | **'e'** | **'**  **'** | **'j'** | **'u'** | **'i'** | **'c'** | **'e'** | **' '** | **'i** **'** | **'c'** | **'e'** |

|  |  |  |
| --- | --- | --- |
| Method | Description | Example using:  **String str1 = "Orange juice ice";** |
| String  [**substring**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#substring(int))(int beginIndex) | Returns a new string that is a substring of this string, starting from **beginIndex** to the end of this string. (Error if **beginIndex** is not valid) | **String str2 = str1.substring(7);**  // str2 = "juice ice"  **"unhappy".substring(2)**  // returns "happy" |
| String  [**substring**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#substring(int, int))(int beginIndex, int endIndex) | Returns a new string that is a substring of this string starting from **beginIndex** to **endIndex - 1**.  (Error if any of the index is not valid) | **String str2 = str1.substring(2, 4);**  // str2 = "an"  **"smiles".substring(1, 5)**  // returns "mile" |

**2.4.4 String boolean methods**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| **'O'** | **'r'** | **'a'** | **'n'** | **'g'** | **'e'** | **'**  **'** | **'j'** | **'u'** | **'i'** | **'c'** | **'e'** | **' '** | **'i** **'** | **'c'** | **'e'** |

|  |  |  |
| --- | --- | --- |
| Method | Description | Example using:  **String str1 = "Orange juice ice";** |
| boolean [**isEmpty**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#isEmpty())( ) | Returns true if, and only if, [**length( )**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#length()) is 0. | **boolean t1 = str1.isEmpty( );**  // t1 = false  **boolean t2 = "".isEmpty( );**  //t2 = true |
| boolean  [**endsWith**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#endsWith(java.lang.String))([**String**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html) suffix) | Tests if this string ends with the specified suffix. | boolean t1 = **str1.endsWith("space");**  // t1 = false  **boolean t2 = str1.endsWith("ice");**  // t2 = true |
| boolean  [**startsWith**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#startsWith(java.lang.String))([**String**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html) prefix) | Tests if this string starts with the specified prefix. | **boolean t1 = str1.startsWith("range");**  // t1 = false  **boolean t2 = str1.startsWith("Orange ju");**  // t2 = true |
| boolean  [**startsWith**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#startsWith(java.lang.String, int))([String](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html) prefix, int toffset) | Tests if the substring of this string beginning at the specified index starts with the specified prefix. | **boolean t2 = str1.startsWith("juice", 7);**  // t2 = true |
| boolean [**contains**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#contains(java.lang.CharSequence))([**String**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\CharSequence.html) str) | Returns **true** if and only if this string contains the string **str**. | **boolean t1 = str1.contains("space");**  // t1 = false  **boolean t2 = str1.contains("ran");**  // t2 = true |
| boolean  [**startsWith**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#startsWith(java.lang.String))([**String**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html) prefix) | Tests if this string starts with the specified prefix. | **boolean t1 = str1.startsWith("range");**  // t1 = false  **boolean t2 = str1.startsWith("Orange ju");**  // t2 = true |
| boolean  [**startsWith**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#startsWith(java.lang.String, int))([String](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html) prefix, int toffset) | Tests if the substring of this string beginning at the specified index starts with the specified prefix. | **boolean t2 = str1.startsWith("juice", 7);**  // t2 = true |
| boolean [**contains**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#contains(java.lang.CharSequence))([**String**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\CharSequence.html) str) | Returns **true** if and only if this string contains the string **str**. | **boolean t1 = str1.contains("space");**  // t1 = false  **boolean t2 = str1.contains("ran");**  // t2 = true |

**2.4.5 Other String methods that usually return a new modified String object**

|  |  |  |
| --- | --- | --- |
| Method | Description | Example using:  **String str1 = "Orange juice ice";** |
| String [**toLowerCase**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#toLowerCase())( ) | Returns a new string in which all characters in this string are in lower-case. | **String str2 = str1.toLowerCase( );**  // str2 = "orange juice ice" |
| String [**toUpperCase**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#toUpperCase())( ) | Returns a new string in which all characters in this string are in upper case. | String str2 = str1.toUpperCase( );  // str2 = "ORANGE JUICE ICE" |
| String [**trim**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#trim())( ) | Returns a copy of the string, with leading and trailing whitespace omitted. | **" ICS 102 ".trim( )**  //returns "ICS 102" |
| String [**replace**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#replace(char, char))(char oldChar, char newChar) | Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar. If there is no replacement, a reference to this string is returned | **"turtle".replace('t', 'p')**  // returns "purple" |
| String [**replace**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html#replace(java.lang.CharSequence, java.lang.CharSequence))([**String**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\CharSequence.html) str1, [**String**](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\CharSequence.html) str2) | Returns a new string with all occurrences of **str1** replaced by **str2** | **String str3, str4;**  **str3 = "ICS 102 and ICS 103";**  **str4 = str3.replace("ICS", "EE");**  // str4 = "EE 102 and EE 103" |

Note: String method calls may be cascaded provided such cascading results in valid method calls:

**String s1 = "Hello there";**

**char ch = s1.toUpperCase().charAt(2); // ch = 'L'**

**String s2 = s1.substring(6).toUpperCase(); // s2 = "THERE"**

**3. Example**

A student userID string is formed from a name string containing the student’s first, second, and third name by taking the initials of his first, second, and last name (all in lowercase) followed by his student ID.

1. Design a pseudo-code algorithm for creating student userIDs.
2. Translate the algorithm into a Java program

Solution:

* + 1. Prompt for student name: “firstName secondName thirdName”
    2. Input: Student name
    3. Prompt for student ID
    4. Input: Student ID
    5. Extract initial of firstName
    6. Extract initial of secondName
    7. Extract initial of thirdName
    8. Convert the initials to strings: **str1**, **str2**, **str3**
    9. Concatenate the strings: **str** 🡨 **str1** join **str2** join **str3**
    10. Obtain a lowercase copy of str: **strLower** 🡨 **lowerCase(str)**
    11. Concatenate student ID to strLower: **userIDString** 🡨 **strLower** join **student ID**
    12. Display: **userIDString**
    13. Stop.

**/\* generates a user-ID for a student using his initials and student ID \*/**

**import java.util.Scanner;**

**public class MakeUserID {  
 public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.println("Enter a three word name");  
 String name = scanner.nextLine();**

**System.out.println("Enter the ID");  
 int id = scanner.nextInt();  
 int index1 = name.indexOf(' ');**

**int index2 = name.lastIndexOf(' ');**

**//extract initials  
 String initials = "" + name.charAt(0)+  
 name.charAt(index1 + 1) + name.charAt(index2 + 1);**

**//append id  
 String userID = initials.toLowerCase()+id;  
 System.out.println("Your UserID = " + userID);  
 }  
}**

**Lab Tasks**

**Task1:** Each one of the following strings is an Arabic sentence in Unicode:

"\u062a\u0639\u0637\u064a \u064a\u0648\u0646\u064a\u0643\u0648\u062f \u0631\u0642\u0645\u0627 \u0641\u0631\u064a\u062f\u0627 \u0644\u0643\u0644 \u062d\u0631\u0641"

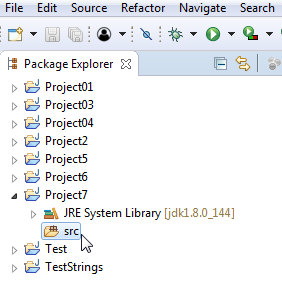
"\u0627\u0644\u0628\u0646\u0643 \u0627\u0644\u0645\u062a\u062d\u062f"

"\u0627\u0644\u0645\u0648\u0642\u0639 \u0627\u0644\u0625\u0644\u0643\u062a\u0631\u0648\u0646\u064a"

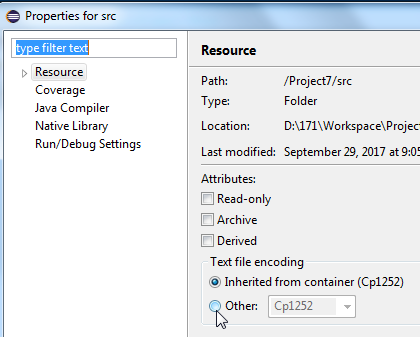
Create an Eclipse or NetBeans Java project whose Java source file is encoded in UTF-8, and then use that file to display the three sentences in the console window.

Note: To be able to display Arabic text, your Java file must be encoded in UTF-8. One way of doing that in Eclipse is to use the following steps:

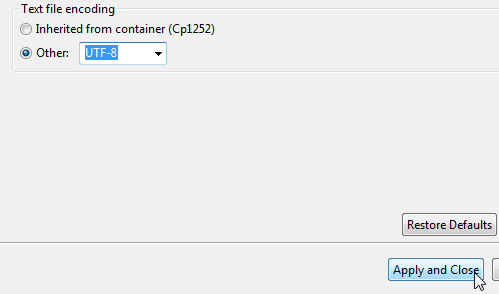
1. Create a new Java project.
2. Right-click on the project’s source folder in the Package Explorer:



1. In the pop-up menu that is displayed, click **Properties**, and then change Text file encoding from Cp1252 to UTF-8



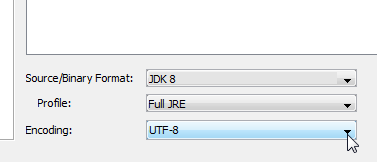
1. Click the **Apply and Close** button:



1. Create a class and paste or write the Arabic text as a string in an output statement like System.out.println
2. Execute your program.

Changing the source-file character encoding in NetBeans

In the Projects explorer window, right-click on the projects icon, select Properties 🡪 Sources 🡪 Encoding



**Task2:** Write an interactive Java program that prompts for and reads a 4-letter word string, it then displays the integer Unicode code for each character in the string.

Sample program runs:

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Task3:** Write an interactive Java program that prompts for and reads a string that contains three or more words. The program then displays the string without the first and the last words.

Sample program runs:

|  |  |
| --- | --- |
|  |  |

**Task4:** Write an interactive Java program that prompts for and reads a string containing the first name and the last name of a person, it then prints the last name followed by a comma and the first initial followed by a full stop. Both the last name and the initial must be in uppercase.

Sample program runs:

|  |  |
| --- | --- |
|  |  |

**Task5:** Write an interactive Java program that prompts for and reads a string, it then prompts for and reads two integer indexes. It finally swaps two characters at the two indexes. Your program must work for any pair of valid indexes.

Sample program runs:

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Task6:** Normally a KFUPM course is described by a string in the format:

"XXX ddd: Course-Title" where **XXX** is a two-, three-, or four-character Course code and **ddd** is Course number

Write an interactive Java program that prompts for and reads a course description of the form: **"PYP 003: University Study Skills".** It thensplits it into the three strings: **"PYP"**, **"003"**, and **"University Study Skills"**

it then displays the strings in the format:

**Course Code: PYP**

**Course Number: 003**

**Course Title: University Study Skills**

**Note:** Your program must be general; it should work for any course description following the above format.

Sample program runs:

|  |  |
| --- | --- |
|  |  |