



SRI AKILANDESWARI WOMEN'S COLLEGE, WANDIWASH

Wrapper Classes & Strings

Class : II BCA

Mrs.R.SAIKUMARI

Assistant Professor

Department of Computer Applications

SWAMY ABEDHANADHA EDUCATIONAL TRUST, WANDIWASH

Chapter Topics



topics

We will cover the following topics in this order:

- 7.3.2 Wrapper Classes
- 2.5.6 Type Conversion (Parse Methods)
- 2.3.3 Operations on Strings
- 5.3.1 Some Built-in Classes (StringBuilder)

Wrapper Classes

- Java provides 8 primitive data types: byte, short, int, long, float, double, boolean, char
- One of the limitations of the primitive data types is that we cannot create ArrayLists of primitive data types.
- However, this limitation turns out not to be very limiting after all, because of the so-called wrapper classes:
 - Byte, Short, **Integer**, Long, Float, Double, Boolean, **Character**
- These wrapper classes are part of java.lang (just like String and Math) and there is no need to import them.

Wrapper Classes Examples

- Creating a new object:

```
Integer studentCount = new Integer(12);
```

- Changing the value stored in the object:

```
studentCount = new Integer(20);
```

- Getting a primitive data type from the object:

```
int count = studentCount.intValue();
```

Auto Boxing / UnBoxing

- You can also assign a primitive value to a wrapper class object directly without creating an object. This is called **Autoboxing**

Integer studentCount = 12;

- You can get the primitive value out of the wrapper class object directly without calling a method (as we did when we called `.intValue()`). This is called **Unboxing**

System.out.println(studentCount);

Wrapper Classes and ArrayList Example

```
public static void main(String[] args) {  
    ArrayList<Integer> list = new ArrayList<Integer>();  
    Scanner k = new Scanner(System.in);  
    System.out.println("Enter some non-zero integers. Enter 0 to end.");  
    int number = k.nextInt();  
    while (number != 0)  
    {  
        list.add(number); // autoboxing happening here  
        number = k.nextInt();  
    }  
  
    System.out.println("Your numbers in reverse are:");  
    for (int i = list.size() - 1; i >= 0; i--) {  
        System.out.println(list.get(i)); // unboxing happening here  
    }  
}
```

The Parse Methods

- One of the useful methods on the Wrapper classes is the **parse** methods. These are static methods that allow you to convert a String to a number.
- Each class has a different name for its parse method:
 - The **Integer** class has a **parseInt** method that converts a String to an **int**
 - The **Short** class has a **parseShort** method that converts a String to a **Short**
 - The **Float** class has a **parseFloat** method that converts a String to a **Float**
 - Etc.

The Parse Methods Examples

```
byte b = Byte.parseByte("8");  
short sVar = Short.parseShort("17");  
int num = Integer.parseInt("28");  
long longVal = Long.parseLong("149");  
float f = Float.parseFloat("3.14");  
double price = Double.parseDouble("18.99");
```

- If the String cannot be converted to a number, an exception is thrown. We will discuss exceptions later.

Helpful Methods on Wrapper Classes

- The **toString** is static method that can convert a number back to a String:

```
int months = 12;
double PI = 3.14;
String monthsStr = Integer.toString(months);
String PIStr = Double.toString(PI);
```

- The Integer and Long classes have three additional methods to do base conversions: `toBinaryString`, `toHexString`, and `toOctalString`

```
int number = 16;
System.out.print(Integer.toBinaryString(number) + " " +
Integer.toHexString(number) + " " + Integer.toOctalString(number));
```

– output: 10000 10 20

Helpful Static Variables on Wrapper Classes

- The numeric wrapper classes each have a set of static final variables to know the range of allowable values for the data type:
 - `MIN_VALUE`
 - `MAX_VALUE`

```
System.out.println("The minimum val for an int is " +  
    Integer.MIN_VALUE);
```

```
System.out.println("The maximum val for an int is " +  
    Integer.MAX_VALUE);
```

Character Testing and Conversion With The Character Class

- The `Character` class allows a `char` data type to be *wrapped* in an object.

`Character x = new Character('K');`

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

The Character Class – Static Methods

Method	Description
<code>boolean isDigit(char ch)</code>	Returns true if the argument passed into <i>ch</i> is a digit from 0 through 9. Otherwise returns false.
<code>boolean isLetter(char ch)</code>	Returns true if the argument passed into <i>ch</i> is an alphabetic letter. Otherwise returns false.
<code>boolean isLetterOrDigit(char ch)</code>	Returns true if the character passed into <i>ch</i> contains a digit (0 through 9) or an alphabetic letter. Otherwise returns false.
<code>boolean isLowerCase(char ch)</code>	Returns true if the argument passed into <i>ch</i> is a lowercase letter. Otherwise returns false.
<code>boolean isUpperCase(char ch)</code>	Returns true if the argument passed into <i>ch</i> is an uppercase letter. Otherwise returns false.
<code>boolean isSpaceChar(char ch)</code>	Returns true if the argument passed into <i>ch</i> is a space character. Otherwise returns false.

Character Testing and Conversion With The `Character` Class

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

Method	Description
<code>char toLowerCase(char ch)</code>	Returns the lowercase equivalent of the argument passed to <i>ch</i> .
<code>char toUpperCase(char ch)</code>	Returns the uppercase equivalent of the argument passed to <i>ch</i> .

Example

```
public static void main(String[] args)
{
    Scanner k = new Scanner(System.in);
    System.out.println("Enter a character please: ");
    char ch = k.nextLine().charAt(0);
    if (Character.isLetter(ch))
    {
        System.out.println("Found a letter!");
        if (Character.isLowerCase(ch))
            System.out.println("Found a lowercase letter!");

        if (Character.isUpperCase(ch))
            System.out.println("Found an uppercase letter!");
    }

    if (Character.isDigit(ch))
        System.out.println("Found a digit!");

    if (Character.isSpaceChar(ch))
        System.out.println("Found a single space!");

    if (Character.isWhitespace(ch))
        System.out.println("Found a whitespace! (could be tab or enter too);");
}
```

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 - startsWith

- 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 - endsWith

- 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 - regionMatches

- The `String` class provides methods that determine 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 - regionMatches

Location 6

Location 15

```
String str = "Four score and seven years ago";  
String str2 = "Those seven years passed quickly!";  
if (str.regionMatches(15, str2, 6, 11))  
    System.out.println("The regions match.");  
else  
    System.out.println("The regions do not match.");
```

11 characters
to be compared

```
String str = "Four score and seven years ago";  
String str2 = "THOSE SEVEN YEARS PASSED QUICKLY!";  
if (str.regionMatches(true, 15, str2, 6, 11))  
    System.out.println("The regions match.");  
else  
    System.out.println("The regions do not match.");
```

true:
means
ignore the
case when
comparing

Searching Strings – indexOf, lastIndexOf

- 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 – indexOf, lastIndexOf

```
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);
```

```
// This code will find ALL occurrences
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);
}
```

String Methods For Getting Character Or Substring Location

See Table 9-4

Method	Description
<code>int indexOf(char ch)</code>	Searches the calling <code>String</code> object for the character passed into <code>ch</code> . If the character is found, the position of its first occurrence is returned. Otherwise, <code>-1</code> is returned.
<code>int indexOf(char ch, int start)</code>	Searches the calling <code>String</code> object for the character passed into <code>ch</code> , beginning at the position passed into <code>start</code> and going to the end of the string. If the character is found, the position of its first occurrence is returned. Otherwise, <code>-1</code> is returned.
<code>int indexOf(String str)</code>	Searches the calling <code>String</code> object for the string passed into <code>str</code> . If the string is found, the beginning position of its first occurrence is returned. Otherwise, <code>-1</code> is returned.
<code>int indexOf(String str, int start)</code>	Searches the calling <code>String</code> object for the string passed into <code>str</code> . The search begins at the position passed into <code>start</code> and goes to the end of the string. If the string is found, the beginning position of its first occurrence is returned. Otherwise, <code>-1</code> is returned.

String Methods For Getting Character Or Substring Location

See Table 9-4

```
int lastIndexOf(char ch)
```

Searches the calling `String` object for the character passed into `ch`. If the character is found, the position of its last occurrence is returned. Otherwise, `-1` is returned.

```
int lastIndexOf(char ch, int start)
```

Searches the calling `String` object for the character passed into `ch`, beginning at the position passed into `start`. The search is conducted backward through the string, to position `0`. If the character is found, the position of its last occurrence is returned. Otherwise, `-1` is returned.

```
int lastIndexOf(String str)
```

Searches the calling `String` object for the string passed into `str`. If the string is found, the beginning position of its last occurrence is returned. Otherwise, `-1` is returned.

```
int lastIndexOf(String str,  
                int start)
```

Searches the calling `String` object for the string passed into `str`, beginning at the position passed into `start`. The search is conducted backward through the string, to position `0`. If the string is found, the beginning position of its last occurrence is returned. Otherwise, `-1` is returned.

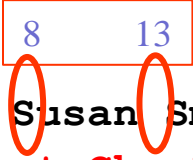
Extracting Substrings

- 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);  
String firstName = fullName.substring(0, 7);  
System.out.println("The full name is " + fullName);  
System.out.println("The last name is " + lastName);
```


Extracting Characters to Arrays

- The `String` class provides methods to extract substrings in a `String` object and store them in `char` arrays.
 - **`getChars`**(int srcBegin, int srcEnd, char[] dst, int dstBegin)
 - Stores a substring in a `char` array
 - `srcBegin`: first index to start copying from in the `src` string `getChars` is called on (e.g. `src.getChars(...)`)
 - `srcEnd`: index after the last character in the string to copy
 - `dst`: the destination array to copy to. Must be created already
 - `dstBegin`: the start offset in the destination array to start copying
 - **`toCharArray`** ()
 - Returns the `String` object's contents in an array of `char` values.



```
String fullName = "Cynthia Susan Smith";
char[] nameArray = fullName.toCharArray();
char[] middleName;
fullName.getChars(8, 13, middleName, 0);
char[] chars = fullName.toCharArray();
```

Returning Modified Strings

STRING IS IMMUTABLE

- The `String` class provides methods that **return** modified `String` objects.
 - `concat(String str)`
 - Returns a `String` object that is the concatenation of two `String` objects; the original and the `str` given as input.

```
String s1 = "Hello";  
s1 = s1.concat(" there");
```

- `replace(char oldChar, char newChar)`
 - Returns a `String` object with all occurrences of one character being replaced by another character.

```
s1 = s1.replace('l', 'L');
```

- `trim()`
 - Returns a `String` object with all leading and trailing whitespace characters removed.

```
s1 = s1.trim();
```

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.

`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));
System.out.println(String.valueOf(letters));
System.out.println(String.valueOf(letters, 1, 3));
System.out.println(String.valueOf(d));
System.out.println(String.valueOf(i));
```

- Produces the following output:

```
true
abcde
bcd
2.4981567
7
```

CW Part-1-1: Wrapper Classes, Strings and Characters

Write a program that:

Part A: asks the user for a series of floats until the user enters -1. The program should store the numbers in an ArrayList of Floats then calls the Collections.sort method to sort the ArrayList and print the contents back on separate lines.

Part B: asks the user for a String. The program reads in the String and displays the following statistics:

- Number of upper case letters
- Number of digits
- Number of white spaces
- The location/index of all occurrences of the letter 'e'. If there are no e's, it should print "String has no e's". Use an ArrayList to collect the location of all the e's.

Compile and test your code in NetBeans and then on Hackerrank at

<https://www.hackerrank.com/csc128-part-1-classwork>

then choose CSC128-Classwork-1-1

Submit your .java file and a screenshot of passing all test cases on Hackerrank.

The StringBuilder Class

- The String class is immutable – changes cannot be made to an existing String.
- The StringBuilder class is a class similar to the String class, but it is mutable – changes can be made.
- There are three ways to construct a StringBuilder:
 - `StringBuilder()`: create an empty StringBuilder of length 16
 - `StringBuilder(int length)`: create an empty StringBuilder with the specified length
 - `StringBuilder(String str)`: create a StringBuilder with the string's contents.

Common Methods between String and StringBuilder

- The String and StringBuilder have some methods in common:

```
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`**

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

- These methods accept two arguments:
 - *start*: an `int` that specifies the position to begin insertion, and
 - *item*: 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.

Replacing a Substring in a `StringBuilder` Object

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

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

- *start*: an `int` that specifies the starting position of a substring in the calling object
 - *end*: an `int` that specifies the ending position of the substring. (The starting position is included in the substring, but the ending position is not.)
 - *str*: `String` object to replace in the original string
- 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); // starting at 9 and ending at 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`

```
StringBuilder strb = new StringBuilder("This is a test.");  
String str = strb.toString();
```

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.

```
// 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);
```

- This code will produce the following output:

```
one  
two  
three  
four
```

CW Part-1-2: String Tokenizer

- Ask the user for the prices of items bought for lunch. The prices should be entered separated by commas using a single String. (e.g. \$6.99, \$1.09, \$1.99)
- Read in the String, remove the \$ signs and use the split method to get a String[] of the prices entered.
- Trim the white spaces around the strings and convert the Strings to **floats** using the parse methods and add them all up.
- Display the total price to be paid.

Compile and test your code in NetBeans and then on Hackerrank at

<https://www.hackerrank.com/csc128-part-1-classwork>

then choose CSC128-Classwork-1-2

Submit your .java file and a screenshot of passing all test cases on Hackerrank.

Programming Assignment (50 Points)

Write a class with the following static methods:

- **WordCounter**: This method takes a String and returns the number of words in the String.
- **convertToString**: This method takes an ArrayList of Characters and returns a String representation of the characters.
- **mostFound**: This method takes a String and returns the character that appears the most in the String. Ignore the case when counting.
- **replacePart**: This method takes 3 Strings *original*, *toReplace*, *replaceWith*. It finds all occurrences of *toReplace* in the *original* String and returns the *original* String with *toReplace* replaced with *replaceWith*. For example, if the *original* String was “I have two dogs and two cats”, *toReplace* is “two” and *replaceWith* is “three”, the method returns the String “I have three dogs and three cats”.

Write a main method that:

- Asks the user for a String, a toReplace String and a replaceWith String. It prints:
 - Number of words in the String
 - The character that appears the most in the String
 - The new String after calling replacePart
- Asks the user for a series of characters and creates an ArrayList of these characters. The user should press . when done entering characters. It then prints out the Characters as a String (using convertToString) and prints the String in all upper case.

Compile and test your code in NetBeans and then on Hackerrank at

<https://www.hackerrank.com/contests/csc128-programmingassignments> then choose CSC128-Part-1-PA

Submit your .java file and a screenshot of passing all test cases on Hackerrank.

Acknowledgment

Attribution-NonCommercial-
ShareAlike 4.0 International



"Java II – Part 1 – Wrapper Classes and Strings" by Ibtam Mahfouz, [Manchester Community College](#) is licensed under [CC BY-NC-SA 4.0](#) / A derivative from the [original work](#)