Programming in Java

String and StringBuilder classes

Introduction

- A string is a sequence of characters.
- In many languages, strings are treated as an array of characters, but in Java a string is an object.
- Every string we create is actually an object of type String.
- String constants are actually String objects.
- Example:

```
System.out.println("This is a String, too");
```

- Objects of type String are immutable i.e. once a String object is created, its contents cannot be altered.

Introduction

- In java, four predefined classes are provided that either represent strings or provide functionality to manipulate them. Those classes are:
  - String
  - StringBuffer
  - StringBuilder
  - StringTokenizer

- String, StringBuffer, and StringBuilder classes are defined in java.lang package and all are final.

- All three implement the CharSequence interface.

Why String Handling?

String handling is required to perform following operations on some string:

- compare two strings
- search for a substring
- concatenate two strings
- change the case of letters within a string

Creating String objects

```
class StringDemo
        {
          public static void main(String args[])
                {
                  String strOb1 = “Students";
                  String strOb2 = “LPU";
                  String strOb3 = strOb1 + " and " + strOb2;
                  System.out.println(strOb1);
                  System.out.println(strOb2);
                  System.out.println(strOb3);
                }
        }
```

String Class

String Constructor:

public String ()
public String (String strObj)
public String (char chars[])
public String (byte asciiChars [])
public String (char chars[ ], int startIndex, int numChars)
public String (byte asciiChars[ ], int startIndex, int numChars)

Examples

```
char [] a = {'c', 'o', 'n', 'g', 'r', 'a', 't', 's'};
byte [] b = {65, 66, 67, 68, 69, 70, 71, 72};
String s1 = new String (a); System.out.println(s1);
String s2 = new String (a, 1,5); System.out.println(s2);
String s3 = new String (s1); System.out.println(s3);
String s4 = new String (b); System.out.println(s4);
String s5 = new String (b, 4, 4); System.out.println(s5);
```

```
congrats
ongra
congrats
ABCDEFGH
EFGH
```

String Concatenation

- Concatenating Strings:

```
String age = "9";
String s = "He is " + age + " years old.";
System.out.println(s);
```

- Using concatenation to prevent long lines:

```
String longStr = “This could have been” +
                  “a very long line that would have” +
                  “wrapped around.  But string”+
                 “concatenation  prevents this.”;
System.out.println(longStr);
```

String Concatenation with Other Data Types

- We can concatenate strings with other types of data.

Example:

```
int age = 9;
String s = "He is " + age + " years old.";
System.out.println(s);
```

Methods of String class

- String Length:

  length() returns the length of the string i.e. number of characters.

int length()

Example:

```
char chars[] = { 'a', 'b', 'c' };
String s = new String(chars);
System.out.println(s.length());
```

Character Extraction

- charAt(): used to obtain the character from the specified index from a string.

public char charAt (int index);

Example:

```
char ch;
ch = "abc".charAt(1);
```

Methods Cont…

- getChars(): used to obtain set of characters from the string.

void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)

Example:
```
String s = “KAMAL”;
char b[] = new char [10];
b[0] = ‘N’;    b[1] = ‘E’;
b[2] = ‘E’;    b[3] = ‘L’;
s.getChars(0, 4, b, 4);
System.out.println(b);
```

Output: NEELKAMA

Methods Cont…

- toCharArray(): returns a character array initialized by the contents of the string.

char [] to Char Array();

Example:

```
String s = “INDIA”;
char c[] = s.toCharArray();
    for (int i=0; i<c.length; i++)
        {
          if (c[i]>= 65 && c[i]<=90)
              c[i] += 32;
              System.out.println(c);
        }
```

String Comparison

- equals(): used to compare two strings for equality.
  Comparison is case-sensitive.

  public boolean equals (Object str)

- equalsIgnoreCase( ): To perform a comparison that ignores case differences.

Note:

- This method is defined in Object class and overridden in String class.
- equals(), in Object class, compares the value of reference not the content.
- In String class, equals method is overridden for content-wise comparison of two strings.

Example

```
class equalsDemo {
        public static void main(String args[]) {
                String s1 = "Hello";
                String s2 = "Hello";
                String s3 = "Good-bye";
                String s4 = "HELLO";
                System.out.println(s1 + " equals " + s2 + " -> " +
                s1.equals(s2));
                System.out.println(s1 + " equals " + s3 + " -> " +
                s1.equals(s3));
                System.out.println(s1 + " equals " + s4 + " -> " +
                s1.equals(s4));
                System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> “
                                        +s1.equalsIgnoreCase(s4));
                }
        }
```

String Comparison

- startsWith( ) and endsWith( ):
  - The startsWith( ) method determines whether a given String begins with a specified string.
  - Conversely, endsWith( ) determines whether the String in question ends with a specified string.

boolean startsWith(String str)

boolean endsWith(String str)

String Comparison

compareTo( ):

- A string is less than another if it comes before the other in dictionary order.
- A string is greater than another if it comes after the other in dictionary order.

int compareTo(String str)

Value	Meaning
Less than zero	The invoking string is less than *str*.
Greater than zero	The invoking string is greater than *str*.
Zero	The two strings are equal.

Example

```
class SortString {
    static String arr[] = {"Now", "is", "the", "time", "for", "all", "good", "men",
                           "to", "come", "to", "the", "aid", "of", "their", "country"};
    public static void main(String args[]) {
        for(int j = 0; j < arr.length; j++) {
            for(int i = j + 1; i < arr.length; i++) {
                if(arr[j].compareTo(arr[i]) > 0) {
                    String t = arr[j];
                    arr[j] = arr[i];
                    arr[i] = t;
                }
            }
          System.out.println(arr[j]);
        }
    }
}
```

String Comparison

boolean regionMatches(int toffset, String other, int ooffset, int len)

- Tests if two string regions are equal

boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len) :

- Tests if two string regions are equal, with case considerations

Example:

```
System.out.println("Hello".regionMatches(false,0,"hello",0,3));
System.out.println("Hello".regionMatches(true,0,"hello",0,3));
```

Output

false

true

Searching Strings

- The String class provides two methods that allow us to search a string for a specified character or substring:

  indexOf( ): Searches for the first occurrence of a character or substring.

  int indexOf(int ch)

  lastIndexOf( ): Searches for the last occurrence of a character or substring.

  int lastIndexOf(int ch)

- To search for the first or last occurrence of a substring, use

  int indexOf(String str)

  int lastIndexOf(String str)

- We can specify a starting point for the search using these forms:

  int indexOf(int ch, int startIndex)
  int lastIndexOf(int ch, int startIndex)
  int indexOf(String str, int startIndex)
  int lastIndexOf(String str, int startIndex)

- Here, startIndex specifies the index at which point the search begins.

- For indexOf( ), the search runs from startIndex to the end of the string.

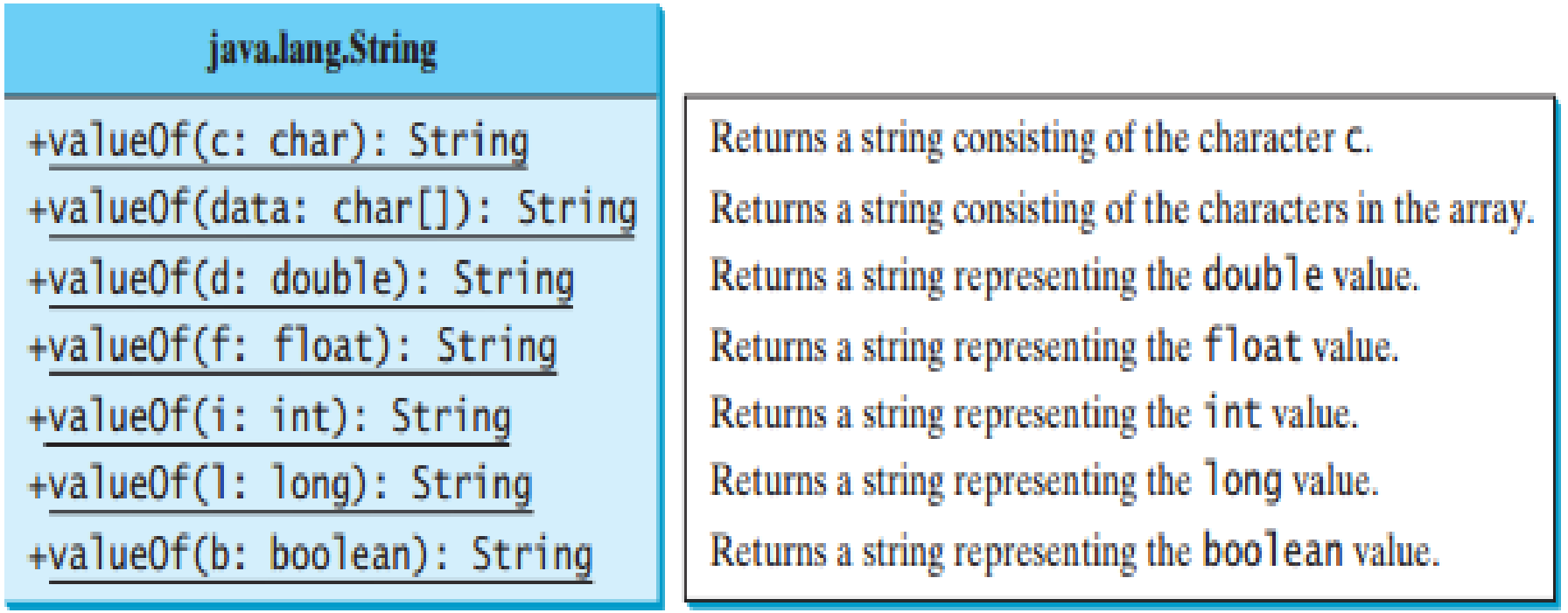
- For lastIndexOf( ), the search runs from startIndex to zero.

Example

```
class indexOfDemo {
    public static void main(String args[]) {
        String s = "Now is the time for all good men " +
                   "to come to the aid of their country.";
        System.out.println(s);
        System.out.println("indexOf(t) = " + s.indexOf('t'));
        System.out.println("lastIndexOf(t) = " + s.lastIndexOf('t'));
        System.out.println("indexOf(the) = " + s.indexOf("the"));
        System.out.println("lastIndexOf(the) = " + s.lastIndexOf("the"));
        System.out.println("indexOf(t, 10) = " + s.indexOf('t', 10));
        System.out.println("lastIndexOf(t, 60) = " + s.lastIndexOf('t', 60));
        System.out.println("indexOf(the, 10) = " + s.indexOf("the", 10));
        System.out.println("lastIndexOf(the, 60) = " + s.lastIndexOf("the", 60));
        }
    }
```

- Converting Characters and Numeric Values to Strings

- The static valueOf method can be used to convert an array of characters into a string.



Modifying a String

- Because String objects are immutable, whenever we want to modify a String, we must either copy it into a StringBuffer or StringBuilder, or use one of the following String methods, which will construct a new copy of the string with modifications.

- substring(): used to extract a part of a string.

  public String substring (int start_index)

  public String substring (int start_index, int end_index)

Example:

```
String s = "ABCDEFG";
String t = s.substring(2);      System.out.println (t);
String u = s.substring (1, 4); System.out.println (u);
```

CDEFG

BCD

concat( ): used to concatenate two strings.

String concat(String str)

- This method creates a new object that contains the invoking string with the contents of str appended to the end.

- concat( ) performs the same function as +.

Example:

String s1 = "one"; String s2 = s1.concat("two");

- It generates the same result as the following sequence:

String s1 = "one"; String s2 = s1 + "two";

replace( ): The replace( ) method has two forms.

- The first replaces all occurrences of one character in the invoking string with another character. It has the following general form:

String replace(char original, char replacement)

- Here, original specifies the character to be replaced by the character specified by replacement.

Example: String s = "Hello".replace('l', 'w');

- The second form of replace( ) replaces one character sequence with another. It has this general form:

String replace(CharSequence original, CharSequence replacement)

trim( )

- The trim( ) method returns a copy of the invoking string from which any leading and trailing whitespace has been removed.

String trim( )

Example:

```
String s = "   Hello World    ".trim();
```

This puts the string “Hello World” into s.

Changing the Case of Characters Within a String

toLowerCase() & toUpperCase()

- Both methods return a String object that contains the uppercase or lowercase equivalent of the invoking String.

String toLowerCase( )

String toUpperCase( )

Java String join

- The java string join() method returns a string joined with given delimiter.
- In string join method, delimiter is copied for each elements.
- In case of null element, "null" is added.
- The join() method is included in java string since JDK 1.8.

public static String join(CharSequence delimiter, CharSequence... elements)

Java String join

```
class StringJoinDemo
{ public static void main(String args[])
{       String result = String.join(" ", "Alpha", "Beta", "Gamma");
        System.out.println(result);
        result = String.join(", ", "John", "ID#: 569","E-mail:
                                        John@HerbSchildt.com");
        System.out.println(result);
}}
```

The output is:

Alpha Beta Gamma

John, ID#: 569, E-mail: John@HerbSchildt.com

Java String intern

- The java string intern() method returns the interned string.
- It returns the canonical representation of string.
- It can be used to return string from pool memory, if it is created by new keyword.

public String intern()

Java String intern

```
public class InternExample
{ public static void main(String args[])
  {String s1=new String("hello");
   String s2="hello";
   String s3=s1.intern();
  System.out.println(s1==s2);
  System.out.println(s2==s3);
   }}
```

- Output:

  false

  true

StringBuilder

- A mutable sequence of characters.
- StringBuilder is non-synchronized.
- The principal operations on a StringBuilder are the append and insert methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string builder.
- The append method always adds these characters at the end of the builder; the insert method adds the characters at a specified point.

Constructors

- StringBuilder()

  Constructs a string builder with no characters in it and an initial capacity of 16 characters.

- StringBuilder(CharSequence seq)
- StringBuilder(int capacity)
- StringBuilder(String str)

Methods

- public StringBuilder append(String s)

  The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.

- public StringBuilder insert(int offset, String s)
- public StringBuilder replace(int startIndex, int endIndex, String str)
- public StringBuilder delete(int startIndex, int endIndex)
- public StringBuilder reverse()
- public int capacity()

Methods

- public void ensureCapacity(int minimumCapacity)
- public char charAt(int index)
- public int length()
- public String substring(int beginIndex)
- public String substring(int beginIndex, int endIndex)