String Handling
Lesson - 6

Objectives

- Review of last class
- Strings.
- String Operations.
- StringBuffer.
- StringBuffer Operations.

Strings

- Java string is a sequence of characters. They are objects of type String.
- Once a String object is created it cannot be changed. Stings are Immutable.
- To get changeable strings use the class called StringBuffer.
- String and StringBuffer classes are declared final, so there cannot be subclasses of these classes.
- The default constructor creates an empty string.

```
String s = new String();
```

Creating Strings

- `String str = "abc";` is equivalent to:

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

- If data array in the above example is modified after the string object str is created, then str remains unchanged.
- Construct a string object by passing another string object.

  String str2 = new String(str);

String Operations

- The length() method returns the length of the string.

  Eg: System.out.println(“Hello”.length()); // prints 5
- The + operator is used to concatenate two or more strings.

  Eg: String myname = “Harry”

  String str = “My name is” + myname+ “.”;
- For string concatenation the Java compiler converts an operand to a String whenever the other operand of the + is a String object.

String Operations

- Characters in a string can be extracted in a number of ways.

public char **charAt**(int index)

  - Returns the character at the specified index. An index ranges from 0 to length() - 1. The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

```
char ch;
ch = “abc”.charAt(1); // ch = “b”
```

String Operations

- **getChars() -** Copies characters from this string into the destination character array.

```
public void getChars(int srcBegin, int srcEnd,
char[] dst, int dstBegin)
```

  - srcBegin - index of the first character in the string to copy.
  - srcEnd - index after the last character in the string to copy.
  - dst - the destination array.
  - dstBegin - the start offset in the destination array.

String Operations

- **equals() -** Compares the invoking string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as the invoking object.

  ```
  public boolean equals(Object anObject)
  ```

- **equalsIgnoreCase()-** Compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

  ```
  public boolean equalsIgnoreCase(String anotherString)
  ```

String Operations

- **startsWith()** – Tests if this string starts with the specified prefix.

```
public boolean startsWith(String prefix)
"Figure".startsWith("Fig"); // true
```

- **endsWith() -** Tests if this string ends with the specified suffix.

```
public boolean endsWith(String suffix)
"Figure".endsWith("re"); // true
```

String Operations

- **startsWith()** -Tests if this string starts with the specified prefix beginning at a specified index.

```
public boolean startsWith(String prefix,
int toffset)

    prefix - the prefix.
    toffset - where to begin looking in the
    string.

"figure".startsWith("gure", 2); // true
```

String Operations

- **compareTo() -** Compares two strings lexicographically.
  - The result is a negative integer if this String object lexicographically precedes the argument string.
  - The result is a positive integer if this String object lexicographically follows the argument string.
  - The result is zero if the strings are equal.
  - compareTo returns 0 exactly when the equals(Object) method would return true.

```
public int compareTo(String anotherString)
public int compareToIgnoreCase(String str)
```

String Operations

indexOf – Searches for the first occurrence of a character or substring. Returns -1 if the character does not occur.

`public int` **`indexOf`**`(int ch)` – Returns the index within this string of the first occurrence of the specified character.

`public int` **`indexOf`**`(String str)` - Returns the index within this string of the first occurrence of the specified substring.

```
String str = "How was your day today?";
str.indexof('t');
str("was");
```

String Operations

`public int` **`indexOf`**`(int ch, int fromIndex)` – Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

`public int` **`indexOf`**`(String str, int fromIndex)` - Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

String str = “How was your day today?”;
str.indexof(‘a’, 6);
str(“was”, 2);

String Operations

lastIndexOf() –Searches for the last occurrence of a character or substring. The methods are similar to indexOf().

substring() - Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

```
public String substring(int beginIndex)
```

Eg: `"unhappy".substring(2) returns "happy"`

String Operations

- `public String` **`substring`**`(int beginIndex, int endIndex)`

`Eg: "smiles".substring(1, 5)`
`returns "mile“`

String Operations

concat() - Concatenates the specified string to the end of this string.

If the length of the argument string is 0, then this String object is returned.

Otherwise, a new String object is created, containing the invoking string with the contents of the str appended to it.

```
public String concat(String str)
"to".concat("get").concat("her") returns
  "together"
```

String Operations

- replace()- Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

```
public String replace(char oldChar, char newChar)
```

```
"mesquite in your cellar".replace('e', 'o')
   returns "mosquito in your collar"
```

String Operations

- **trim() -** Returns a copy of the string, with leading and trailing whitespace omitted.

```
public String trim()

String s = "  Hi Mom!  ".trim();
S = "Hi Mom!"
```

- **valueOf() –** Returns the string representation of the char array argument.

```
public static String valueOf(char[] data)
```

String Operations

- The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.

Other forms are:

```
public static String valueOf(char c)
public static String valueOf(boolean b)
public static String valueOf(int i)
public static String valueOf(long l)
public static String valueOf(float f)
public static String valueOf(double d)
```

String Operations

- **toLowerCase():** Converts all of the characters in a String to lower case.
- **toUpperCase():** Converts all of the characters in this String to upper case.

```
public String toLowerCase()
public String toUpperCase()

Eg: "HELLO THERE".toLowerCase();
      "hello there".toUpperCase();
```

StringBuffer

- A StringBuffer is like a String, but can be modified.
- The length and content of the StringBuffer sequence can be changed through certain method calls.
- StringBuffer defines three constructors:
  - StringBuffer()
  - StringBuffer(int size)
  - StringBuffer(String str)

StringBuffer Operations

- The principal operations on a StringBuffer are the append and insert methods, which are overloaded so as to accept data of any type.

Here are few append methods:

```
StringBuffer append(String str)
StringBuffer append(int num)
```

- The append method always adds these characters at the end of the buffer.

StringBuffer Operations

- The insert method adds the characters at a specified point.

Here are few insert methods:

```
StringBuffer insert(int index, String str)
StringBuffer append(int index, char ch)
```

Index specifies at which point the string will be inserted into the invoking StringBuffer object.

StringBuffer Operations

- **delete() -** Removes the characters in a substring of this StringBuffer. The substring begins at the specified start and extends to the character at index end - 1 or to the end of the StringBuffer if no such character exists. If start is equal to end, no changes are made.

```
public StringBuffer delete(int start, int end)
```

StringBuffer Operations

- **replace() -** Replaces the characters in a substring of this StringBuffer with characters in the specified String.

```
public StringBuffer replace(int start, int end,
   String str)
```

- **substring() -** Returns a new String that contains a subsequence of characters currently contained in this StringBuffer. The substring begins at the specified index and extends to the end of the StringBuffer.

```
public String substring(int start)
```

StringBuffer Operations

- **reverse() -** The character sequence contained in this string buffer is replaced by the reverse of the sequence.

```
public StringBuffer reverse()
```

- **length() -** Returns the length of this string buffer.

```
public int length()
```

StringBuffer Operations

- **capacity() -** Returns the current capacity of the String buffer. The capacity is the amount of storage available for newly inserted characters.

```
public int capacity()
```

- **charAt() -** The specified character of the sequence currently represented by the string buffer, as indicated by the index argument, is returned.

```
public char charAt(int index)
```

StringBuffer Operations

- **getChars() -** Characters are copied from this string buffer into the destination character array dst. The first character to be copied is at index srcBegin; the last character to be copied is at index srcEnd-1.

```
public void getChars(int srcBegin, int srcEnd,
char[] dst, int dstBegin)
```

- **setLength**() **-** Sets the length of the StringBuffer.

```
public void setLength(int newLength)
```

Examples: StringBuffer

```
StringBuffer sb = new StringBuffer("Hello");
  sb.length(); // 5
  sb.capacity(); // 21 (16 characters room is
                    added if no size is specified)
  sb.charAt(1); // e
  sb.setCharAt(1,'i'); // Hillo
  sb.setLength(2); // Hi
  sb.append("l").append("l"); // Hill
  sb.insert(0, "Big "); // Big Hill
```

Examples: StringBuffer

```
sb.replace(3, 11, “”); // Big
sb.reverse(); // gib
```