String Handling

Strings

- Java string is a sequence of characters.
- In Java Strings are class objects .
- Once a String object is created it cannot be changed. Strings are Immutable.
- In Java, string is an instantiated object of the **String** class.
- 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 Arrays

- We can also create and use arrays that contain strings.

```
String a[]=new String[3];
String a[]=new String[n];
```

String Operations

- s1.length(): Gives length of s1

- 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.

S1.charAt(n) : Gives nth character of s1.

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

- **equals() – s1.equals(s2)** : Returns true if s1 is equal to s2

  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.

- **equalsIgnoreCase()- s1.equalsIgnoreCase(s2)**

  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.

String Operations

- **s1.startsWith(“substring”)** – Tests if this string starts with the specified prefix.

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

- **s1.endsWith(“substring”) -** Tests if this string ends with the specified suffix.

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

String Operations

- **s1.startsWith(“substring”,toffset)** -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

- **s1.compareTo(s2) -** Compares two strings lexicographically.
  - Returns negative if s1<s2
  - Returns positive if s1>s2
  - Returns zero if the strings are equal.

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

String Operations

s1.indexOf('x') – Gives the position of the first occurrence of 'x' in the string s1. 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');
```

String Operations

`s1.indexOf('x',n):` `Gives the position of 'x' that occurs after nth position in the string s1.`

`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. indexOf("was", 2);

String Operations

s1.substring(n): Gives substring starting from nth character.

s1.substring(n,m): Gives substring starting from nth character up to mth character(not including mth character)

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

s1.concat(s2) - Concatenates s1 and s2.

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

- s2= s1.replace(‘x’,’y’)
- Returns a new string resulting from replacing all occurrences of x in this string with y.

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

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

String Operations

- **s2=s1.trim() –** Remove white spaces at the beginning and end of string s1.
- Returns a copy of the string, with leading and trailing whitespace omitted.

```
public String trim()

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

String Operations

- **s2=s1.toLowerCase():** Converts all of the characters in a String s1 to lower case.
- **s2=s1.toUpperCase():** Converts all of the characters in this String s1 to upper case.

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

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

String Operations

- **p.toString():** Creates a string representation of the object p.

Program

Write a program to accept n number of strings from the user and sort them in alphabetical order.

StringBuffer

- StringBuffer is a peer class of **String**.
- StringBuffer creates strings of flexible length and content.
- We can insert characters and substrings in the middle of a string, or append another string to the end.
- The length and content of the StringBuffer sequence can be changed through certain method calls.
- StringBuffer defines three constructors:

  - **StringBuffer()** reserves room for 16 characters.
  - **StringBuffer(int size)** int argument explicitly specifies its size
  - **StringBuffer(String str)** it accepts string argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters.

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.
- **s1.setCharAt(n,'x')** modifies the nth character to x.
- **s1.append(s2)** appends the string s2 to s1 at the end
- **s1.insert(n,s2)** inserts the string s2 at the position n of s1
- **S1.setLength(n)** sets the length of the string s1 to n.If n<s1.length() s1 is truncated.If n>s1.length() zeros are added to s1.

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

- **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
```

Examples: StringBuffer

Write a program to accept string from the user and ask user whether he/she wants to modify that string, if yes accept the substring with the index value and then calculate and display number of vowels in it.