String Handling

String

String in Java is basically an object that represents sequence of char values. The **java.lang.String** class is used to create a string object. The **String**, **StringBuffer**, **StringBuilder** classes are defined in java.lang. These classes are final, that means none of classes can be subclassed. All three implement the **CharSequence** interface.

JAVA String is immutable!

When you create a String object, you are creating a string that cannot be changed. That is, once a string object has been created, you cannot change the characters that comprise the string. However, you can still perform all types of string operations. The difference is that each time you need an altered version of an existing string, a new string object is created that contains the modifications. The original string is left unchanged. This is because fixed, immutable strings can be implemented more efficiently than changeable ones. StringBuffer and StringBuilder classes hold strings that can be modified after they are created.

- A variable declared as a String reference can be changed to point at some other String object at any time.

Example:

```
public static void main(String[] args) {
    String s="ABC";
    s.concat(" DEF");
    System.out.println(s);
}
```

This will print only ABC

But if we explicitly assign it to the reference variable, it will refer to "ABC DEF" object.

```
public static void main(String[] args) {
    String s="ABC";
    s = s.concat(" DEF");
    System.out.println(s);
}
```

This will print ABC DEF

String Constructors

Default constructor:

String s = new String()

Parameterized constructor:

Char chars[]={'a', 'b', 'c'}

String s = new String(chars)

You can specify a subrange of a character array as an initializer using the following constructor:

String(char chars[], int startindex, int numChars)

Example:

```
public static void main(String[] args) {
    char chars[]={'a','b','c','d','e','f'};
     String s=new String(chars, 2,3);
     System.out.println(s);
}
```

Output: cde

String Length

The length of a string is the number of characters that it contains.

```
public static void main(String[] args) {
    char chars[]={'a','b','c','d','e','f'};
     String s=new String(chars);
     System.out.println(s.length());
     String p = new String("Welcome");
     System.out.println(p.length());
     System.out.println("abc".length());
}
```

Output: 6

7

3

String Operations

1. String Concatenation

```
public static void main(String[] args) {
   String longStr = "This could have been "+
            "a very long line that would have "+
            "wrapped aroung. But string concatenation "+
            "prevents this.";
   System.out.println(longStr);
```

Guess the output for:

i) String s = "four" + 2+2

ii) String s = "four" + (2+2)

2. String Conversion and toString()

It is the means by which you can determine the string representation for objects of classes that you create. Every class implements toString() because it is defined by object. By overriding toString() for classes that you create, you allow them to be fully integrated into Java's programming environment.

```
class Box
{
    double width, height, depth;
    Box(double w,double h,double d)
    {
        width=w;
        height=h;
        depth=d;
    }
    public String toString()
    {
        return "Dimensions are: "+width+" by "+depth+" by "+height+".";
    }
}
public class StringManipulation {

    public static void main(String[] args) {
      Box b = new Box(10,12,14);
      String s = "Box"+b;
      System.out.println(b);
      System.out.println(s);

    }

}
```

3. Character Extraction

1. To extract a single character from a string, you can refer directly to an individual character through the charAt() method. It has the general form:

char charAt(int where)

```
public static void main(String[] args) {
  char s = "abc".charAt(1);
  System.out.println(s);

}
```

Output: ?

2. getChars(): to extract more than one character at a time. General form:

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

output: demo

```
public static void main(String[] args) {
  String s = "This is a demo of the getChars method";
  int start = 10;
  int end = 14;
  char buf[]=new char[end-start];
  s.getChars(start, end, buf, 0);
  System.out.println(buf);

}
```

4. String Comparison

-> to compare two strings for equality, use equals. It has the general form:

boolean equals(Object str)

-> To perform a comparison that ignores case differences, call equalsIgnoreCase(). When it compares two strings, it considers A-Z to be the same as a-z. It has the general form:

boolean equalsIgnoreCase(String str)

```
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+" equals "+s4+" -> "+s1.equalsIgnoreCase(s4));

}
```

Output: true, false, false, true

#self-study (equals() versus ==)

5. Searching Strings:

- indexOf(): searches for the first occurrence of a character or substring
- lastIndexOf(): searches for the last occurrence of a character or substring

int indexOf(String str, int startindex)

int lastIndexOf(String str, int startindex)

```
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(t) = "+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));
}
```

N.B: for indexOf(), the search runs from startindex to the end of the string. For lastIndexOf(), the search runs from startindex to zero.

6. Modifying a string

1. substring:

You can extract a substring using **substring().** It has two forms. The first one is:

String substring(int startindex)

The second one is: String substring(int startindex, int endindex)

```
public static void main(String[] args) {
 String s = "Now is the time for all good men "+
       "to come to the aid of their country.";
 String search = "to";
 int i = s.indexOf(search);
 int j = s.indexOf("to");
 String result = s.substring(0,i);
 String out = s.substring(j);
 System.out.println(result);
 System.out.println(out);
}
```

2. concat:

String s1 = “one”;

String s2 = s1.concat(“two”);

3. replace:

String replace (char original, char replacement)

```
public static void main(String[] args) {
 String s = "Hello".replace('l', 'w');
 System.out.println(s);
}
```

4. trim:

The trim method returns a copy of the invoking string from which any leading and trailing whitespaces has been removed. It has the general form:

String trim()

```
public static void main(String[] args) {
 String s = "   Hello World   ".trim();
 System.out.println(s);
}
```

7. Changing the case of characters within a string

The method toLowerCase() converts all the characters in a string from uppercase to lowercase. The toUpperCase() method converts all the characters in a string from lowercase to uppercase.

```
public static void main(String[] args) {
String s = "This is a test";
System.out.println("Original: "+s);
String upper = s.toUpperCase();
String lower = s.toLowerCase();
System.out.println("Uppercase: "+upper);
System.out.println("LowerCase: "+lower);
}
```