JAVA STRING

Presented By: Pankaj Kumar

String

- object that represents sequence of char values.
- Present in a package called java.lang.String
- **For example:**

  char[] ch={‘C’,’S’,’E’};

  String s=**new** String(ch);

  Is same as:

  String s=“CSE”

String Constructors

String Constructors

1. String();
2. String(char *chars*[ ])
3. String(char *chars*[ ], int *startIndex*, int *numChars*)
4. String(byte *asciiChars*[ ])
5. String(byte *asciiChars*[ ], int *startIndex*, int *numChars*)

- **String** class supports several constructors.
- To create an empty **String**, we call the default constructor.
- For example,

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

will create an instance of **String** with no characters in it.

❖ To create a **String** initialized by an array of characters, use the constructor shown here:

String(char *chars*[ ])

❖ Here is an example:

```
char chars[] = { 'a', 'b', 'c' };

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

This constructor initializes **s** with the string “abc”.

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

  String(char *chars*[ ], int *startIndex*, int *numChars*)

  ✓ *startIndex* specifies the index at which the subrange begins

  ✓*numChars* specifies the number of characters to use.

Example:

```
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };

String s = new String(chars, 2, 3);
```

This initializes **s** with the characters **cde**.

- **String** class provides constructors that initialize a string when given a **byte** array.
- Their forms are shown here:
  - String(byte *asciiChars*[ ])
  - String(byte *asciiChars*[ ], int *startIndex*, int *numChars*)

Example

```
class SubStringCons {

public static void main(String args[]) {

byte ascii[] = {65, 66, 67, 68, 69, 70 };

String s1 = new String(ascii);

System.out.println(s1);

String s2 = new String(ascii, 2, 3);

System.out.println(s2);

} }
```

Output:

ABCDEF

CDE

How to create String object?

Two ways to create String object:

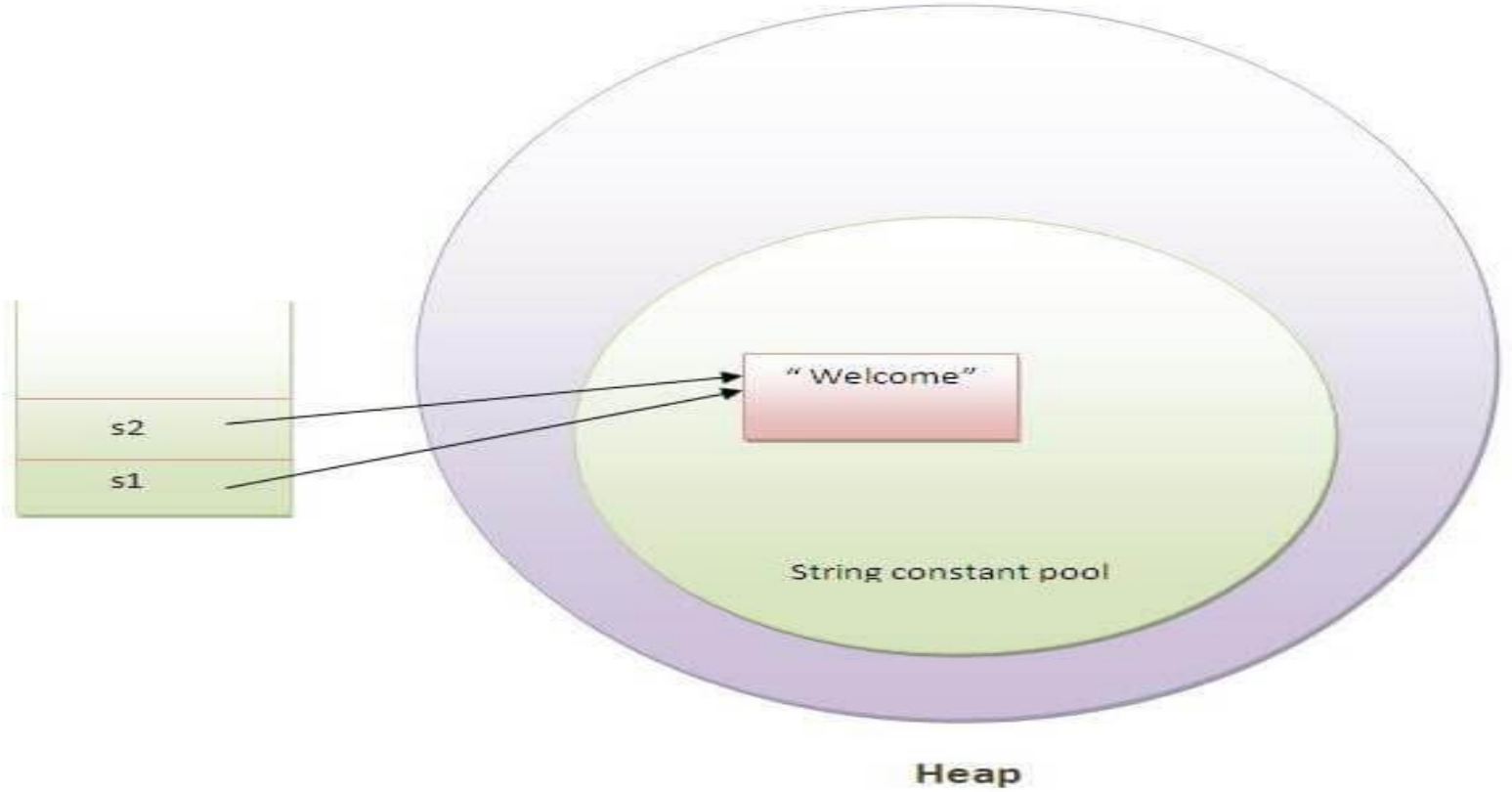
- By string literal
- By new keyword

String Literal

- Is created by using double quotes.
- **For Example:**

```
String s="welcome";
```

- String s1="Welcome";
- String s2="Welcome“;



By new keyword

```
public class StringExample
{
public static void main(String args[]) {
String s1="java“;
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);
String s3=new String("example");
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
} }
```

String Concatenation in Java

- String concatenation forms a new string that is the combination of multiple strings.
- Two ways to concat string in java:
    1. By + (string concatenation) operator
    2. By concat() method

By + (string concatenation) operator

Example:

```
String s=“ACIT"+“Engineering";

System.out.println(s);
```

Output:

ACIT Engineering

Guess the answer

```
String s=50+30+“CSE"+40+40;
System.out.println(s);
```

Output:

80CSE4040

Note: After a string literal, all the + will be treated as string concatenation operator.

String Concatenation by concat()

```
String s1=“Acharya ";

String s2=“ Institute";

String s3=s1.concat(s2);

System.out.println(s3);
```

Output:

Acharya Institute

Java toString() method

- If you want to represent any object as a string, **toString() method** comes into existence.
- toString() method returns the string representation of the object.

Understanding problem without toString()

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city)
{
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }

 public static void main(String args[]){
Student s1=new Student(101,"Raj","lucknow");
Student s2=new Student(102,"Vijay",“Bengaluru");
System.out.println(s1);//compiler writes here s1.toString()
System.out.println(s2);//compiler writes here s2.toString()
 }
}
```

Output:

Student@1fee6fc

Student@1eed786

Example of Java toString() method

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city)
{
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }
 public String toString()
{
 return rollno+" "+name+" "+city;
 }
 public static void main(String args[]){
 Student s1=new Student(101,"Raj","lucknow");
 Student s2=new Student(102,"Vijay",“Bengaluru'
 System.out.println(s1);
 System.out.println(s2);
 }
}
```

Output:

101 Raj lucknow

102 Vijay Bengaluru

Character Extraction

Presented By: Pankaj Kumar

Java String charAt()

- **String charAt()** method returns a char value at the given index number.
- It has this general form:

  char charAt(int *where*)

  Here, *where* is the index of the character that we want to obtain
- Index number starts from 0 and goes to n-1, where n is length of the string
- returns **StringIndexOutOfBoundsException** if given index number is greater than or equal to this string length or a negative number.

Example

```
public class CharAtExample    {

public static void main(String args[])

{

String name=“Acharya";

char ch=name.charAt(4);

System.out.println(ch);

}    }
```

Output:

r

getChars()

- Method copies the content of this string into specified char array.
- To extract more than one character at a time, we can use the **getChars( )** method.
- Syntax

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

Parameters description:

sourceStart – index of the first character in the string to copy.

sourceEnd – index after the last character in the string to copy.

target – Destination array of characters in which the characters from String gets copied.

targetStart – The index in Array starting from where the chars will be pushed into the Array.

Example

```
public class StringGetCharsExample  {

public static void main(String args[])   {

 String str = new String(“Hello Kumar how r u");

      char[] ch = new char[10];

      try{

         str.getChars(6, 16, ch, 0);

         System.out.println(ch);

      }catch(Exception ex)

{  System.out.println(ex);   }

}}
```

Output:

Kumar how

Example

```
String str=new String("Hello Engg how r u");

char[]ch=new char[10];

ch[0]='a';

ch[1]='b';

try{

str.getChars(6,10,ch,1);

System.out.println(ch);}

catch(Exception ex){   System.out.println(ex);   }
```

Output:

aEngg

Example:

```
String str=new String("Hello  Kumar how r u");

char[] ch=new char[10];

ch[0]='a';

ch[1]='b';

try

{

str.getChars(6,10,ch,4);

System.out.println(ch);

System.out.println(ch[4]);

}catch(Exception ex)

{  System.out.println(ex);    }
```

Output:

```
abKuma
K
```

String getBytes()

- **getBytes()** method returns the byte array of the string.
- its simplest form:

```
byte[ ] getBytes( )
```

Example

```
String s1="ABCDEFG";

byte[] barr=s1.getBytes();

for(int i=0;i<barr.length;i++)

 {

System.out.println(barr[i]);

 }
```

Output:

65

66

67

68

69

70

71

String toCharArray

- **toCharArray()** method converts this string into character array
- returns a newly created character array, its length is similar to this string
- **syntax**

```
public char[] toCharArray()
```

Example

```
String s1="hello";

char[] ch=s1.toCharArray();

for(int i=0;i<ch.length;i++)   {

System.out.println(ch[i]);    }
```

Output:

h
e
l
l
o

String Comparison

String comparison

There are three ways to compare string in java:

- By equals() method
- By = = operator
- By compareTo() method

1) String compare by equals()

- Compares **values** of string for equality.
- String class provides two methods:
  - *I. public boolean equals(String str)*
  - *II. public boolean equalsIgnoreCase(String str)*

Example for *equals*

```
String s1=“Kumar";
String s2=“Kumar";
String s3=new String(“Kumar");
String s4=“Atlas";
System.out.println(s1.equals(s2));
System.out.println(s1.equals(s3));
System.out.println(s1.equals(s4));
```

Output

true

true

false

Example for *equalsIgnoreCase*

```
String s1=“Kumar";

String s2=“KUMAR";

System.out.println(s1.equals(s2))

System.out.println(s1.equalsIgnoreCase(s2));
```

Output:

false

true

Example for *equals* and *equalsIgnoreCase*

```
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));
}
}
```

Output:

Hello equals Hello -> true
Hello equals Good-bye -> false
Hello equals HELLO -> false
Hello equalsIgnoreCase HELLO -> true

2) String compare by == operator

- The == operator compares **two object references** to see whether they refer to the same instance
- The = = operator compares **references not values.**

Example for compare by == operator

```
class Teststringcomparison3{

 public static void main(String args[]){

   String s1=“Kumar";

   String s2=“Kumar";

   String s3=new String(“Kumaa");

   System.out.println(s1==s2);

   System.out.println(s1==s3);

}   }
```

Output:
true
false

Example for equals() vs ==

```
class  Demo1

public static void main(String  args[])

{

String s1 = "Hello";

String s2 = new String(s1);

System.out.println(s1  + " equals " + s2 + " -> " +s1.equals(s2));

System.out.println(s1  + " == " + s2 + " -> " + (s1 == s2));

} }
```

Output:

Hello equals Hello -> true

Hello == Hello -> false

3) String compare by compareTo() method

- Compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string
- It has this general form:

  int compareTo(String *str*).

  str is the **String** being compared with the invoking **Stri**

- Suppose s1 and s2 are two string variables. If:
  - s1 == s2 :0
  - s1 > s2 :positive value
  - s1 < s2 :negative value

Example

```
String s1="Sachin";

String s2="Sachin";

String s3="Ratan";

System.out.println(s1.compareTo(s2));

System.out.println(s1.compareTo(s3));

System.out.println(s3.compareTo(s1));
```

Output:

0

1

-1

Example

```
String s1="hello";

String s2="hello";

String s3="mello";

String s4="hemlo";

String s5="flag";

System.out.println(s1.compareTo(s2));

System.out.println(s1.compareTo(s3));

System.out.println(s1.compareTo(s4));

System.out.println(s1.compareTo(s5));
```

Output:

- 0 because both are equal
- -5 because "h" is 5 times lower than "m"
- -1 because "l" is 1 times lower than "m"
- 2 because "h" is 2 times greater ”f"

Example

```
String s1="Sachin";

String s2="sachin";

String s3="Ratan";

System.out.println(s1.compareTo(s2));

System.out.println(s1.compareTo(s3));

System.out.println(s3.compareTo(s1));
```

Output:
-32
1
-1

Java String compareTo(): empty string

- If you compare string with blank or empty string, it returns length of the string.
- If second string is empty, result would be positive. If first string is empty, result would be negative.

```
public class CompareToExample2{
public static void main(String args[])
{
String s1="hello";
String s2="";
String s3="me";
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s3));
}}
```

Output:
5
-2

String startsWith

- Checks if this string starts with given prefix.
- Returns true if this string starts with given prefix else returns false
- syntax

```
public boolean startsWith(String prefix)
```

Example

```
String s1="java string split method";

System.out.println(s1.startsWith("ja"));

System.out.println(s1.startsWith("java string"));
```

Output:

true

true

String endsWith

- Checks if this string ends with given suffix.
- Returns true if this string ends with given suffix else returns false.
- syntax

```
public boolean endsWith(String suffix)
```

Example:

```
String s1=“Acharya Institute of technology";

System.out.println(s1.endsWith(“yt”));

System.out.println(s1.endsWith("technology"));
```

Output:

false

true

Searching String

Presented By: Pankaj Kumar

String indexOf

- Returns index of given character value or substring.
- If it is not found, it returns -1.
- The index counter starts from zero.

4 types of indexOf method in java.

No.	Method	Description
1	int indexOf(char ch)	returns index position for the given char value
2	int indexOf(char ch, int startIndex)	returns index position for the given char value and from index
3	int indexOf(String substring)	returns index position for the given substring
4	int indexOf(String substring, int startIndex)	returns index position for the given substring and from index

Example

```
String s1="this is index of example";

int index1=s1.indexOf("is");

int index2=s1.indexOf("index");

System.out.println(index1+"   "+index2);

int index3=s1.indexOf("is",4);

System.out.println(index3);

int index4=s1.indexOf('s');

System.out.println(index4);
```

Output:

```
2    8
5
3
```

Java String lastIndexOf

- Returns last index of the given character value or substring.
- If it is not found, it returns -1.
- Index counter starts from zero.

No.	Method	Description
1	int lastIndexOf(int ch)	returns last index position for the given char value
2	int lastIndexOf(int ch, int fromIndex)	returns last index position for the given char value and from index
3	int lastIndexOf(String substring)	returns last index position for the given substring
4	int lastIndexOf(String substring, int fromIndex)	returns last index position for the given substring and from index

Example

```
String s1="this is index of example";

 index1=s1.lastIndexOf('s');

System.out.println(index1);
```

Output:

6

Example:

```
String str = "This is last index of example";
        int index = str.lastIndexOf('s',5);
        System.out.println(index);
        int index = str.lastIndexOf("of", 25);
        System.out.println(index);
        index = str.lastIndexOf("of", 10);
        System.out.println(index);
```

Output:

```
3
19
-1
```

Modifying a String

Substring in Java

- Part of string is called **substring**.
- Substring is a subset of another string.
- In case of substring startIndex is inclusive and endIndex is exclusive.

Substring as two methods:

- public String substring(int startIndex)
- public String substring(int startIndex, int endIndex)

Example

```
String s="SachinTendulkar";

System.out.println(s.substring(6));

System.out.println(s.substring(0,6));
```

Output:

Tendulkar

Sachin

String concat

- combines specified string at the end of this string.
- It returns combined string.
- Like appending another string.
- **Syntax**

```
String concat(String anotherString)
```

Example

```
String s1="java string";
s1.concat("is immutable");
System.out.println(s1);
s1=s1.concat(" is immutable so assign it explicitly");
System.out.println(s1);
```

Output:

```
java string
java string is immutable so assign it explicitly
```

Replace()

- Returns a string replacing all the old char or CharSequence to new char or CharSequence.
- There are two type of replace methods in java string.

  i. **public** String replace(**char** oldChar, **char** newChar)

  ii. public String replace(CharSequence target, CharSequence replacement)

Example

```
String s1=“AST is a very good college";

String replaceString=s1.replace(‘S',’I');

System.out.println(replaceString);
```

Output:

AIT is very good college

Java String replace(CharSequence target, CharSequence replacement)

```
String s1="my name is ABC my name is college";

String replaceString=s1.replace("is","was");

System.out.println(replaceString);
```

Output:

```
my name was ABC my name was college
```

String trim

- Eliminates leading and trailing spaces.
- **Syntax**

```
public String trim()
```

Example

```
String s1="  hello string  ";
System.out.println(s1+"java");
System.out.println(s1.trim()+"java");
```

Output:

```
   hello string java
hello stringjava
```

Example

```
String s1 ="  hello java string   ";
System.out.println(s1.length());
 System.out.println(s1);                          //Without trim()
String tr = s1.trim();
System.out.println(tr.length());
System.out.println(tr);                          //With trim()
```

Output:

```
22
  hello java string
17
hello java string
```

Changing the case of character within a string

- **String toLowerCase():** method returns the string in lowercase letter.
- **String toUpperCase():** method returns the string in uppercase letter.

Example

```
String s1=“Acharya Institute of Technology";

String s1lower=s1.toLowerCase();

System.out.println(s1lower);

String s1upper=s1.toUpperCase();

System.out.println(s1upper);
```

Output:

```
acharya institute of technology

ACHARYA INSTITUTE OF TECHNOLOGY
```

String Buffer

Presented By: Pankaj Kumar

StringBuffer

- Used to create mutable (modifiable) string.
- StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

Constructors of StringBuffer class

Constructor	Description
StringBuffer()	creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)	creates a string buffer with the specified string.
StringBuffer(int capacity)	creates an empty string buffer with the specified capacity as length.

StringBuffer append() method

- Append() method concatenates the given argument with this string.
- It has several overloaded versions. Here are a few of its forms:
  - StringBuffer append(String *str*)
  - StringBuffer append(int *num*)
  - StringBuffer append(Object *obj*)

Example

```
StringBuffer sb=new StringBuffer("Hello ");

sb.append(“AIT");

System.out.println(sb);
```

Output:

Hello AIT

Example

```
StringBuffer sb = new StringBuffer("Hello ");

sb.append("World ");

sb.append(2017);

System.out.println(sb);`
```

Output:
Hello World 2017

StringBuffer insert() method

- Used to insert text at the specified index position.
- These are a few of its forms:
  - StringBuffer insert(int *index*, String *str*)
  - StringBuffer insert(int *index*, char *ch*)
  - StringBuffer insert(int *index*, Object *obj*)

Example

```
StringBuffer sb=new StringBuffer("Hello ");

sb.insert(1,“AIT");

System.out.println(sb);
```

Output:

HAITello

Example for insert().

```
class insertDemo {

public static void main(String args[]) {

StringBuffer sb = new StringBuffer("I Java!");

sb.insert(2, "like ");

System.out.println(sb);

}
```

Output:

I like Java!

```
char[] c1 = new char[] {'Y','e','s'};

StringBuffer sb3 =  new StringBuffer("Hello  World");

sb3.insert(6,c1);

System.out.println(sb3);

float f = 2.0f;

StringBuffer sb5 =  new StringBuffer("Hello  World");

sb5.insert(6,f);

 System.out.println(sb5);

Object obj = new String("My");

StringBuffer sb8 =  new StringBuffer("Hello  World");

sb8.insert(6,obj);

System.out.println(sb8);
```

Output:
Hello Yes World
Hello 2.0 World
Hello My World

StringBuffer replace() method

- replace() method replaces the given string from the specified beginIndex and endIndex.
- Its signature is shown here:

StringBuffer replace(int *startIndex*, int *endIndex*, String *str*)

Example

```
StringBuffer sb=new StringBuffer(" Hello World ");

sb.replace( 6, 11, "java");

System.out.println(sb);
```

Output:

Hellojavad

```
StringBuffer sb = new StringBuffer("program compile time");
        System.out.println("string: "+sb);
        System.out.println("after replace: "+sb.replace(8, 15, "run"));
```

Output:
string: program compile time
after replace: program run time

StringBuffer delete() method

- deletes the string from the specified beginIndex to endIndex.
- These methods are shown here:
  - StringBuffer delete(int *startIndex*, int *endIndex*)

Example

```
StringBuffer sb=new StringBuffer("Hello");

sb.delete(1,3);

System.out.println(sb);
```

Output:

Hlo

deleteCharAt( )

- **deleteCharAt( )** method deletes the character at the index specified by *loc*.

  StringBuffer deleteCharAt(int *loc*)

- It returns the resulting **StringBuffer** object.

demonstrates the delete( ) and deleteCharAt( ) method

```
class deleteDemo {

public static void main(String args[]) {

StringBuffer sb = new StringBuffer("This is a test.");

sb.delete(4, 7);

System.out.println("After delete:" + sb);

sb.deleteCharAt(0);

System.out.println("After deleteCharAt:" + sb);

} }
```

Output:
After delete: This a test.
After deleteCharAt: his a test.

StringBuffer reverse() method

- reverses the characters within a **StringBuffer** object.

SringBuffer reverse( )

Example

```
StringBuffer  str = new StringBuffer("Hello");

 str.reverse();

System.out.println(str);
```

Output:

olleH

capacity()

- Returns the **current capacity** of StringBuffer object.

- **Example:**

```
StringBuffer str = new StringBuffer();

System.out.println( str.capacity() );
```

Output:

16

ensureCapacity()

- Used to ensure minimum capacity of **StringBuffer** object.
- If the argument of the ensureCapacity() method is less than the existing capacity, then there will be no change in existing capacity.
- If the argument of the ensureCapacity() method is greater than the existing capacity, then there will be change in the current capacity using following rule:

newCapacity = (oldCapacity*2) + 2.

Example

```
StringBuffer  str = new StringBuffer();

System.out.println(  str.capacity());

str.ensureCapacity(30);

System.out.println(  str.capacity());
```

Output:

34

length( ) and capacity( ):

- Length of a StringBuffer can be found by the length( ) method,
- the total allocated capacity can be found by the capacity( ) method
- general forms:
  - int length( )
  - int capacity()

Example

```
StringBuffer s=new StringBuffer(“AIT");

 int p=s.length();

int q=s.capacity();

System.out.println("Length of string AIT="+p);

System.out.println("Capacity of string AIT="+q);
```

Output:

- Length of string Atria=3
- Capacity of string Atria=19 **(because room for 16 additional character is automatically added)**

charAt( ) and setCharAt( )

- ***charAt()*** method returns a char value at the given index number.
- ***setCharAt()*** is used to set the specified character at the given index.
  - char charAt(int *where*)
  - void setCharAt(int *where*, char *ch*)

Demonstrate charAt() and setCharAt().

```
StringBuffer sb = new StringBuffer("Hello");

System.out.println("buffer before = " + sb);

System.out.println("charAt(1) before = " + sb.charAt(1));

sb.setCharAt(1, 'i');

sb.setLength(2);

System.out.println("buffer after = " + sb);

System.out.println("charAt(1) after = " + sb.charAt(1));
```

Output:
buffer before = Hello
charAt(1) before = e
buffer after = Hi
charAt(1) after = i

StringBuilder class

- StringBuilder is identical to StringBuffer except for one important difference that it is not synchronized, which means it is not thread safe.

Differences

String	StringBuffer
The length of the String object is fixed.	The length of the StringBuffer can be increased.
String object is immutable.	StringBuffer object is mutable.
Performance is slower during concatenation.	Performance is faster during concatenation.
Consumes more memory.	Consumes less memory.
Stores the strings in String constant pool.	Stores the strings in Heap Memory

Differences

StringBuffer	StringBuilder
StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
StringBuffer is less efficient than StringBuilder.	StringBuilder is more efficient than StringBuffer