Object Oriented Programming in Java

Er.Sital Prasad Mandal

BCA- 2nd sem
Mechi Campus
Bhadrapur, Jhapa, Nepal
(Email : info.sitalmandal@gmail.com)
https://ctaljava.blogspot.com/

Text Book

1. Deitel & Dietel. -Java: How to-program-. 9th Edition. TearsorrEducation. 2011, ISBN: 9780273759168

2. Herbert Schildt. "Java: The CoriviaeReferi4.ic e 61 Seventh Edition. McGraw -Hill 2006, 1SBN; 0072263857

Handling Strings

1. Creation
2. Concatenation and Conversion of a String
3. Changing Case
4. Character Extraction
5. String Comparision
6. Searching Strings
7. Modifying Strings
8. String Buffer

Introduction

- String is a sequence of characters.
- In the Java programming language, strings are objects.
- Java provides a class called “String” in “java.lang” package to create and manipulate strings.
- The string value must be represented in “ ” (double cotes).

Creating String

Declaring & Initialising

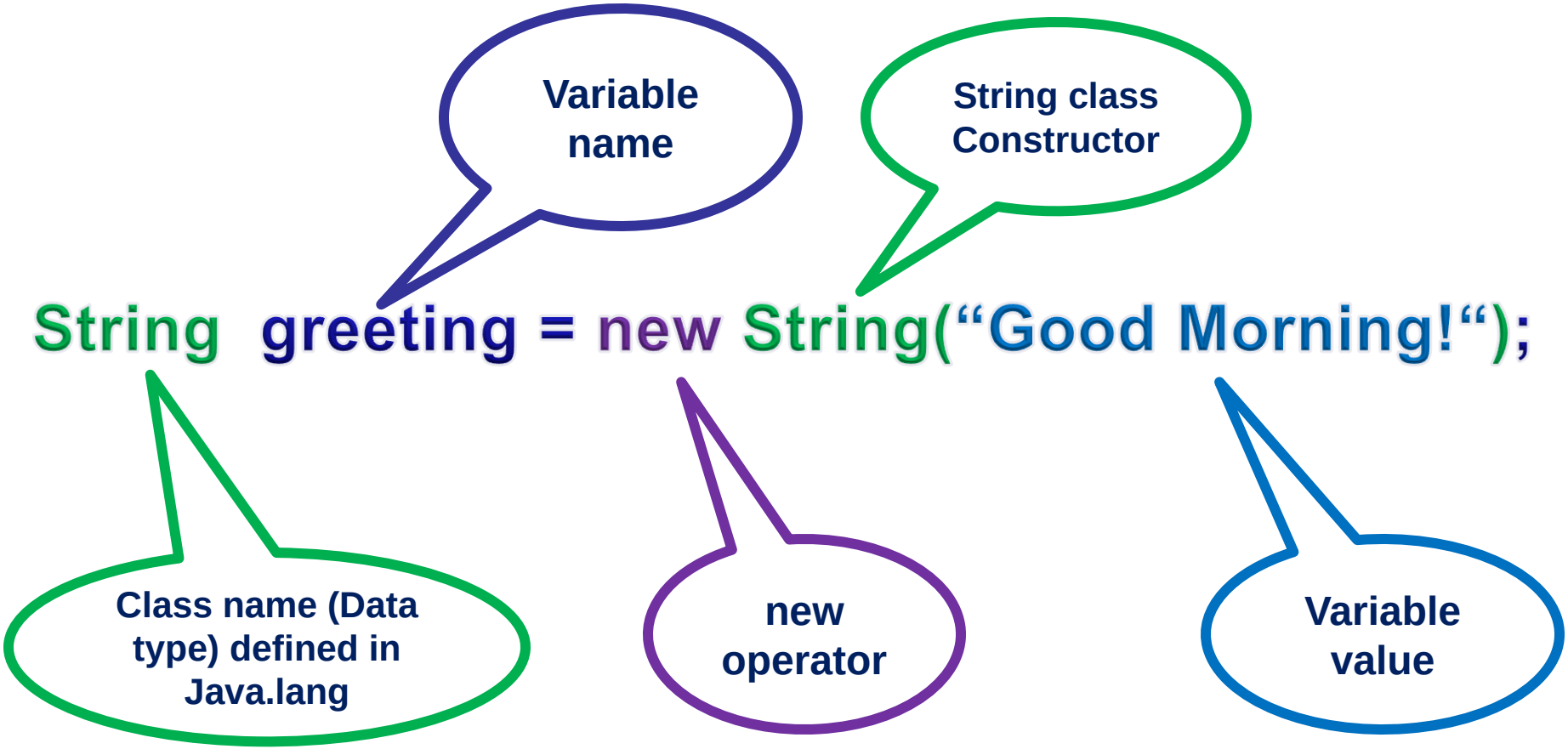
```
char Array[ ] = new char [5];
```

```
char  msg[ ] = {‘’J,’A’,’V’,’A’};
String  name = new String(msg);
```

Here, first we create a character array and pass that array as argument to the string class constructor.

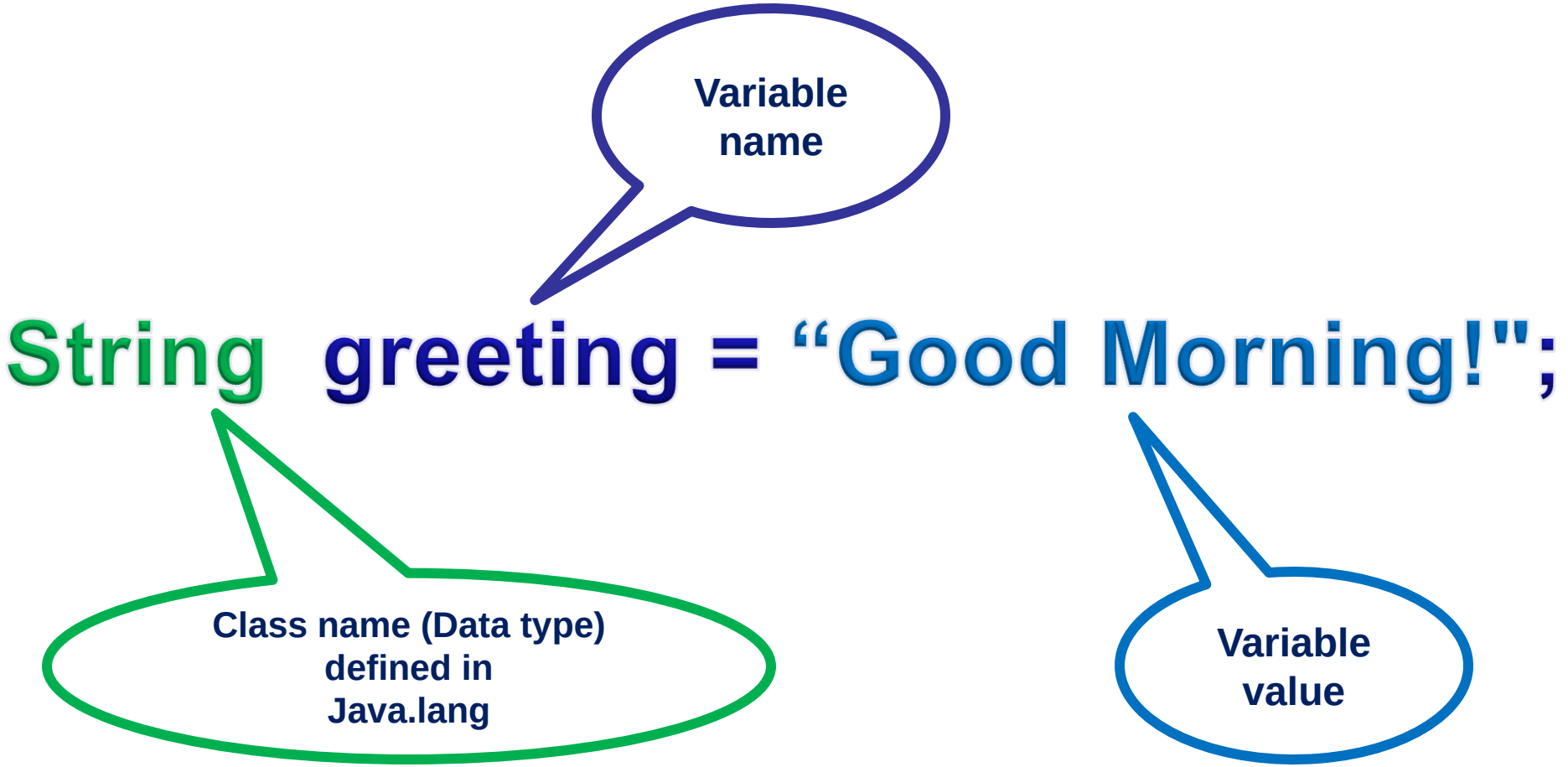
Creating String

Another way to create a string



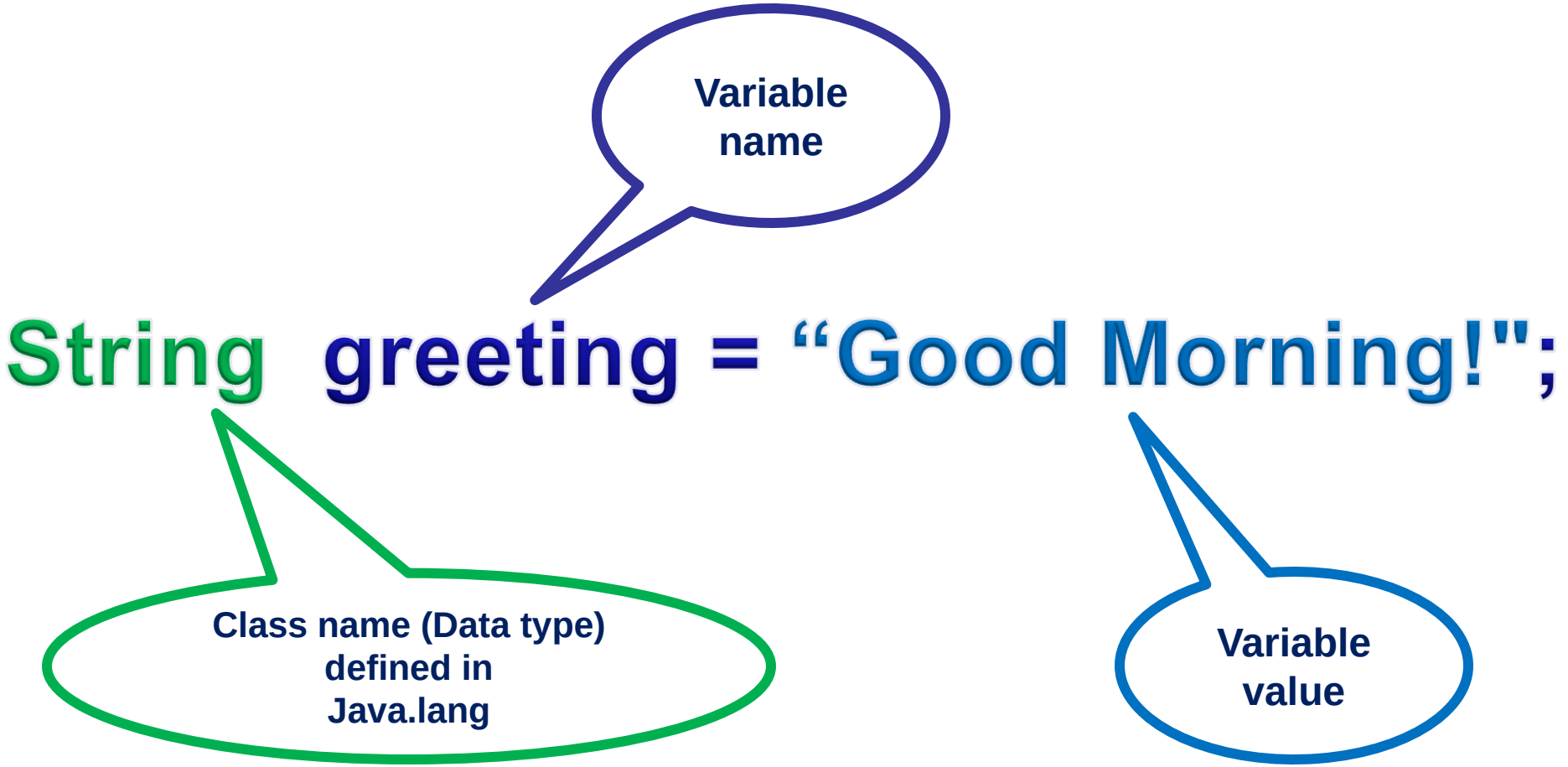
Creating String

Direct way to create a string



Creating String

Direct way to create a string



Concatenation String

concat()

String concatenation is the operation of joining two string objects end to end.

For example, the concatenation of two strings "Foot" and "ball" is "Football"

Syntax:

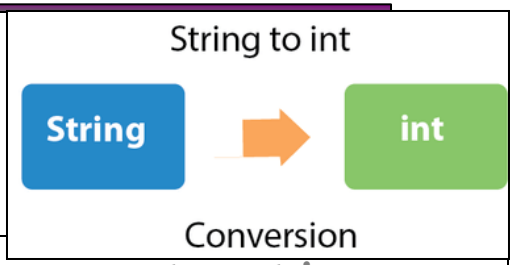
```
stringVariable1. concat(stringVariable2);
```

Example:

```
greeting.concat(name);
```

Conversion of a String

Java Convert String to int



We can convert **String to an int in java** using ***Integer.parseInt()*** method.

```
//Java Program to understand the working of string concatenation operator
public class StringToIntExample1 {
    public static void main ( String[] args ) {
//Declaring String variable
        String s = "200";
//Converting String into int using Integer.parseInt()
        int i = Integer.parseInt(s);
//200100, because "200"+100, here + is a string concatenation operator
        System.out.println(s + 100);
//300, because 200+100, here + is a binary plus operator
        System.out.println(i + 100);
}
}
```


https://www.javatpoint.com/java-string-to-int

Conversion of a String

Java Convert String to integer

To convert String into Integer , we can use ***Integer.valueOf()*** method which returns instance of Integer class. The Integer.valueOf() method converts String into Integer object.

```
//Java Program to demonstrate the conversion of String into Integer
//using Integer.valueOf() method
public class StringToIntegerExample2 {
    public static void main ( String[] args ) {
//Declaring a string
        String s = "200";
//converting String into Integer using Integer.valueOf() method
        Integer i = Integer.valueOf(s);
        System.out.println(i);
    }

}
```

Output:
200

https://www.javatpoint.com/java-string-to-int

Conversion of a String

NumberFormatException Case

If you don't have numbers in string literal, calling ***Integer.parseInt()*** *or* ***Integer.valueOf()*** *methods* ***throw NumberFormatException****.*

```
//Java Program to demonstrate the case of NumberFormatException
public class StringToIntegerExample3 {
    public static void main ( String[] args ) {
        String s = “Java";
        int i = Integer.parseInt(s);
        System.out.println(i);
    }

}
```

Output:

Exception in thread "main" java.lang.NumberFormatException: For input string: “Java"

https://www.javatpoint.com/java-string-to-int

Assignment of Conversion of a String

Signature of parseInt()

The parseInt() is the static method of Integer class.

The **signature** of parseInt() method is given below:

public static int parseInt(String s)

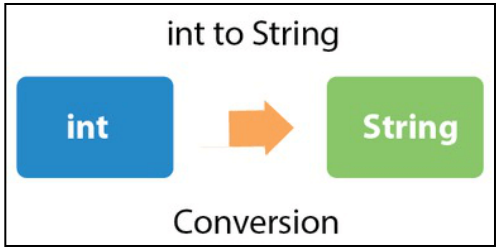
1. ***Integer.parseInt()***
2. ***Integer.valueOf()***

https://www.javatpoint.com/java-string-to-int

Conversion of a String

Java Convert int to String

We can convert int to String in java using **String.valueOf()** and **Integer.toString()** methods. Alternatively, we can use String.format() method, string concatenation operator etc.



public static String valueOf(**int** i)

```
public class IntToStringExample1 {
    public static void main ( String[] args ) {
        int i = 200;
        String s = String.valueOf(i);
        //300 because + is binary plus operator
        System.out.println(i + 100);
        //200100 because + is string concatenation operator
        System.out.println(s + 100);
    }
}
```

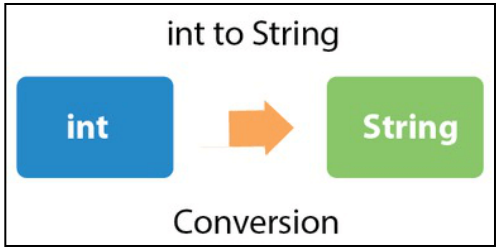

https://www.javatpoint.com/java-string-to-int

Conversion of a String

Java Convert int to String

Integer.toString() methods

public static String toString(**int** i)

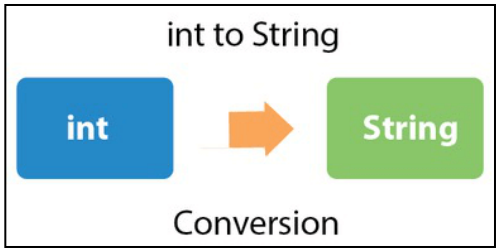



```
public class IntToStringExample2 {
    public static void main ( String[] args ) {
        int i = 200;
        String s = Integer.toString(i);
        //300 because + is binary plus operator
        System.out.println(i + 100);
        //200100 because + is string concatenation operator
        System.out.println(s + 100);
    }
}
```


https://www.javatpoint.com/java-string-to-int

Conversion of a String

Java Convert int to String



Java int to String Example using String.format()

public static String format(String format, Object... args)

```
public class IntToStringExample3 {
    public static void main ( String[] args ) {
        int i = 200;
        String txt = "ram";
        String s = String.format("%d", i);
        String stxt = String.format("%s", txt);
        System.out.println(s);
        System.out.println(stxt);
    }
}
```

```
200
ram
```

https://www.javatpoint.com/java-string-to-int

Changing Case

`String` Methods

Method name	Description
indexOf(str)	index where the start of the given string appears in this string (-1 if not found)
length()	number of characters in this string
substring(index1, index2) or substring(index1)	the characters in this string from index1 (inclusive) to index2 (exclusive); if index2 is omitted, grabs till end of string
toLowerCase()	a new string with all lowercase letters
toUpperCase()	a new string with all uppercase letters
charAt(index)	Returns the character at the specified index

- These methods are called using the dot notation:

```
String name = "Hello World";
System.out.println(name.length());
```

Changing Case

```
public class ChangingCaseDemo {
    public static void main ( String[] args ) {
        // index     012345678910
        String s1 = "HELLo World";
        System.out.println(s1.length());
        System.out.println(s1.indexOf("r"));
        System.out.println(s1.charAt(10));
        System.out.println(s1.substring(3, 7));
        String s2 = s1.substring(0);
        System.out.println(s2.toLowerCase());
        System.out.println(s2.toUpperCase());
    }
}
```

```
11
8
d
Lo W
hello world
HELLO WORLD
```

Character Extraction

There are several ways by which characters can be extracted from String class object. String is treated as an object in Java so we can't directly access the characters that comprise a string. For doing this String class provides various predefined methods.

charAt()

charAt() method is used to extract a single character at an index. It has following syntax.

Syntax:

char charAt(**int** index)

```
1 class temp
2 {
3 public static void main(String...s)
4 {
5 String str="Hello";
6 char ch=str.charAt(2);
7 System.out.println(ch);
8 }
9 }
```

Output

l

Character Extraction

getChars()

It is used to extract more than one character. getChars() has following syntax.

Syntax:

void getChars(**int** stringStart, **int** stringEnd, **char** storearr[], **int** arrStart)

```
1  class temp
2  {
3  public static void main(String...s)
4  {
5  String str="Hello World";
6  char ch[]=new char[4];
7  str.getChars(1,5,ch,0);
8  System.out.println(ch);
9  }
10 }
```

Output

ello

Character Extraction

getBytes()

The Java String getBytes() method encodes the string into a sequence of bytes and stores it in a byte array.

getBytes() extract characters from String object and then convert the characters in a byte array. It has following syntax.

Syntax:

byte [] getBytes()

```
import java.util.Arrays;
public class temp {
    public static void main(String...s)
    {
        String str="Java";
        byte b[]=str.getBytes();
        System.out.println(Arrays.toString(b));
    }
}
```

Output

[74, 97, 118, 97]

Character Extraction

The syntax of the String getBytes() method are:

string.getBytes(Charset charset)

Here, string is an object of the String class.
The getBytes() method returns a byte array.

```
import java.util.Arrays;
import java.nio.charset.Charset;
public class getBytesDemo2 {
    public static void main ( String[] args ) {
        String str = "Java";
        byte[] byteArray;
        // using UTF-8 for encoding
        byteArray = str.getBytes(Charset.forName("UTF-8"));
        System.out.println(Arrays.toString(byteArray));
    }
}
```

[74, 97, 118, 97]

Character Extraction

The syntax of the String getBytes() method are:

string.getBytes(String charsetName)

Here, string is an object of the String class.
The getBytes() method returns a byte array.

```
import java.util.Arrays;

public class getBytesDemo3 {
    public static void main ( String[] args ) {
        String str = "Java";
        byte[] byteArray;
        try {
            byteArray = str.getBytes("UTF-8");
            System.out.println(Arrays.toString(byteArray));

        } catch (Exception e) {
            System.out.println(e + " encoding is wrong");
        }
    }
}
```

[74, 97, 118, 97]

Character Extraction

Here are different CharSet available in java:

- **UTF-8** - Eight-bit UCS Transformation Format (Normal understand Encode)
- **UTF-16** - Sixteen-bit UCS Transformation Format
- **UTF-16BE** - Sixteen-bit UCS Transformation Format, big-endian byte order
- **UTF-16LE** - Sixteen-bit UCS Transformation Format, little-endian byte order
- **US-ASCII** - Seven-bit ASCII
- **ISO-8859-1** - ISO Latin Alphabet No. 1

Character Extraction

toCharArray()

It is an alternative of getChars() method. toCharArray() convert all the characters in a String object into an array of characters. It is the best and easiest way to convert string to character array. It has following syntax.

Syntax:

char [] toCharArray()

```
1  class temp
2  {
3  public static void main(String...s)
4  {
5  String str="Hello World";
6  char ch[]=str.toCharArray();
7  System.out.println(ch);
8  }
9  }
```

Output

```
Hello World
```

Character Extraction

toCharArray()

It is an alternative of getChars() method. toCharArray() convert all the characters in a String object into an array of characters. It is the best and easiest way to convert string to character array. It has following syntax.

Syntax:

char [] toCharArray()

```
1  class temp
2  {
3  public static void main(String...s)
4  {
5  String str="Hello World";
6  char ch[]=str.toCharArray();
7  System.out.println(ch);
8  }
9  }
```

Output

```
Hello World
```

String Comparision

compareTo()

The compareTo() method compares two strings lexicographically (in the dictionary order). It returns a positive number, negative number, or 0.

The comparison is based on the Unicode value of each character in the strings.

if s1 > s2, it returns positive number
if s1 < s2, it returns negative number
if s1 == s2, it returns 0

compareToIgnoreCase()

The Java String compareTo() method compares two strings lexicographically (in the dictionary order), ignoring case differences.

The syntax of the string compareToIgnoreCase() method is:

string.compareToIgnoreCase(String str)

String Comparision

compareTo()

```
public class CompareToDemo {
    public static void main(String[] args) {
        String s1="hello";
        String s2="hello";
        String s3="meklo";
        String s4="hemlo";
        String s5="flag";
        //0 because both are equal
        System.out.println(s1.compareTo(s2));
        //-5 because "h" is 5 times lower than "m"
        System.out.println(s1.compareTo(s3));
        //-1 because "l" is 1 times lower than "m"
        System.out.println(s1.compareTo(s4));
        //2 because "h" is 2 times greater than "f"
        System.out.println(s1.compareTo(s5));
    }
}
```

```
0
-5
-1
2
```

String Comparision

equals()

The equals() method returns true if two strings are equal. If not, it returns false.

equals() Return Value

- **returns true** if the strings are equal
- **returns false** if the strings are not equal
- **returns false** if the str argument is null

equalsIgnoreCase()

The Java String equalsIgnoreCase() method compares two strings, ignoring case differences. If the strings are equal, equalsIgnoreCase() returns true. If not, it returns false.

The syntax of the string equalsIgnoreCase() method is:

string.equalsIgnoreCase(String str)

String Comparision

equals()

```
public class EqualsExample {
    public static void main ( String[] args ) {
        String str1 = "Learn Java";
        String str2 = "Learn Java";
        String str3 = new String("Learn Java");
        String str4 = "Learn Kolin";
        // comparing str1 with str2
        boolean result ;
        result= str1.equals(str2);
        System.out.println(result); //true

        System.out.println(str1.equals(str3)); //true

        // comparing str3 with str1
        result = str3.equals(str4);
        System.out.println(result);  // false
    }
}
```

String Comparision

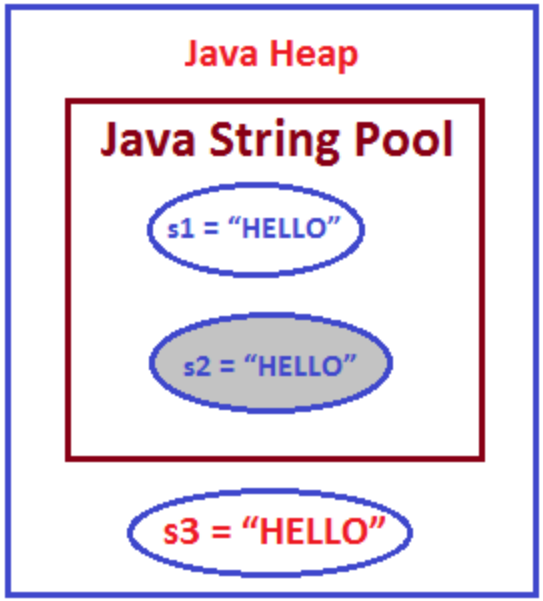
== operator

== operator compares reference or memory location of objects in a heap, whether they point to the same location or not.

Whenever we create an object using the operator new, it will create a new memory location for that object. So we use the == operator to check memory location or address of two objects are the same or not.

```
public class EqualOperatorDemo {
    public static void main(String[] args)
    {
        String s1 = "HELLO";
        String s2 = "HELLO";
        String s3 =  new String("HELLO");

        System.out.println(s1 == s2); // true
        System.out.println(s1 == s3); // false
        System.out.println(s1.equals(s2)); // true
        System.out.println(s1.equals(s3)); // true
    }
}
```



Assignment

Difference between comparing String using == and .equals() method in Java

1. The main difference between the .equals() method and == operator is that one is a method, and the other is the operator.
2. We can use == operators for reference comparison (**address comparison**) and .equals() method for **content comparison**. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.
3. If a class does not override the equals method, then by default, it uses the equals(Object o) method of the closest parent class that has overridden this method.

Searching Strings

The String class provides 2 methods for searching a string.

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

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

Syntax:

```
stringVariable.indexOf(charVariable);
stringVariable.lastIndexOf(charVariable);
stringVariable.indexOf(stringVariable);
stringVariable.indexOf(charVariable, startIndex);
stringVariable.lastIndexOf(charVariable, startIndex);
```

Modifying Strings

Because **String** objects are immutable, whenever you want to modify a **String**, you must either copy it into a **StringBuffer** or **StringBuilder**, or use a **String** method that constructs a new copy of the string with your modifications complete.

we will discuss below methods for modifying a String objects.

- *substring()*
- *concat()*
- *replace()*
- *replaceAll()*
- *replaceFirst()*
- *trim()*

Modifying Strings

replace( ) methods:

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*. The resulting *string* is returned.

For example,

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

puts the string "Hewwo" into s.

The ***second form*** of replace( ) replaces one character sequence with another.

It has this general form:

String replace(CharSequence original, CharSequence replacement)

Modifying Strings

replaceAll(String regex, String replacement)

Replaces each substring of this string that matches the given regular expression with the given replacement.

replaceFirst(String regex, String replacement)

Replaces the first substring of this string that matches the given regular expression with the given replacement.

Modifying Strings

```
public class ReplaceExample {
    public static void main(String[] args) {
        String str = "javaguides";
        System.out.println(str.length());
        String subStr = str.replace('a', 'b');
        System.out.println("replace string 1 : " + subStr);
        subStr = str.replace("guides", "programmer");
        System.out.println("replace string 2 : " + subStr);
        subStr = str.replaceAll("[a-z]", "java");
        System.out.println("replace string 3 :"+ subStr);
        subStr = str.replaceFirst("[a-z]", "java");
        System.out.println("replace string 4 :" + subStr);
    }
}
```

```
10
replace string 1 : jbvbguides
replace string 2 : javaprogrammer
replace string 3 :javajavajavajavajavajavajavajavajavajava
replace string 4 :javaavaguides
```

Modifying Strings

trim( )

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

The *trim( )* method is quite useful when you process user commands.

It has this general form:

String trim( )


```
public class TrimExample {
    public static void main(String[] args) {
        String str = "javaguides ";
        String subStr = str.trim();
        System.out.println("trim space string 1 : " + subStr);

        String s = "     Hello World     ".trim();
        System.out.println("trim space string 2 : " + s);
    }
}
```

```
trim space string 1 : javaguides
trim space string 2 : Hello World
```

String Buffer

Java *StringBuffer* class is used to create a mutable (modifiable) string. The *StringBuffer* class in Java is the same as **String** class except it is mutable i.e. it can be changed.

StringBuffer may have characters and substrings inserted in the middle or appended to the end.

StringBuffer will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

String Buffer

StringBuffer defines these four constructors:

1. *StringBuffer( )*
2. *StringBuffer(int size)*
3. *StringBuffer(String str)*
4. *StringBuffer(CharSequence chars)*

String Buffer

StringBuffer()

Reserves room for 16 characters without reallocation.

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

Example:

```
StringBuffer buffer = new StringBuffer();
System.out.println(buffer.capacity());
```

Output:

```
16
```

String Buffer

StringBuffer(int capacity)

- Constructs a string buffer with no characters in it and the specified initial capacity.
- The parameters capacity of the initial capacity.

This method throws *NegativeArraySizeException*
- if the capacity argument is less than 0.

Example:

```
StringBuffer buffer4 = new StringBuffer(20);
System.out.println(buffer4.capacity());
```

Output:

20

String Buffer

StringBuffer(String str)

- Constructs a string buffer initialized to the contents of the specified string.
- The initial capacity of the string buffer is 16 plus the length of the string argument.

- the parameter *str* the initial contents of the buffer.

Example:

```
StringBuffer buffer2 = new StringBuffer("javaguides");
System.out.println(buffer2.capacity());
```

Output:

```
26
```

Note that the capacity in the above example, it gives String length plus (16) initial capacity.

String Buffer

StringBuffer(CharSequence seq)

- Constructs a string buffer that contains the same characters as the specified *CharSequence*.
- The initial capacity of the string buffer is 16 plus the length of the *CharSequence* argument.
- If the length of the specified *CharSequence* is less than or equal to zero, then an empty buffer of capacity 16 is returned.

Example:

```
CharSequence charSequence = new StringBuilder("Hello");
StringBuffer buffer3 = new StringBuffer(charSequence);
System.out.println(buffer3);
System.out.println(buffer3.capacity());
```

Output:

```
Hello
21
```

String Buffer Functions

- **length():-**Returns the current length of the string.
- **capacity():-** Returns the total allocated capacity.
- **void ensureCapacity():-** Preallocates room for a certain number of characters.
- **void setLength(int *len*):-** Sets the length of the string s1 to len.
  If len<s1.length(), s1 is truncated.
  If len>s1.length(), zeros are added to s1.
- **charAt(int *where*):-** Extracts value of a single character.
- **setCharAt(int *where*, char *ch*):-** Sets the value of character at specified position.

String Buffer Functions

- **append*(s2)*:-** Appends string s2 to s1 at the end.
- **insert*(n,s2)*:-** Inserts the string s2 at the position n of the string s1
- **reverse**():- Returns the reversed object on when it is called.
- **delete*(int n1,int n2)*:-** Deletes a sequence of characters from the invoking object.

  n1 → Specifies index of first character to remove

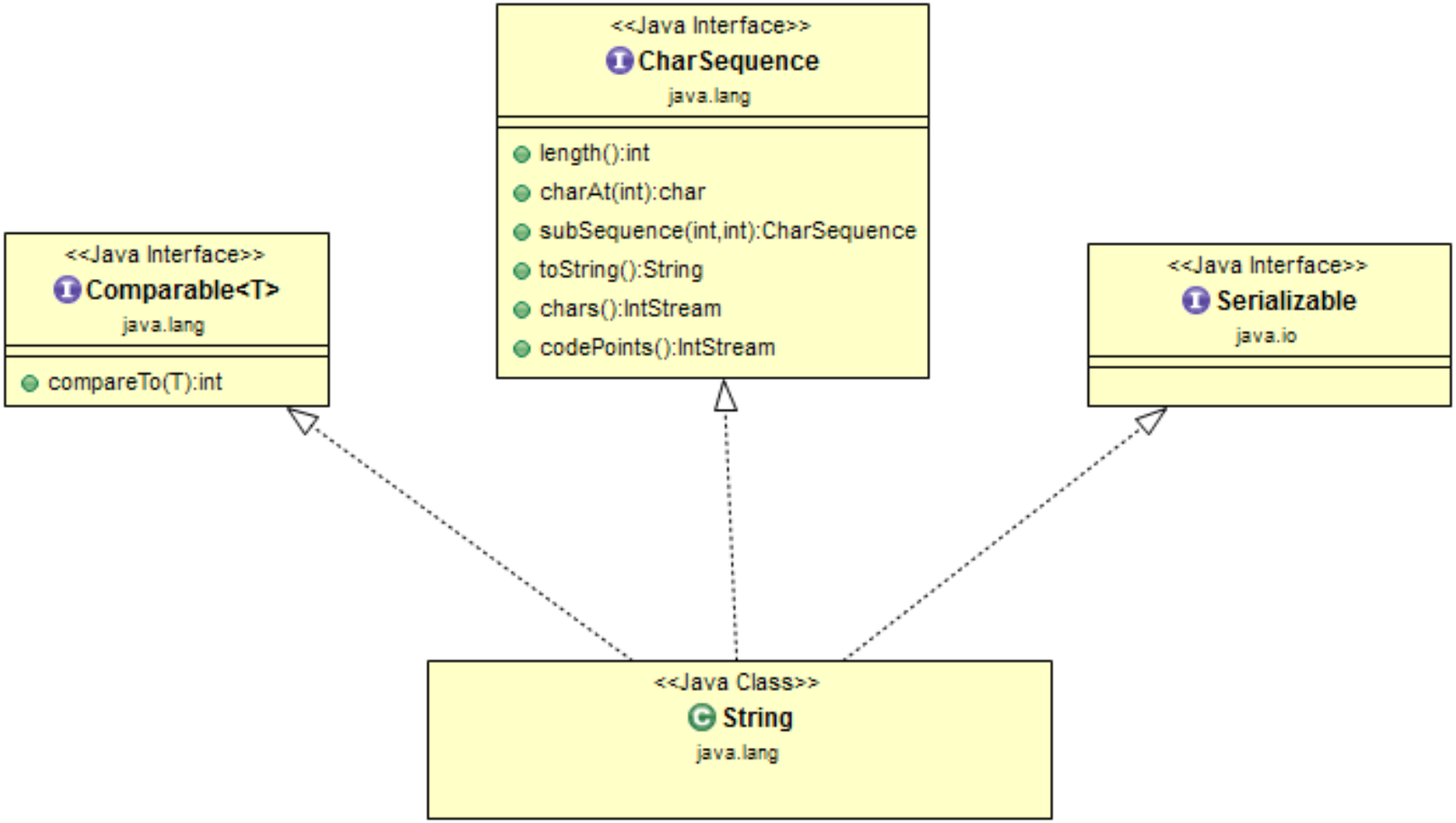
  n2 → Specifies index one past the lastcharacter to remove

- **deleteCharAt*(int loc)*:-** Deletes the character at the index specified by loc.

String Buffer Functions

- *replace(int n1,int n2,String s1):-* Replaces one set of characters with another set.
- substring(*int startIndex*):- Returns the substring that starts at starts at *startIndex* and runs to the end.
- substring(*int startIndex, int endIndex*):- Returns the substring that starts at starts at *startIndex* and runs to the *endIndex-1*

Review



String class implemented interfaces