# CS205 Object Oriented Programming in Java

# Module 4 - **Advanced features of Java** (Part 1)

**Prepared by**

**Renetha J.B.**

AP

Dept.of CSE,

Lourdes Matha College of Science and Technology

## Topics

- ☑ Java Library
  - ☑ **String Handling**
    - ☑String Constructors
    - ☑String Length
    - ☑Special String Operations

## String Handling

- String is a **class** in Java.
- Java implements strings as **objects** of type **String.**
  - The String type is used to **declare string variables**
- Java has methods to compare two strings, search for a substring, concatenate two strings, and change the case of letters within a string.
- A quoted string constant(E.g. “hello”) can be assigned to a **String** variable**.**
- A variable of *type String* can be assigned to another variable of *type String*.

## String Constructors

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

**String**()

- For example,

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

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

## String Constructors
## (initialize array of characters)

- To create a **String** **initialized by an array of characters**, use the constructor

```
String(char chars[ ])
```

- Example:

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

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

# String Constructors

## (initialize with a subrange of character array)

- To initialize a string with a subrange of a character array(substring) the following constructor is used:

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

  - Here, *startIndex specifies the staindex at which the subrange begins, and*
  - *numChars specifies* the number of characters to use.

E.g.

```
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
                  0    1    2    3    4    5
String s = new String(chars, 2, 3);
```

- This initializes s with the characters starting from index 2 and number of letters =3. i.e. s will contain **cde.**

## String Constructors
**(initialize using another string)**

- We can construct a **String** object that contains the <u>same character sequence as another String object</u> using this constructor:

**String**(String *strObj)*

```
// Construct one String from another.
class MakeString {
public static void main(String args[])
{
      char c[] = {'J', 'a', 'v', 'a'};
      String s1 = new String(c);
      String s2 = new String(s1);
      System.out.println(s1);
      System.out.println(s2);
}
}
```

**OUTPUT**

```
Java
Java
```

## String Constructors
## (initialize using byte array)

- **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)*

- Here *asciiChars* specifies the array of bytes.
  - In each of these constructors, **the byte-to-character conversion** is done by using the default character encoding of the platform.

## String Constructors
**(initialize using byte array) contd.**

```
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
```

# String Constructors (contd.)

- We construct a String from a **StringBuffer** by using the constructor :

```
String(StringBuffer strBufObj)
```

- J2SE 5 added two constructors to String.

➢ The first supports the ***extended Unicode character set*** :

```
String(int codePoints[ ], int startIndex, int numChars)
```

  – Here, *codePoints is an array that contains Unicode code points*

➢ The second new constructor supports the new **StringBuilder** **class:-**

```
String(StringBuilder strBuildObj)
```

  – This constructs a **String** from the **StringBuilder** passed in *strBuildObj*.

## String Length

- The length of a string is the number of characters in the string

E.g. length of the string “hello” is 5

- The method **length()** is used to find the length of the string.

```
int length( )
```

```
class Stringlen
{
    public static void main(String args[])
    {
    String s="Hello";
    System.out.println("Length="+s.length());
    }
}
```

**OUTPUT**

```
Length=5
```

## Special String Operations

- These operations include
  - the *automatic creation of new String instances(object)* from **string literals**
  - **concatenation** of multiple String objects by use of the + operator, and
  - the **conversion** of other data types **to a string** representation.

## String Literals

- Java automatically constructs a **String** object **for each string literal** in our program,.
  - So we can use a string literal to initialize a **String object**

```
String s2 = "abc";
```

is same as

```
char chars[] = { 'a', 'b', 'c' };
String s2 = new String(chars);
```

- We can use a string literal at any place where we use a **String object.**
- String literals can call the length( ) method on the string

E.g.

System.out.println("abc"**.length()**);

# String Concatenation

- String concatenation is used to join two strings
- Method 1:The + operator can be used between strings to combine them. This is called concatenation.
  - ❑Operator + can be chained to concatenate many strings

```
String age = "9";
String s = "He is " + age + " years old.";
System.out.println(s);
```

- This fragment displays the string **He is 9 years old.**
- Instead of letting long strings wrap around within our source code, we can break them into smaller pieces, using the **+ to concatenate them**

## String Concatenation with Other Data Types

- We can concatenate strings with other types of data.
- If one of the **operand of the + is an instance of String** then compiler will convert other operand to its string equivalent.

```
String s = "four: " + 2 + 2;

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

  - This fragment displays
    **four: 22**
  - Operator precedence causes the concatenation of “four” with 2. So 2is converted into string and “four: ” concatenates with string equivalent of 2.
  - Then this result is then concatenated with the string equivalent of 2.
- Parentheses can be used for grouping integers and + to perform addition.

```
String s = "four: " + (2 + 2);
```

  - Here parentheses is first computed. So (2+2) is 4 then string “four: ” is concatenated with that. So **s** contains the string **“four: 4”**

## String Concatenation(contd.)

- Method 2:We can use **concat()** method to concatenate two strings.

```
String concat(String str)
```

- This method creates a new object that contains the invoking string with the contents of *str appended to the end.* ***concat( ) performs the same function as +.***
- ***For example,***

```
String s1 = "one";
String s2 = s1.concat("two");
```

  – puts the string “onetwo” into **s2**.

- It generates the same result as the following :

```
String s1 = "one";
String s2 = s1 + "two";
```

## String Conversion and toString( )

- When Java **converts** data into its string representation during concatenation, it calls one of the overloaded versions of the string conversion method **valueOf( )** by class **String.**
- **valueOf( )** is overloaded for all the simple types and for type Object
  - For the simple types, **valueOf( ) returns a string that contains the human-readable equivalent of the value** with which it is called.
  - For objects, **valueOf( ) calls the toString( )** method on the Object.

## String Conversion and toString( )

The valueOf() returns the string representation of the corresponding argument. Different overloaded form of valueOf() in String class.

- **valueOf(boolean b)** − Returns the string representation of boolean argument.
- **valueOf(char c)** − char argument.
- **valueOf(char[] data)** char array argument.
- **valueOf(char[] data, int offset, int count)** − specific subarray of the char array argument.
- **valueOf(double d)** − double argument.
- **valueOf(float f)** − float argument.
- **valueOf(int i)** − int argument.
- **valueOf(long l)** − long argument.
- **valueOf(Object obj)** − Object argument. (calls toString() method of the class Object(parent class of all classes n Java)

## toString( )(contd.)

- The **toString( )** method has this general form:

```
String toString( )
```

- When we try to print an object of a class, it will call method valueOf(object) which calls toString( ) function :-
  - if toString( ) is present (overridden) in the class, then it is called.
  - If there is no toString( ) function in the class, when we try to print an object of that class, it prints **clasname@the memory location of the object**(the hexidecimal address of where that **object** is stored in memory.)

## Without using toString()

```
class Box {
double width;
double height;
double depth;
Box(double w, double d, double h,) {
width = w;
height = h;
depth = d;
}
```
~~public String toString() {~~

~~return "Dimensions are " + width + " by " +depth + " by " + height + ".";~~

~~}~~

```
}
```

```
class toStringDemo {
public static void main(String args[]) {
Box b = new Box(10, 14,12);
String s = "Box b: "+ b;

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

**OUTPUT**

```
Box@106d69c
Box b: Box@1db9742
```

Here when we print the object **b** ,since there is no toString( ) function in the class it will call toString() in class Object and prints **clasname@the memory location of the object**
(Here it prints **Box**@106d69c)

# Using toString()

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

```
class StringDemo {
public static void main(String args[])
 {
Box b = new Box(10, 14,12);
String s = "Box b: "+ b;

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

**OUTPUT**

```
Dimensions are 10.0 by 14.0 by 12.0
Box b: Dimensions are 10.0 by 14.0 by 12.0
```

Class **Box's toString( )** method is **automatically invoked** when a **Box object** is used in a concatenation expression or used in **println( ).**

## Reference

- Herbert Schildt, Java: **The Complete Reference, 8/e, Tata McGraw Hill, 2011**.