CS6308- Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

MODULE II JAVA OBJECTS -1	L	T	P	EL
	3	0	4	3
Classes and Objects, Constructor, Destructor, Static instances, this, constants, Thinking in Objects, String class, Text I/O				
SUGGESTED ACTIVITIES : • Flipped classroom • Practical - Implementation of Java programs – using String class, Creating Classes and objects • EL – Thinking in Objects				
SUGGESTED EVALUATION METHODS: • Assignment problems • Quizzes				

String handling in java

Strings

- In Java a string is a sequence of characters.
- In Java a string is an Object
    - Other languages that implement strings as character arrays
- Strings are Immutable.
    - String object that is created cannot be changed.
- However, a variable declared as a String reference can be changed to point at some other String object at any time.

Strings

- String can be created using three string classes namely String, StringBuffer and StringBuilder
- Use the class called StringBuffer to perform changes in original strings.
- String, StringBuilder and StringBuffer classes are declared final and there cannot be subclasses of these classes.
- The String, StringBuffer, and StringBuilder classes are defined in java.lang.
- Java.lang is the default package in java
- Java.lang is automatically imported without needing to explicitly import classes from this package
    - Common classes in lang are String, Math, System, Object, Thread

Creating Strings

- The default constructor creates an empty string object.
  - String s = new String();
- Create string object that have initial values from a character array
  - String s= new String(char[] chars)
- Create string object using String literals
  - String s = "String Literal";
- **Examples:** **String str = "abc";**

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

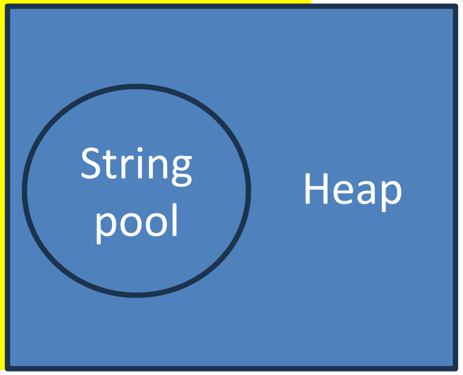
  String str = new String(data);
- Construct a string object by passing another string object.
  - String(String strObj)
  - Example: String str2 = new String(str);

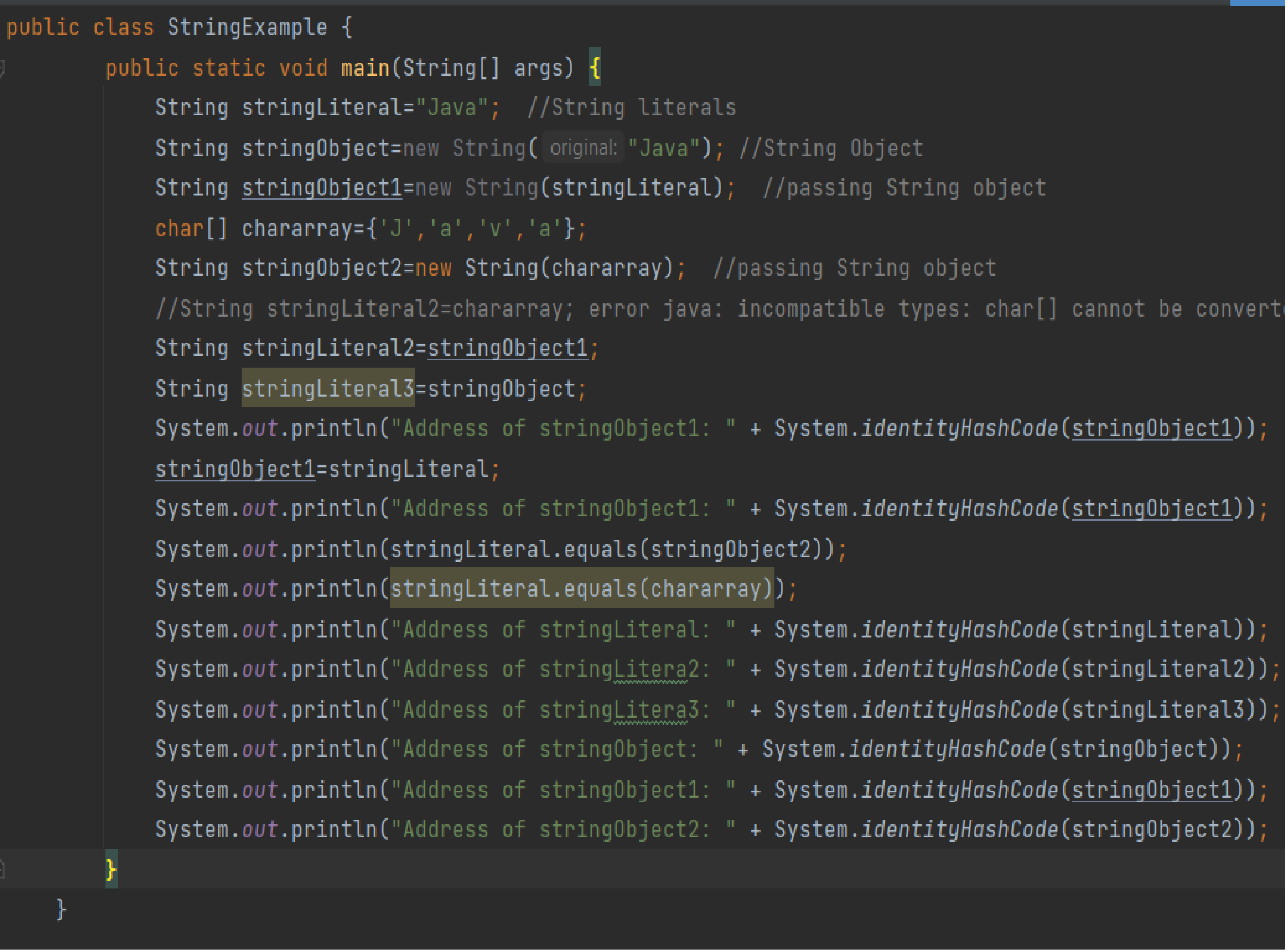
String memory

- The string pool or string constant pool, is a special area of the heap where Java stores unique string literals.
- String pool helps in saving memory and improving performance by avoiding duplicate strings.
- The heap is the runtime data area from which memory for all class instances and arrays is allocated.
- Strings created using the new keyword are allocated on the heap

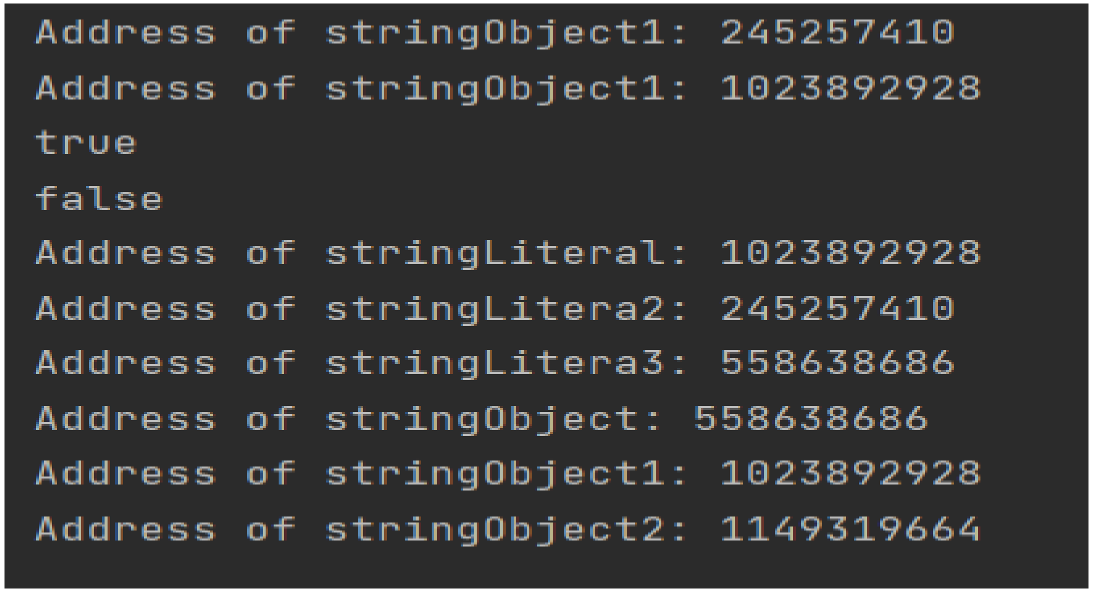
```
public class StringExample {
public static void main(String[] args) {
    String str1 = new String("hello"); // String created in the heap
    String str2 = "hello"; // string literal created in string pool of heap
    String str3 = "hello"; // Reuses the interned string literal
System.out.println(str1 == str2);
System.out.println(str1 == str3);
System.out.println(str1.equals(str2));
System.out.println(str1.equals(str3));

}}
```

```
false
true
true
```





```java
public class StringExample {
        public static void main(String[] args) {
            String stringLiteral="Java";  //String literals
            String stringObject=new String( original: "Java"); //String Object
            String stringObject1=new String(stringLiteral);  //passing String object
            char[] chararray={'J','a','v','a'};
            String stringObject2=new String(chararray);  //passing String object
            //String stringLiteral2=chararray; error java: incompatible types: char[] cannot be convert
            String stringLiteral2=stringObject1;
            String stringLiteral3=stringObject;
            System.out.println("Address of stringObject1: " + System.identityHashCode(stringObject1));
            stringObject1=stringLiteral;
            System.out.println("Address of stringObject1: " + System.identityHashCode(stringObject1));
            System.out.println(stringLiteral.equals(stringObject2));
            System.out.println(stringLiteral.equals(chararray));
            System.out.println("Address of stringLiteral: " + System.identityHashCode(stringLiteral));
            System.out.println("Address of stringLitera2: " + System.identityHashCode(stringLiteral2));
            System.out.println("Address of stringLitera3: " + System.identityHashCode(stringLiteral3));
            System.out.println("Address of stringObject: " + System.identityHashCode(stringObject));
            System.out.println("Address of stringObject1: " + System.identityHashCode(stringObject1));
            System.out.println("Address of stringObject2: " + System.identityHashCode(stringObject2));
        }
    }
```



```
Address of stringObject1: 245257410
Address of stringObject1: 1023892928
true
false
Address of stringLiteral: 1023892928
Address of stringLitera2: 245257410
Address of stringLitera3: 558638686
Address of stringObject: 558638686
Address of stringObject1: 1023892928
Address of stringObject2: 1149319664
```

Since char is not String type and hence
equals method return false

Creating Strings

- To specify a subrange of a character array as an initializer using the following constructor:
  - String(char chars[ ], int startIndex, int numChars)
- Here, startIndex specifies the index at which the subrange begins, and numChars specifies the number of characters to use.
- **Examples:**

```
char arr[] = {'J', 'A', 'V', 'A'};
String str = new String(arr,2,1);
String str11 = new String(arr,2,2);
System.out.println(arr);
System.out.println(str);
System.out.println(str11);
```

```
JAVA
V
VA
```

Creating Strings

- String class provides constructors that initialize a string when given a byte array.
  - String(byte chrs[ ])
  - String(byte chrs[ ], int startIndex, int numChars)
    - Here, chrs specifies the array of bytes. The second form allows you to specify a subrange.
  - **Examples:**

```
class PrintStringValues{
    public static void main(String args[]) {
        byte[] values = {65, 66,67, 68};
        String fullValues = new String(values);
        System.out.println("Full value " + fullValues);
        String partialValue = new String(values, 1, 3);
        System.out.println("Partial values: " + partialValue);
    }
}
```

```
Full value ABCD
Partial values: BCD
```

String METHODS

- int length( )
  - The length() method returns the length of the string.

    Eg: System.out.println(“Hello”.length()); // prints 5

```
char vowels[] = { 'a', 'e', 'i', 'o', 'u' };
String vowelString = new String(vowels);
System.out.println("Number of vowels: " +
vowelString.length());
```

- The + operator is used to concatenate two or more strings.

  Eg: String name = “Harry”

  String str = “Name : ” + name+ “.”;

- Java compiler converts an operand to a String whenever the other operand of the + is a String object.

String Concatenation with Other Data Types

```
double temperature = 25.5;
String weather = "The temperature is " + temperature + " degrees Celsius.";
System.out.println(weather);

String result = "Sum: " + (10 + 20);
System.out.println(result);
// output
// Sum: 30
// rather than
// Sum: 1020

String resultWithParentheses = "Sum: " + ((10 + 20));
// Now resultWithParentheses contains the string "Sum: 30".
```

String Conversion and toString( )

- To determine the string representation for objects of classes that is created.
- Classes that is created has to override toString( ) and provide your own string representations.
- The toString( ) method has this general form:
  - **String toString( )**
  - can be used in print( ) and println( ) statements and in concatenation expressions.

String toString()

```
class Car {
    String make;
    String model;
    int year;

    Car(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }

    public String toString() {
        return year + " " + make + " " + model;
    }
}
class CarDemo {
    public static void main(String args[]) {
        Car myCar = new Car("Toyota", "Corolla", 2022);
        String carDescription = "My car: " + myCar;

        System.out.println(myCar); // implicitly calls toString()
        System.out.println(carDescription);
    }
}
```

```
2022 Toyota Corolla
My car: 2022 Toyota Corolla
```

Character Extraction

Character Extraction

- The String class provides a number of ways in which characters can be extracted from a String object.
- The characters that comprise a string within a String object cannot be indexed as if they were a character array.
- Many of the String methods employ an index (or offset) into the string for their operation.
- Like arrays, the string indexes begin at zero.

Character Extraction

- public char **charAt**(int INDEX)
  - Returns the character at the specified index.
  - INDEX is the index of the character that is to be to obtained.
  - An index ranges from 0 to length() - 1.

```
char ch;
ch = “XYZ”.charAt(1); // ch = “Y”
```

- **Method getChars**

Get entire set of characters in `String`

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

s1.getChars(start, end, charArray, start);

compareTo( )

int compareTo(String str)

Here, str is the String being compared with the invoking String.

Character Extraction

```
public class StringExample2 {
   public static void main(String args[]) {
      //character extraction
      String email = "contact@example.com";
      int start=8;
      int end=15;
      char buffer[] = new char[end - start];
      email.getChars(start, end, buffer,0);
      System.out.println("Domain: " + new String(buffer));        }
   }
```

```
Domain: example
```

Searching Strings

int indexOf(int ch):

Finds the first occurrence of a character.

// Here ch is represented by its Unicode code point

// example '@' is represented by its Unicode code point 64

int indexOf(String str):

Finds the first occurrence of a substring.

int lastIndexOf(int ch):

Finds the last occurrence of a character.

int lastIndexOf(String str):

Finds the last occurrence of a substring.

int indexOf(int ch, int fromIndex):

Finds the first occurrence starting from a specified index.

int lastIndexOf(int ch, int fromIndex):

Finds the last occurrence searching backward from a specified index.

boolean contains(CharSequence sequence):

Checks if a substring is present.

boolean startsWith(String prefix):

Checks if the string starts with a specified prefix.

boolean endsWith(String suffix):

Checks if the string ends with a specified suffix.

```java
public class StringExample3 {
    public static void main(String[] args) {
        String email = "contact@domain.com";
        System.out.println("email:"+ email);
        int atIndex = email.indexOf('@’); // 1. indexOf(int ch)
        System.out.println("Index of '@': " + atIndex);
        int domainIndex = email.indexOf("domain"); // 2. indexOf(String str)
        System.out.println("Index of 'domain': " + domainIndex);
        int lastDotIndex = email.lastIndexOf('.’); // 3. lastIndexOf(int ch)
        System.out.println("Last index of '.': " + lastDotIndex);
        int lastDomainIndex = email.lastIndexOf("domain"); // 4. lastIndexOf(String str)
        System.out.println("Last index of 'domain': " + lastDomainIndex);
        int atIndexAfter5 = email.indexOf('@', 5); // 5. indexOf(int ch, int fromIndex)
        System.out.println("Index of '@' after index 5: " + atIndexAfter5);
        int lastDotIndexBefore15 = email.lastIndexOf('.', 15); // 6. lastIndexOf(int ch, int fromIndex
        System.out.println("Last index of '.' before index 15: " + lastDotIndexBefore15);
       boolean containsDomain = email.contains("domain"); // 7. contains(CharSequence seq)
        System.out.println("Contains 'domain': " + containsDomain);
        boolean startsWithContact = email.startsWith("contact"); // 8. startsWith(String prefix)
        System.out.println("Starts with 'contact': " + startsWithContact);
       boolean endsWithCom = email.endsWith(".com");    // 9. endsWith(String suffix)
        System.out.println("Ends with '.com': " + endsWithCom);
    }
  }
```

```
email:contact@domain.com
Index of '@': 7
Index of 'domain': 8
Last index of '.': 14
Last index of 'domain': 8
Index of '@' after index 5: 7
Last index of '.' before index 15: 14
Contains 'domain': true
Starts with 'contact': true
Ends with '.com': true
```

```java
class EmailExtractor {
    public static void main(String args[]) {
        String email = "contact@example.com";
        int atIndex = email.indexOf('@');
        int dotIndex = email.lastIndexOf('.');

        char domain[] = new char[dotIndex - atIndex - 1];

        email.getChars(atIndex + 1, dotIndex, domain, 0);
        System.out.println("Domain: " + new String(domain));
    }
}
```

```
Domain: example
```

```
public class StringSort {
    public static void main(String[] args) {
         String[] cskPlayers = {"Dhoni","Ruturaj","Stokes","Rachin","Ambati"};
         for(int j = 0; j < cskPlayers.length; j++) {
              for(int i = j + 1; i <cskPlayers.length; i++) {
                  if(cskPlayers[i].compareToIgnoreCase(cskPlayers[j] )< 0) {
                      String t =cskPlayers[j];
                      cskPlayers[j] = cskPlayers[i];
                      cskPlayers[i] = t;
                  }
              }
              System.out.println(cskPlayers[j]);
          }
      }
    }
```

```
Ambati
Dhoni
Rachin
Ruturaj
Stokes
```

Character Extraction

- getBytes( )
- There is an alternative to getChars( ) that stores the characters in an array of bytes.
- This method is called getBytes( ), and it uses the default character-to-byte conversions provided by the platform.
- Here is its simplest form:
  - **byte[ ] getBytes( )**
- getBytes( ) is most useful when you are exporting a String value into an environment that does not support 16-bit Unicode characters.

Character Extraction

- toCharArray( )
- To convert all the characters in a String object into a character array, the easiest way is to call toCharArray( ).
- It returns an array of characters for the entire string.
- It has this general form:
  - **char[ ] toCharArray( )**
- This function is provided as a convenience, since it is possible to use getChars( ) to achieve the same result.

String Comparison

- **equals()** - 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.

  public boolean **equals**(Object anObject)

- **equalsIgnoreCase()-** Compares this String to another String, ignoring case considerations.

  - When it compares two strings, it considers A-Z to be the same as a-z.
  - 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.

  public boolean **equalsIgnoreCase**(String anotherString)

String Comparison

- **regionMatches( )**
- The regionMatches( ) method compares a specific region inside a string with another specific region in another string.
- There is an overloaded form that allows you to ignore case in such comparisons.
- Here are the general forms for these two methods:

```
boolean regionMatches(int startIndex,
String str, int strStartIndex, int numChars)
```

```
boolean regionMatches(boolean
ignoreCase, int startIndex, String str, int
strStartIndex, int numChars)
```

- For both versions, startIndex specifies the index at which the region begins within the invoking String object.
- The String being compared is specified by str.
- The index at which the comparison will start within str is specified by strStartIndex.
- The length of the substring being compared is passed in numChars.
- In the second version, if ignoreCase is true, the case of the characters is ignored. Otherwise, case is significant.

```
public class RegionMatchesExample {
    public static void main(String[] args) {
        String str1 = "Hello World";
        String str2 = "world";
        String str3 = "Hello";

        // Case-sensitive match
       // 'World' starting from index 5 of str1 compared to 'world'
        boolean result1 = str1.regionMatches(6, str2, 0, 5);
        System.out.println("Case-sensitive match: " + result1);

        // Case-insensitive match
        boolean result2 = str1.regionMatches(true, 6, str2, 0, 5);
     // 'World' starting from index 5 of str1 compared to 'world' ignoring case
        System.out.println("Case-insensitive match: " + result2);

        // Checking a specific region with exact length
        boolean result3 = str1.regionMatches(0, str3, 0, 5);
      // 'Hello' starting from index 0 of str1 compared to 'Hello'
        System.out.println("Exact length match: " + result3);     }
}
```

```
Case-sensitive match: false
Case-insensitive match: true
Exact length match: true
```

String Comparison

- **startsWith()** - Tests if this string starts with the specified prefix.

  public boolean **startsWith**(String prefix)

  “Figure”.startsWith(“Fig”); // true

- **endsWith()** - Tests if this string ends with the specified suffix.

  public boolean **endsWith**(String suffix)

  “Figure”.endsWith(“re”); // true

  - boolean startsWith(String str, int startIndex)
    - Example : "Foobar".startsWith("bar", 3) => returns true.

String Comparison

- **compareTo()** - Compares two strings.
  - A string is less than another if it comes before the other in dictionary order.
  - A string is greater than another if it comes after the other in dictionary order
  - The result is a negative integer if this String object lexicographically precedes the argument string.
  - The result is a positive integer if this String object lexicographically

    follows the argument string.
  - The result is zero if the strings are equal.
  - compareTo returns 0 exactly when the equals(Object) method would retu true.

public int **compareTo**(String anotherString) public int

compareToIgnoreCase(String str)

Modifying a String

- **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"

public String **substring**(int beginIndex, int endIndex)

Eg: "smiles".substring(1, 5)

returns "mile“

```java
class CharReplace {
    public static void main(String args[]) {
        String org = "Hello, World! Hello, Java!";
        char search = 'o';
        char sub = '0';
        StringBuilder result = new StringBuilder();
        int i;

        do { // replace all matching characters
            System.out.println(org);
            i = org.indexOf(search);

            if(i != -1) {
                result = new StringBuilder(org.substring(0, i));
                result.append(sub);
                result.append(org.substring(i + 1));
                org = result.toString();
            }
        } while(i != -1);
    }
}
```

```
Hello, World! Hello, Java!
Hell0, World! Hello, Java!
Hell0, W0rld! Hello, Java!
Hell0, W0rld! Hell0, Java!
Hell0, W0rld! Hell0, Java!
```

String METHODS

Method call	Meaning
S2=s1.toLowerCase()	Convert string s1 to lowercase
S2=s1.toUpperCase()	Convert string s1 to uppercase
S2=s1.repalce(„x", „y")	Replace occurrence x with y
S2=s1.trim()	Remove whitespaces at the beginning and end of the string s1
S1.equals(s2)	If s1 equals to s2 return true
S1.equalsIgnoreCase(s2)	If s1==s2 then return true with irrespective of case of charecters
S1.length()	Give length of s1
S1.CharAt(n)	Give nth character of s1 string
S1.compareTo(s2)	If s1<s2 -ve no If s1>s2 +ve no If s1==s2 then 0
S1.concat(s2)	Concatenate s1 and s2
S1.substring(n)	Give substring staring from nth character

String Operations

- **concat()** - Concatenates the specified string to the end of this string.
- 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

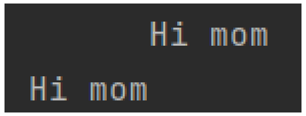
- replace()- Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
- public String **replace**(char oldChar, char newChar)
- “iam aq iqdiaq ” .replace(„q', „n’) //returns “I am an indian"

String Operations

- **trim()** - Returns a copy of the string, with leading and trailing whitespace omitted.

  public String trim()

```
String s="      Hi mom";
System.out.println(s);
System.out.println(s.trim());
```



```
        Hi mom
Hi mom
```

- **valueOf()** – Returns the string representation of the char array argument.

  public static String **valueOf**(char[] data)

String Operations

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

public String **toLowerCase**()

public String **toUpperCase**()

Eg: “HELLO YOU”.toLowerCase();

“hello you”.toUpperCase();

```
public class CharacterMethodsExample {
    public static void main(String[] args) {

        char[] characters = {'a', '1', ' ', 'A', 'b', 'D', 'z', '@', '9', '1', '_’};
            for (char ch : characters) {
            System.out.println("Character: " + ch);

            System.out.println("Is Letter: " + Character.isLetter(ch));    // isLetter(char ch)

            System.out.println("Is Digit: " + Character.isDigit(ch));      // isDigit(char ch)

          System.out.println("Is Whitespace: "+Character.isWhitespace(ch)); // isWhitespace(char ch)

          System.out.println("Is Upper Case: " + Character.isUpperCase(ch)); // isUpperCase(char ch)

           System.out.println("Is Lower Case: " + Character.isLowerCase(ch)); // isLowerCase(char ch)

         System.out.println("To Upper Case: " + Character.toUpperCase(ch)); // toUpperCase(char ch)

        System.out.println("To Lower Case: " + Character.toLowerCase(ch)); // toLowerCase(char ch)

  System.out.println("Is Letter or Digit: " + Character.isLetterOrDigit(ch));// isLetterOrDigit(char ch)
}
    }
}
```

```
public class ValidationExample {
    public static void main(String[] args) {
                String phoneNumber = "+1234567890";
                validatePhoneNumber(phoneNumber); }
}
public static void validatePhoneNumber(
String phoneNumber) {
   // Remove non-numeric characters manually
   String cleanedNumber = "";
  for (char c : phoneNumber.toCharArray()) {
         if (Character.isDigit(c)) {
        cleanedNumber += c;
     }
      }
   // Check if cleaned number has exactly 10 digits
    if (cleanedNumber.length() == 10) {
         System.out.println("Phone number is valid.");
      }
else
{
         System.out.println("Invalid phone number. It should be exactly 10 digits long.");
}
 }
```

```
public static void validatePassword(String password) {
        if (password.length() < 8) {
            System.out.println("Invalid password. It must be at least 8 characters long.");
            return;
        }
        boolean hasUpperCase = false;       boolean hasLowerCase = false;
        boolean hasDigit = false;        boolean hasSpecialChar = false;
        boolean hasSpace = false;
        for (char c : password.toCharArray()) {
            if (Character.isUpperCase(c)) {
                hasUpperCase = true;
            } else if (Character.isLowerCase(c)) {
                hasLowerCase = true;
            } else if (Character.isDigit(c)) {
                hasDigit = true;
            } else if (c == '@' || c == '#’
|| c == '$’ || c == '%' ) {
                hasSpecialChar = true;
            } else if (Character.isWhitespace(c)) {
                hasSpace = true;
                break;
            }
        }
```

```
 if (hasSpace) {
            System.out.println("Invalid password.");
        }
else if (hasUpperCase && hasLowerCase &&
hasDigit && hasSpecialChar) {
            System.out.println("Password is valid.");
        } else {
            System.out.println("Invalid password.");
        }
    }
}
}
```

Method	Description	Example
int codePointCount(int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range.	int count = exampleString.codePointCount (0, 5); // Returns 5
int codePointBefore(int index)	Returns the Unicode code point before the specified index in the string.	int codePoint = exampleString.codePointBefor e(6); // output 32(Unicode of space)
int codePointAt(int index)	Returns the Unicode code point at the specified index.	int codePoint = exampleString.codePointAt(5); // output: 32(space character)
int codePointCount(int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range	int count = exampleString.codePointCount (0, 5); // Output: 5
boolean contentEquals(CharSequence str)	Compares the content of the string with the specified str.	boolean result = exampleString.contentEquals(" Hello World"); // Output: true

Method	Description	Example
String format(Locale loc, String frmstr, Object... args)	Returns a formatted string using the specified loc, format string, and arguments.	String formatted = String.format(Locale.US, "Formatted example: %s", exampleString); //output: "Formatted example: Hello World"
boolean contains(CharSequence str)	Checks if the string contains the specified str of characters.	boolean result = exampleString.contains("World "); // Returns true
String format(String format, Object... args)	Returns a formatted string using the specified format string and arguments.	String formatted = String.format("Message: %s", "Hello World"); // Returns "Message: Hello World"
boolean isEmpty()	Checks if the string is empty (""). Returns true if the string is empty, otherwise false.	boolean result = "Hello World".isEmpty(); // Returns false

Method	Description	Example

Stream<String> lines()	Returns a stream of lines extracted from the string, split by line separators.	"Hello\nWorld".lines().forEach(System.out::println);
		// Output: "Hello"
		// Output: "World"

String replaceFirst(String regex, String replacement)	Replaces the first substring that matches the given regular expression with the specified replacement string.	String replaced = exampleString.replaceFirst("Hello", "Hi"); // Returns "Hi World, Hello Universe"
String replaceAll(String regex, String replacement)	Replaces all substrings that match the given regular expression with the specified replacement string.	String replaced = exampleString.replaceAll("Hello", "Hi"); // Returns "Hi World, Hi Universe"
String[] split(String regex)	Splits the string around matches of the given regular expression and returns an array of substrings.	String[] parts = exampleString.split(" "); // Returns ["Hello", "World,", "Hello", "Universe"]

Wrapper class

- To handle primitive data types java support it by using wrapper class.
- **java** provides the mechanism *to convert primitive into object and object into primitive*.
- **autoboxing** and **unboxing** feature converts primitive into object and object into primitive automatically.
- The automatic conversion of primitive into object is known and autoboxing and vice-versa unboxing.

Example of wrapper class

```
public class Wrapper{
public static void main(String args[]){
//Converting int into Integer
 int k=20;
Integer i=new Integer(k); //converting int into Integer
Integer j=k;//autoboxing, compiler will write Integer.valueOf(a) internally
System.out.println(k+" "+i+" "+j);
}}
```

Output:

```
20 20 20
```