Chapter - 16

Strings, I/O Operations and File Management

Java String

- Strings in Java comprise immutable sequences of Unicode characters that make them unique in many ways.
- In Java, the string is basically an object that represents a sequence of char values. An array of characters works same as Java string.
- Strings in Java are objects that are backed by character arrays.
    - If you wish to view the contents of the String in Java in the form of the character array that represents it, you can use the *toCharArray()* method of the String class.

```
public class StringToCharArrayExample {

        public static void main(String[] args) {

                String s1 = "Welcome to Java Programming";
                char[] ch = s1.toCharArray();
                int len = ch.length;
                System.out.println("Char Array length: " + len);
                System.out.println("Char Array elements: ");
                for (int i = 0; i < len; i++) {
                        System.out.print(ch[i] + " ");
                }
        }
}
```

Output:

```
Char Array length: 27
Char Array elements:
W e l c o m e   t o   J a v a   P r o g r a m m i n g
```

- To create strings in java by using these three classes
    - String class
    - StringBuffer class
    - StringBuilder class

- The Java String is immutable, which means it cannot be changed.
    - Whenever we change any string, a new instance is created.
    - For mutable strings, you can use StringBuffer and StringBuilder classes.

- **String class**
  - The *java.lang.String* class is used to create a string object.

  - There are two ways to create String object:
    - string literal
    - new keyword

- **String Literal**

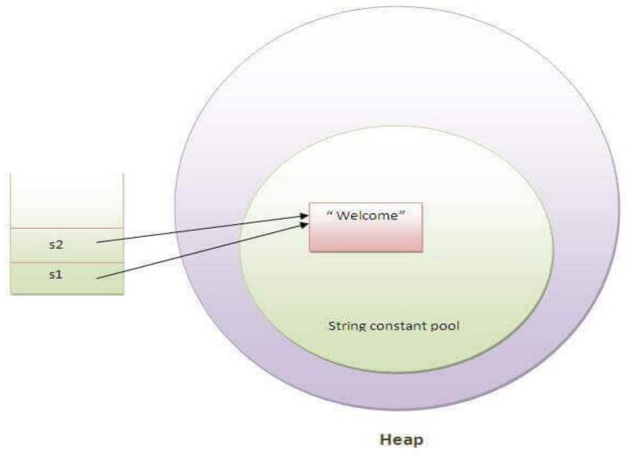
  - String objects are created using string literals are as follows:
    - String message = “Hello World! I love Java”;

  - Whenever string objects are created using *string literals* in Java, these objects are stored in the *String Constant Pool*.
    - *String constant pool* is a special storage space in the heap memory.

  - If we create an object using ***String*** literal syntax, it may return an existing object from the String pool, if it already exists. Otherwise, it will create a new String object and put in the string pool for future re-use.
  - The following example, two *String* objects created, but these two *string* objects will have the same reference.

    - String s1="Welcome"; *// the JVM checks the "string constant pool" first. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.*

    - String s2="Welcome" *//It doesn't create a new instance because the string already exists in the pool, a reference to the pooled instance is returned.*



- **New Keyword**
  - String objects are created using *new keyword* are as follows:
    - One-way:

      String message = new String (“Hello World! I love Java”);
    - Another-way:

      char[] javaArray = { ‘H’, ‘E ‘, ‘L’, ‘L’, ‘O’ };
      String javaString = new String(javaArray);
  - When string objects are created using the *new keyword*, it always creates a new object in heap memory.
    - Here s1 and s2 are two different objects and they have different references
      - String s1 = new String("Welcome");
      - String s2 = new String("Welcome");

Example – 1: Create String by using String literal and new keyword.

```
public class StringExample{

    public static void main(String args[]){

        //creating string by java string literal
        String s1="java";

        char ch[]={'s','t','r','i','n','g','s'};  //character array

        //converting char array to string
        String s2=new String(ch);

        //creating java string by new keyword
        String s3=new String("example");

        //output
        System.out.println(“1st String :  ” + s1);
        System.out.println(“2nd String :  ” + s2);
        System.out.println(“3rd String :  ” + s3);
    }
}
```

Output:

```
1st String :  java
2nd String :  strings
3rd String :  example
```

Example:

```
public class StringObjectComparison {
        public static void main(String[] args) {
                // The following strings are being created using literals
                String literal1 = "xyz";
                String literal2 = "xyz";
                System.out.print("Comparison using == operator for literals : ");
                // The output of this line of code will be true
                System.out.println(literal1 == literal2);

                // The following two strings are being created using the new operator
                String keyword1 = new String("abc");
                String keyword2 = new String("abc");
                System.out.print("Comparison using == operator for objects : ");
                // The output of this line of code will be false
                System.out.println(keyword1 == keyword2);
        }
}
```

Output:

```
Comparison using == operator for literals : true
Comparison using == operator for objects : false
```

String Length

The length of a string can be found by the length() method.

Example: Find the length of the String.

```
public class Length {

     public static void main(String[] args) {

          String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

          System.out.println("The length of the text string is: " + txt.length());
     }
}
```

Output:

```
The length of the text string is: 26
```

To Upper and To Lower

Make the string to upper or lower case.

Example:

```
public class UpperLower {

        public static void main(String[] args) {

                String txt = "Hello World";

                System.out.println("The upper case is " + txt.toUpperCase());

                System.out.println("The lower case is " + txt.toLowerCase());
        }
}
```

Output:

```
        The upper case is HELLO WORLD
        The lower case is hello world
```

Finding Index of a Character in a String

- The indexOf() method returns the **index** (the position) of the first occurrence of a specified text in a string (including whitespace).

Example: Find the index of "locate" in a string "Please locate where 'locate' occurs!", by using indexOf() method.

```
public class FindCharIndex {

          public static void main(String[] args) {

                    String txt = "Please locate where 'locate' occurs!";
                    int i= txt.indexOf("locate");

                    //int i= txt.indexOf('e');
                    System.out.println("The index value is " + i);
          }
}
```

Output:

```
        The index value is 7
```

Processing a String One Character at a Time

- A string's **charAt()** method retrieves a given character by index number (starting at zero) from within the String object.

- To process all the characters in a String, one after another, use a for loop ranging from zero to String.length()-1.

Example : Retrieves a given character by index number

```
public class CharAtExample{

    public static void main(String args[]){

        String name="Hello";

        char ch=name.charAt(4);   //returns the char value at the 4th index
        System.out.println("Character at index 4 is "+ ch);
    }
}
```

Output:

```
        Character at index 4 is o
```

Example: Process all the characters in a String:

```
public class StrCharAt {

        public static void main(String[] av) {

                String a = "A quick bronze fox kept a lazy bovine";
                for (int i=0; i < a.length(); i++)                    // Don't use for each
                        System.out.println("Char " + i + " is " + a.charAt(i));
        }
}
```

Output:

```
        Char 0 is A
        Char 1 is
        Char 2 is q
        Char 3 is u
        Char 4 is i
        Char 5 is c
        Char 6 is k
        Char 7 is
        Char 8 is b
        Char 9 is r
        Char 10 is o
        Char 11 is n
```

```
Char 12 is z
Char 13 is e
Char 14 is
Char 15 is f
Char 16 is o
Char 17 is x
Char 18 is
Char 19 is l
Char 20 is e
Char 21 is p
Char 22 is t
Char 23 is
Char 24 is a
Char 25 is
Char 26 is l
Char 27 is a
Char 28 is z
Char 29 is y
Char 30 is
Char 31 is b
Char 32 is o
Char 33 is v
Char 34 is i
Char 35 is n
Char 36 is e
```

Example: process all the characters in a String using toCharArray() method.

```
public class ForEachChar {

	public static void main(String[] args) {

		String str = "Hello world";

		// toCharArray() returns an Array of chars after
		//converting a String into sequence of characters.
		char[] array= str.toCharArray();

		// for (char ch : s) {...} Does not work, in Java 7

		for (char ch : array) {
			System.out.println(ch);
		}
	}
}
```

Output:

```
H
e
l
l
o

w
o
r
l
d
```

Java String equals() Method

- The **java string equals()** method compares the two given strings based on the content of the string.
- If any character is not matched, it returns false. If all characters are matched, it returns true.

Example:

```
public class StringEquals {

        public static void main(String[] args) {

                String s1="java";
                String s2="java";
                String s3="JAVA";
                String s4="python";

                System.out.println(s1==s2);        //true because content and case is same

                System.out.println(s1.equals(s2));      //true because content and case is same

                System.out.println(s1.equals(s3));      //false because case is not same

                System.out.println(s1.equals(s4));     //false because content is not same
        }
}
```

Output:

```
        true
```

```
true
false
false
```

Java String equalsIgnoreCase() Method

- The **String equalsIgnoreCase()** method compares the two given strings on the basis of content of the string irrespective of case of the string.
- It is like equals() method but doesn't check case.
- If any character is not matched, it returns false otherwise it returns true.

Example:

```
public class StringEquals {

        public static void main(String[] args) {

              String s1="java";
              String s2="java";
              String s3="JAVA";
              String s4="python";

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

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

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

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

         }
}
```

Output:

```
true
true
true
false
```

String Concatenation in Java

- In Java, String concatenation forms a new String that is the combination of multiple strings.

- There are two ways to concatenate strings in Java:
    1. **+** (String concatenation) operator
    2. **concat()** method

1) String Concatenation by + (String concatenation) operator

- Java String concatenation operator (+) is used to add strings.
- **For Example:**

```
class TestStringConcatenation1{
        public static void main(String args[]){

                //add the two string objects
                String s1 = "Hello";
                String s2 = "  JAVA";
                String s =  s1 + s2;
                System.out.println(s);

                //add the contents of two strings
                String str="Welcome To" + " JAVA World";
                System.out.println(str);

                //create a string literal that spans multiple lines
                String PopularQuote = "what's your opinion about An eye for an eye" +
                                " will make the whole world blind.";
                System.out.println(PopularQuote);

                //Concatenating string with variable
                double weight = 50.0;
                System.out.println("My weight is " + weight);
        }
}
```

Output:

```
Hello  JAVA
Welcome To JAVA World
what's your oppinion about An eye for an eye will make the whole world blind.
My weight is 50.0
```

2) String Concatenation by concat() method

- The String concat() method concatenates the specified string to the end of current string.
- **Syntax:** public String concat(String)
- **Example**: String concat() method.

```
class TestStringConcatenation3{

    public static void main(String args[]){

        //concate two string objects
        String s1="Hello ";
        String s2=" World";
        String s3=s1.concat(s2);
        System.out.println(s3);

        //concate string object with string
        String s4 = "My Favourite Programming Language ";
        String s5 = s4.concat("is Java");
        System.out.println(s5);

        //concate string with string object
        String s6 = "Luck";
        String s7 = "Best of ".concat(s6);
        System.out.println(s7);

        //concate string with string
        String s8 = "I Love ".concat(" Java");
        System.out.println(s8);
    }
}
```

Output:

```
    Hello  World
    My Favourite Programming Language is Java
    Best of Luck
    I Love  Java
```

Taking Strings Apart with Substrings

- **substring() method** returns a part of the string.

- There are two types of substring methods in java string.

    - *substring(int startIndex)*
        - return all the characters from a given index (startIndex) to end of the string.

    - *substring(int startIndex, int endIndex)*
        - return all the characters between the starting index (startIndex) to ending index (endIndex) of the string.

Example:

```
public class SubStringDemo {

    public static void main(String[] av) {

        String a = "Java is a great Language.";
        System.out.println(a);                   //Java is a great Language.

        String b = a.substring(5);               //b is the String "is a great Language."
        System.out.println("The b substring is : " + b);

        String c = a.substring(5, 7);            //c is the String "is"
        System.out.println("The c substring is : " + c);

        String d = a.substring(10, a.length());  //d is the String "great Language."
        System.out.println("The d substring is : " + d);
    }
}
```

Output:

```
Java is a great Language.
The b substring is : is a great Language.
The c substring is : is
The d substring is : great Language.
```

Example: Write a program which reads two strings from the user and concat the substring from 0 to 8 index of the first string with the 0 to 5 indexed of the second string and print the resultant string. The two strings are as follows:

ITER is my College.
ITER is in SOA.

```
import java.util.*;
import java.util.Scanner;

public class StringDemo {

    public static void main(String[] args) {

        Scanner S = new Scanner(System.in);
```

```
                System.out.println("Enter the 1st string ");
                String s1 = S.nextLine();
                System.out.println("The 1st string: " + s1);

                System.out.println("Enter the 2nd string ");
                String s2 = S.nextLine();
                System.out.println("The 2nd string: " + s2);

                String a = s1.substring(0,8);
                System.out.println("The 1st substring is: " + a);

                String b = s2.substring(0,5);
                System.out.println("The 2nd substring is: " + b);


                // concat() method concatenates one string
                //to the end of another string.
                a=a.concat(b);
                System.out.println("The resultant string is: " + "  " + a);
        }
}
```

Output:

```
        Enter the 1st string
        ITER is my College.
        The 1st string: ITER is my College.
        Enter the 2nd string
        ITER is in SOA.
        The 2nd string: ITER is in SOA
        The 1st substring is: ITER is
        The 2nd substring is: ITER
        The resultant string is: ITER is ITER
```

Breaking/Splitting Strings Into Words

- Take a string and divide the string into words or tokens.
- In Java, the split() method, splits or divides the input string into multiple parts based on the regular expression that is entered as a parameter.
    - The split() method belong to the String class.
    - Regular expressions are " ", ",", "-", ";",".", "?", "@", ":"
- Two **Syntax for Split method**:
    - split(String regex)
    - split (String regex, int limit)
        - The limit value can be *positive*, *negative* or *zero*.

        - **if limit is Positive value:** the pattern will be repeated a maximum of limit –1 times.
        - **if limit is Negative value:** the pattern will be repeated as many times as possible.
        - **if limit is Zero:** the pattern will be repeated as many times as possible.
    - The result of split() method operation is an array of strings.

- **Example:** Java program to demonstrate working of split() method.

```
import java.util.*;
import java.util.Scanner;

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

                Scanner S = new Scanner(System.in);
                System.out.println("Enter the string ");
                String str = S.nextLine();

                System.out.println("The Entered String: " + str);

                String[] arr = str.split(“ ”);

                for (String word : arr) {
                        System.out.println(word);
                }
        }
}
```

Output:

```
Enter the string
I Love java
The Entered String: I Love java
I
Love
Java
```

- **Example:** Java program to demonstrate working of split(regex, limit) with small limit.

```
public class StringSplitWithLimitExample {

         // Main driver method
        public static void main(String args[])   {
                // Custom input string
                String str = "I@Love@JAVA";
                String[] arrOfStr = str.split("@", 2);

                for (String a : arrOfStr)
                        System.out.println(a);
        }
}
```

Output:

```
        I
        Love@JAVA
```

- **Example:** Java program to demonstrate working of split(regex) with multiple different characters.

```
public class StringSplitOnMultipleCharactersExample {

        public static void main(String args[]) {

                String myStr = "My, Favourite @Programming?Language.Java";
                String[] arrOfStr = myStr.split("[, ?.@]+");

                for (String piece : arrOfStr) {
                        System.out.println(piece);
                }
        }
}
```

Output:

```
        My
        Favourite
        Programming
        Language
        Java
```

Java StringBuffer class

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

- The StringBuffer class is Synchronized .
    - Java StringBuffer class is thread-safe, i.e., multiple threads cannot access it simultaneously. So it is safe and will result in an order.
- Important methods of StringBuffer class are
    1) StringBuffer append() method
    2) StringBuffer insert() method
    3) StringBuffer replace() method
    4) StringBuffer delete() method
    5) StringBuffer reverse() method

1) StringBuffer append() method

- The append() method concatenates the given argument with this string.
- **Example:**

```
class StringBufferExample{

        public static void main(String args[]){

            StringBuffer sb=new StringBuffer("Hello ");
            System.out.println(sb);            //prints Hello
            sb.append("Java ");                //now original string is changed
            sb.append("is language");
            System.out.println(sb);            //prints Hello Java  is language
        }
}
```

Output:

```
        Hello Java is language
```

2) StringBuffer insert() method

- The insert() method inserts the given string with this string at the given position.
- The syntax is :
    - insert(int index, String string)
- **Example:**

```
class StringBufferExample2{

    public static void main(String args[]){
            StringBuffer sb=new StringBuffer("Hello ");
            sb.insert(1,"Java");        //now original string is changed
            System.out.println(sb);     //prints HJavaello
        }
}
```

Output:

```
HJavaello
```

3) StringBuffer replace() method

- The replace() method replaces the given string from the specified beginIndex and endIndex.
- The syntax is :
    - replace(int beginIndex, int endIndex, string String)
- **Example:**

```
class StringBufferExample3{

    public static void main(String args[]){

        StringBuffer sb=new StringBuffer("Hello");
        sb.replace(1, 3, "Java");
        System.out.println(sb);                    //prints HJavalo
    }
}
```

Output:

```
HJavalo
```

4) StringBuffer delete() method

- The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.
- The syntax is :
    - delete(int beginIndex, int endIndex)
- **Example:**

```
class StringBufferExample4{

    public static void main(String args[]){

        StringBuffer sb=new StringBuffer("Hello");
        sb.delete(1,3);
        System.out.println(sb);    //prints Hlo
    }
}
```

Output:

```
Hlo
```

5) StringBuffer reverse() method

- The reverse() method of StringBuilder class reverses the current string.
- **Example:**

```
class StringBufferExample5{

    public static void main(String args[]){

            StringBuffer sb=new StringBuffer("Hello");
            sb.reverse();
            System.out.println(sb);     //prints olleH
    }
}
```

Output:

```
olleH
```

Java StringBuilder class

- The Java StringBuilder class is used to create mutable (modifiable) string.
- The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized.
- It has been available since JDK 1.5.
- Important methods of StringBuilder class are
    1) StringBuilder append() method
    2) StringBuilder insert() method
    3) StringBuilder replace() method
    4) StringBuilder delete() method
    5) StringBuilder reverse() method

1) StringBuilder append() method

- The StringBuilder append() method concatenates the given argument with this string.
- **Example:**

```
class StringBuilderExample{

        public static void main(String args[]){

            StringBuilder sb=new StringBuilder("Hello ");
            sb.append("Java");//now original string is changed
            System.out.println(sb);//prints Hello Java
        }
}
```

Output:

```
Hello Java
```

Example:

```
public class StringBuilderDemo {

    public static void main(String[] args) {

        //using + operator
        String s1 = "Hello" + ", " + "World";
        System.out.println(s1);

        // Build a StringBuilder, and append some things to it.
        StringBuilder sb2 = new StringBuilder();
        sb2.append("Hello");
        sb2.append(',');
        sb2.append(' ');
        sb2.append("World");

        // Get the StringBuilder's value as a String, and print it.
        String s2 = sb2.toString();
        System.out.println(s2);

        // Now do the above all over again, but in a more
        // concise (and typical "real-world" Java) fashion.
        System.out.println(
        new StringBuilder()
        .append("Hello")
        .append(',')
        .append(' ')
        .append("World"));
    }
}
```

Output:

```
Hello, World
Hello, World
Hello, World
```

2) StringBuilder insert() method

- The StringBuilder insert() method inserts the given string with this string at the given position.
- The syntax is :
    - insert(int index, String string)
- **Example:**

```
class StringBuilderExample2{

    public static void main(String args[]){

            StringBuilder sb=new StringBuilder("Hello ");
            sb.insert(1,"Java");              //now original string is changed
            System.out.println(sb);           //prints HJavaello
    }
}
```

Output:

```
        HJavaello
```

3) StringBuilder replace() method

- The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.
- The syntax is :
    - replace(int beginIndex, int endIndex, string String)
- **Example:**
-

```
class StringBuilderExample3{

        public static void main(String args[]){

            StringBuilder sb=new StringBuilder("Hello");
            sb.replace(1,3,"Java");
            System.out.println(sb);    //prints HJavalo
        }
}
```

Output:

```
        HJavalo
```

4) StringBuilder delete() method

- The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.
- **Example:**

```
class StringBuilderExample4{

        public static void main(String args[]){

            StringBuilder sb=new StringBuilder("Hello");
            sb.delete(1,3);
            System.out.println(sb);    //prints Hlo
        }
}
```

Output:

```
Hlo
```

5) StringBuilder reverse() method

- The reverse() method of StringBuilder class reverses the current string.
- The syntax is :
    - delete(int beginIndex, int endIndex)
- **Example:**

```
class StringBuilderExample5{

        public static void main(String args[]){

            StringBuilder sb=new StringBuilder("Hello");
            sb.reverse();
            System.out.println(sb);//prints olleH
        }
}
```

Output:

```
olleH
```

Reversing a String by Character

Example: reverse a string by character easily, using charAt() method.

```
import java.util.*;

public class ReverseString {

        public static void main(String[] args) {

                String original, reverse = "";
                Scanner sc = new Scanner(System.in);

                System.out.println("Enter a string to reverse");
                original = sc.nextLine();

                int length = original.length();

                for (int i = length - 1 ; i >= 0 ; i--)
                  reverse = reverse + original.charAt(i);
                System.out.println("Reverse of the string: " + reverse);
        }
}
```

Output:

```
        Enter a string to reverse
        ITER is My College.
        Reverse of the string: .egelloC yM si RETI
```

Example: To reverse the characters in a string, using the StringBuilder reverse() method.

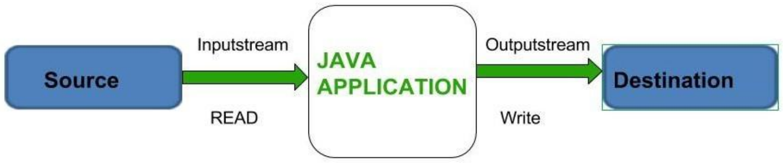
```
import java.util.*;

public class ReverseString {

        public static void main(String[] args) {

                String original, reverse = "";

                Scanner sc = new Scanner(System.in);
                System.out.println("Enter a string to reverse");
                original = sc.nextLine();

               StringBuilder sb=new StringBuilder(original);
               System.out.println("Reverse of the string: " + sb.reverse());
        }
}
```

Output:

```
        Enter a string to reverse
        ITER is My College.
        Reverse of the string: .egelloC yM si RETI
```

Input and Output

- Java brings various Streams with its I/O package that helps the user to perform all the input-output operations.
- These streams support all the types of objects, data-types, characters, files etc to fully execute the I/O operations.



Standard or Default Streams

- Java provides **3** most common **standard or default streams,** that are:
    - System.in
    - System.out
    - System.err

- **System.in**: This is the standard input stream that is used to read characters from the keyboard or any other standard input device.

- **System.out**: This is the standard output stream that is used to produce the result of a program on an output device like the computer screen.

    - Here is a list of the various print functions that we use to output statements:

        - **print():** This method in Java is used to display a text on the console. This text is passed as the parameter to this method in the form of String. This method prints the text on the console and the cursor remains at the end of the text at the console. The next printing takes place from just here.

        - **println():** This method in Java is also used to display a text on the console. It prints the text on the console and the cursor moves to the start of the next line at the console. The next printing takes place from the next line.

        - **printf():** This is the easiest of all methods as this is similar to printf in C.

            - Note that System.out.print() and System.out.println() take a single argument, but printf() may take multiple arguments. This is used to format the output in Java.

- **System.err:** This is the standard error stream that is used to output all the error data that a program might throw, on a computer screen or any standard output device. This stream also uses all the 3 above-mentioned functions to output the error data:

    - print()
    - println()
    - printf()

Example: The following codes shows the input and output stream

```
import java.util.Scanner;

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

                // Following code will create scannerObj object of Scanner class
                Scanner scannerObj = new Scanner(System.in);

                System.out.println("Enter the name of the student");
                // Below line of code ensures that data will be input as string by default
                String studentNAME = scannerObj.next();

                System.out.println("Enter the roll number of the student");
                // Below line of code ensures that data will be input as int by default
                int studentRollNumber = scannerObj.nextInt();

                System.out.println("Enter the marks that the student obtained");
                // Below line of code ensures that data will be input as float by default
                float studentMarks = scannerObj.nextFloat();

                System.out.println("-------Student Report Card-------");
                System.out.println("Student Name:" + studentNAME);
                System.out.println("Student Roll No.:" + studentRollNumber);
                System.out.println("Student Marks:" + studentMarks);
                // Following code is needed to avoid resource leak
                scannerObj.close();
        }
}
```

Output:

```
        Enter the name of the student
        ABC
        Enter the roll number of the student
        12345
        Enter the marks that the student obtained
        100
        -------Student Report Card-------
        Student Name: ABC
        Student Roll No.: 12345
        Student Marks: 100
```

Example: The following codes shows how the System.err stream can be used.

```
    public class Main {

            public static void main(String[ ] args) {
                    try {
                              int[] myNumbers = {1, 2, 3};
                              System.out.println(myNumbers[1]);
                              System.out.println(myNumbers[10]);
                    }

                    catch (Exception e) {
                              System.err.println("Something went wrong.");
                    }
          }
}
```

Output:

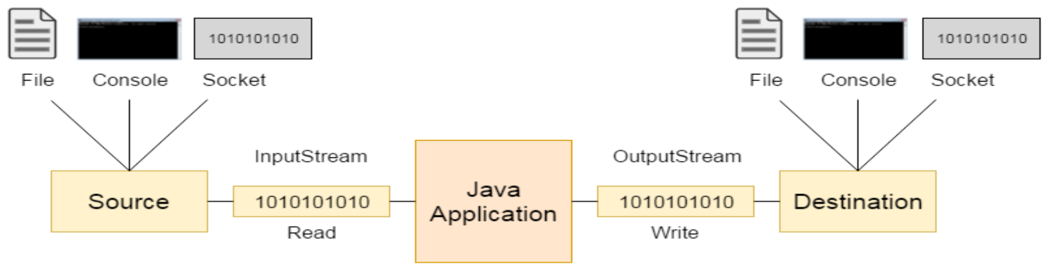
```
          2
          Something went wrong.
```

Difference Between System.out.println() and System.err.println()

System.out.println()	**System.err.println()**
System.out.println() will print to the standard out of the system.	System.err.println() will print to the standard error.
System.out.println() is mostly used to display results on the console.	System.err.println(is mostly used to output error texts.
It gives output on the console with the default(black) color.	It also gives output on the console but most of the IDEs give it a red color to differentiate.

InputStream and OutputStream Class

- The working of Java OutputStream and InputStream in the figure given below



- **InputStream Class**
  - Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.
  - InputStream class is an abstract class.
  - It is the superclass of all classes representing an input stream of bytes.
  - *Useful methods of InputStream*

Method	Description
1) public abstract int read() throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of the file.
2) public int available() throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close() throws IOException	is used to close the current input stream.

- **OutputStream Class**
  - Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.
  - OutputStream class is an abstract class.
  - It is the superclass of all classes representing an output stream of bytes.
  - An output stream accepts output bytes and sends them to some sink.
  - Useful methods of OutputStream

Method	Description
1) public void write(int) throws IOException	is used to write a byte to the current output stream.
2) public void write(byte[]) throws IOException	is used to write an array of byte to the current output stream.
3) public void flush() throws IOException	flushes the current output stream.
4) public void close() throws IOException	is used to close the current output stream.

File Management in Java

- A named location used to store related information is known as a **File**.

- File Handling is an integral part of any programming language as file handling enables us to store the output of any particular program in a file and allows us to perform certain operations on it.
- In Java, a **File** is an abstract data type.
- The File class from the java.io package, allows us to work with files.
- In simple words, file handling means reading and writing data to a file.

  The FileReader and FileWriter classes are of immense importance when it comes to file management in Java language.

- **FileWriter class**
  - Inheriting the OutputStream class, the FileWriter class is used for the creation of files by writing characters.
  - Some of the constructors of the FileWriter class are as follows:
    - **FileWriter(File,File):** As the name suggests, this constructor creates a FileWriter object when a File object is input as a parameter.
    - **FileWriter(String filename):** Since the parameter for this constructor is a String, the constructor will create a FileWriter object whenever a file name is input as a parameter.
  - Some of the methods that are available through the FileWriter class:
    - **public void write (int c):** This method is used to write a single character into the stream that is being created.
    - **public void write(char[] stir):** The character array that needs to be inserted to the output stream will be input into this method as a parameter.

- **File Reader Class**
  - Inheriting the InputStreamReader class, the FileReader class is used to read data from any file – one character at a time.
  - It is important to note that the FileReader class can only be used to read data in the form of characters, while the FileInputStream class is used to read data in the form of raw bytes.
  - Some of the constructors of the FileReader class:
    - **FileReader(File,File):** As evident from the parameters of this constructor, a FileReader object is created when a File object is inserted as a parameter.
    - **FileReader(String filename):** This constructor creates a FileReader object given that the name of the file that needs to be read from is input as a parameter.
  - Some of the methods that are available through the FileReader class:
    - **public int read():** This method is used to read a single character from the stream that is available.
    - **public int read(char[] cbuff):** This method is used to read characters into an array.

Example: Create file and write content on it.

```
import java.io.File;
import java.io.FileWriter;

public class FileHandlingQ1 {

        public static void main(String[] args) {

                // create a file object for the current location
                File file = new File("D:\\ExampleFile.java");

                try {

                        // create a new file with name specified by the file object
                        boolean value = file.createNewFile();
                        if (value) {
                                System.out.println("New Java File " + file.getName() + " is created
                                                                                successfully.");
                        }
                }
                catch(Exception e) {
                        e.getStackTrace();
                        System.out.println("An error occurred.");
                }

                //Create content
                String data = "Date: 10/04/2024, The store related information is referred to as a
                                                                                        File.";
                //length of the file
                long length = file.length();

                try {
                        if (length == 0) {
                                // Creates a Writer object using FileWriter
                                FileWriter myWriter = new FileWriter("D:\\ExampleFile.java");

                                // Writes content to the file
                                myWriter.write(data);
                                System.out.println("Data is written successfully into the " +
                                                file.getName() + " file." );

                        // Closes the writer
                        myWriter.close();
                    }
                    else {
                        System.out.println("The file " + file.getName() + " is already
                                                exists. Hence avoid overwriting.");
                    }
                }
                catch (Exception e) {
                    e.getStackTrace();
                    System.out.println("An error occurred.");
                }
            }
}
```

Output after 1st run:

```
    New Java File ExampleFile.java is created successfully.
    Data is written successfully into the ExampleFile.java file.
```

Output after 2nd run:

```
    The file ExampleFile.java is already exists. Hence avoid overwriting.
```

Example: displays file content on the console using BufferedInputStream.

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;

public class FileHandlingQ2 {
    public static void main(String[] args) {

        try {
            // Creates a FileInputStream
            FileInputStream file = new FileInputStream("D:\\ExampleFile.txt");

            // Creates a BufferedInputStream
            BufferedInputStream input = new BufferedInputStream(file);

            // Reads first byte from file
```

```
                    int i = input.read();

                    while (i != -1) {
                              System.out.print((char) i);

                              // Reads next byte from the file
                              i = input.read();
                    }
                    input.close();
          }

          catch (Exception e) {
                    e.getStackTrace();
                    System.out.println("An error occurred.");
          }
     }
}
```

Output:

```
     Date: 10/04/2024, The store related information is referred to as a File.
```

Example: displays file content on the console using BufferedReader.

```
import java.io.BufferedReader;
import java.io.FileReader;

public class FileHandlingQ2a {

     public static void main(String[] args) {

          // Creates an array of character
          char[] array = new char[500];

          try {
                    // Creates a FileReader
                    FileReader file = new FileReader("D:\\ExampleFile.txt");

                    // Creates a BufferedReader
                    BufferedReader input = new BufferedReader(file);

                    // Reads characters, array represents the destination buffer
                    input.read(array);
                    System.out.println("Data in the file: ");
                    System.out.println(array);
```

```
                // Closes the reader
                input.close();
            }

            catch(Exception e) {
                e.getStackTrace();
            }
        }
}
```

Output:

```
        Date: 10/04/2024, The store related information is referred to as a File.
```

Example: displays file content on the console using character Scanner

```
import java.io.File;
import java.util.Scanner;

public class FileHandlingQ2b {

        public static void main(String[] args) {
            try {
                // create a new file object
                File file = new File("D:\\ExampleFile.txt");

                // create an object of Scanner
                // associated with the file
                Scanner sc = new Scanner(file);

                // read each line from file and print it
                System.out.println("Reading File Using Scanner:");
                while(sc.hasNextLine()) {
                    System.out.println(sc.nextLine());
                }

                // close scanner
                sc.close();

            } catch (Exception e) {
                e.getStackTrace();
            }
        }
}
```

Output:

```
Date: 10/04/2024, The store related information is referred to as a File.
```

Example: Append Content to an Existing File

```
import java.io.FileWriter;
import java.io.IOException;

public class FileHandlingQ3 {

	public static void main(String[] args) {
		String data = "\nDate: 11/04/2024, The new text is added in the File.";

		try {
			FileWriter fw = new FileWriter("D:\\ExampleFile.txt", true);
			fw.write(data);
			fw.close();
		}
		catch(IOException e) {
			e.getStackTrace();
			System.out.println("An error occurred.");
		}
	}
}
```

Output:

```
Date: 10/04/2024, The store related information is referred to as a File.
Date: 11/04/2024, The new text is added in the File.
```

Example: Find the List of all files and directories

```
import java.io.File;

public class FileHandling4a {

	public static void main(String[] args) {

		//Creating a File object for directory
		File directoryPath = new File("D:\\SampleDirectory1");

		//List of all files and directories
		String contents[] = directoryPath.list();
		System.out.println("List of files and directories in the specified directory:");
```

```
                for(int i=0; i<contents.length; i++) {
                        System.out.println(contents[i]);
                }

                System.out.println(" ");

                //List of all files and directories
                File filesList[] = directoryPath.listFiles();
                System.out.println("List of files and directories in the specified directory:");

                for(File file : filesList) {
                        System.out.println("File name: "+file.getName());
                        System.out.println("File path: "+file.getAbsolutePath());
                        System.out.println(" ");
                }
        }
}
```

Output:

```
        List of files and directories in the specified directory:
        Directory1
        Directory2
        ExampleFile.txt
        inputFile.txt

        List of files and directories in the specified directory:
        File name: Directory1
        File path: D:\SampleDirectory1\Directory1

        File name: Directory2
        File path: D:\SampleDirectory1\Directory2

        File name: ExampleFile.txt
        File path: D:\SampleDirectory1\ExampleFile.txt

        File name: inputFile.txt
        File path: D:\SampleDirectory1\inputFile.txt
```