# Java Review 1
# Xia Book, w3schools

Spring, 2022

## Strengths of Using Java

- Object-Oriented Paradigm - Inheritance, Polymorphism, Abstraction, Encapsulation
- Extensible and Reusable
- Statically Typed
- Compile and Interpret
- Architecture Neutral    (Platform-Independent)
- Security (No Pointer, Virtual sand-box) and JVM
- Web Server Programming with Servlet and JSP
- Database Connection
- Portable (Java byte code)
- Multi-Threaded and Distributed Capability (RMI)
- Simple (WO pointer, Operator Overloading, automatic garbage collection)

## Weakness of Java?

- No Independent Function and No Functional Programming
- No System Programming (as in C)

- Translation Speed

JVM and Interpreter – interpret and run

JIT Compiler

Java Chip

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
  `byte` -> `short` -> `char` -> `int` -> `long` -> `float` -> `double`

- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
  `double` -> `float` -> `long` -> `int` -> `char` -> `short` -> `byte`

## Widening Casting

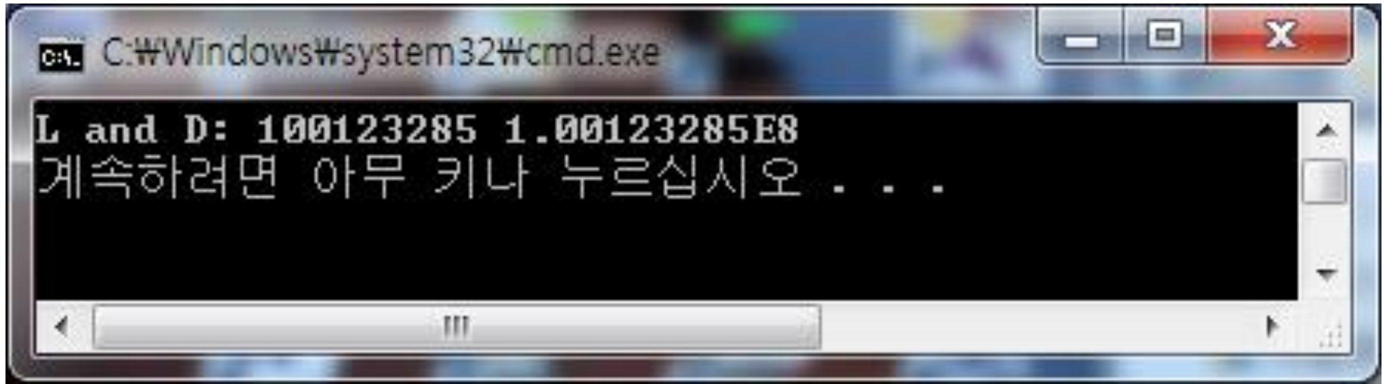
Widening casting is done automatically when passing a smaller size type to a larger size type:

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

## Automatic conversion for widening

```
class LtoD {
  public static void main(String args[]) {
    long L;
    double D;
    L = 100123285L;
    D = L;
    System.out.println("L and D: " + L + " " + D);
  }
}
```



```
L and D: 100123285 1.00123285E8
계속하려면 아무 키나 누르십시오 . . .
```

## Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

```java
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

# Java Character Type

- Internationalization
  - **16-bit Unicode** (standard 2.0 in 1996)
  - ASCII is a **subset of Unicode** --- ISO-8859 (**Latin-1**) – the first 128 characters **of Unicode**
  - **Escape sequence** (**for special character literals**):
    - `\uhhhh`: hex-decimal code, e.g. `\u000A`
    - `\ddd`: octal code, e.g. `\040`
    - `\n, \t, \b, \r, \f, \\, \', \".`

    (\u000a, \u0009, \u0008, \u000D, \u000c, …)

    `\377 (\u00ff) is the max. for octal code of char.`
- Java programs are also in Unicode.
- Unicode standard: http://www.unicode.org

Entity Names Allowed?
_name, $name, 이름, 漢字語, λμξορτφ, na?me

# Java Arrays

- Arrays are objects.
- Arrays are always bound-checked. (***dynamic** semantics*, *not **static** semantics*)
- Array index starts from 0. //Rainfall Example of Textbook

**Static type checking**
**Dynamic type checking**

```
int[] ia = new int[3];
int ia[] = new int[3];
int[] ia = { 1, 2, 3};

float[][] mat = new float[4][4];

for (int y = 0; y < mat.length; y++) {
  for (int x = 0; x < mat[y].length; x++)
    mat[y][x] = 0.0;
}
```

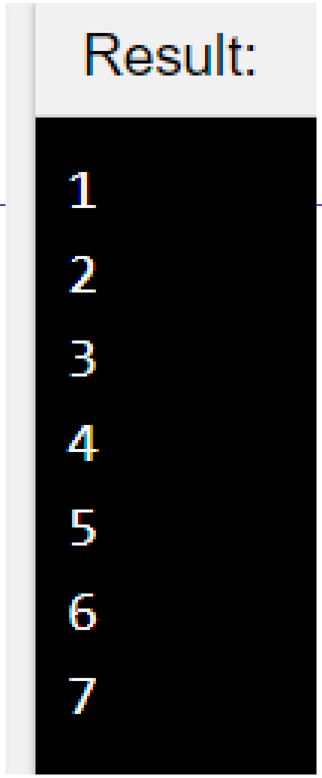
```
int[] myNum = {10, 20, 30, 40};

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel";
System.out.println(cars[0]);
// Now outputs Opel instead of Volvo

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length); // Outputs 4

//Loop Through an Array
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++) {
		System.out.println(cars[i]);
 }
//Loop Through an Array with For-Each
for (String i : cars) {
		System.out.println(i);
}
```

```
public class MyClass {
        public static void main(String[] args) {
                int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
                for (int i = 0; i < myNumbers.length; ++i) {
                        for(int j = 0; j < myNumbers[i].length; ++j) {
                                System.out.println(myNumbers[i][j]);
                        }
                }
        }
}
```



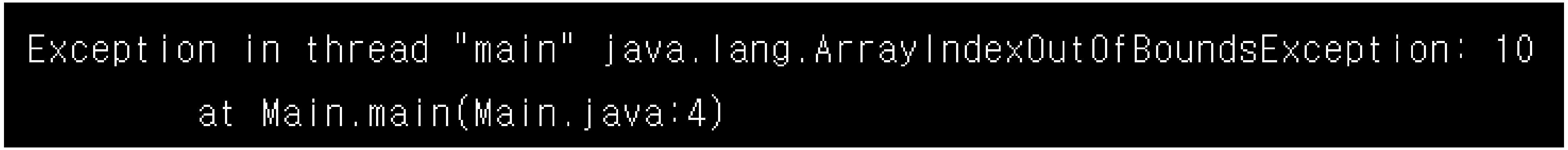
Result:

```
1
2
3
4
5
6
7
```

## Array Bound Checking

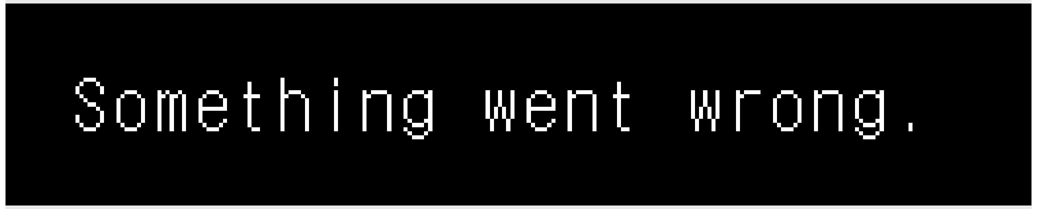
```
int b[] = new int[1000];
try {
   // computation with b[];
}
catch (ArrayIndexOutOfBoundsException e) {
  // handle the exception
}
```

```java
public class Main {
  public static void main(String[ ] args) {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]); // error!
  }
}
```



```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at Main.main(Main.java:4)
```

```java
public class Main {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
  }
}
```



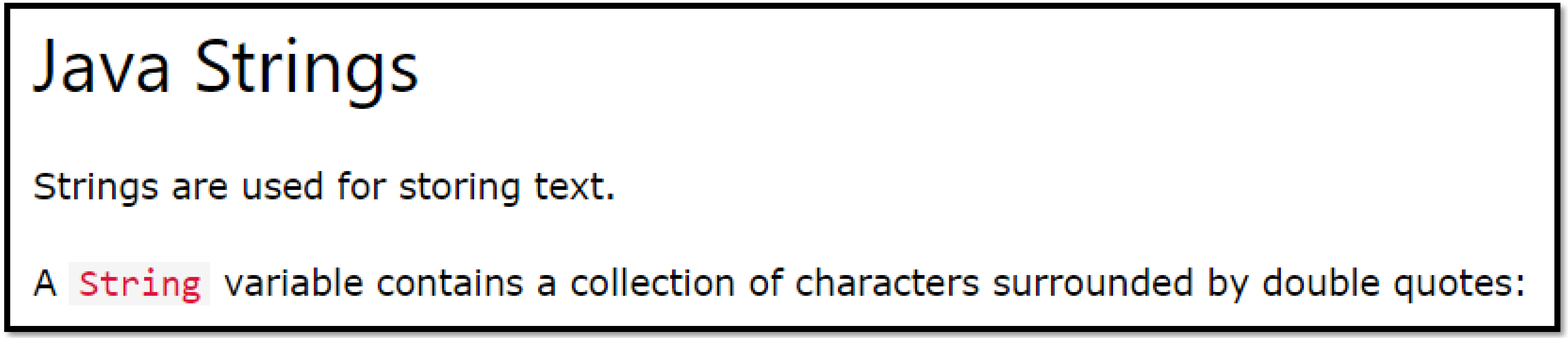
```
Something went wrong.
```

# Java String

- Strings are objects. The String type is a class.
- Strings are ***not* arrays of `char`'s**.
- String index starts from 0.

| | |
|---|---|
| String constant | `"AStringconstant"` |
| String concatenation | `s1+s2` and `s1+=s2` |
| `s.length()` | the length of a string `s`. |
| `s.charAt(i)` | character at position `i`. |

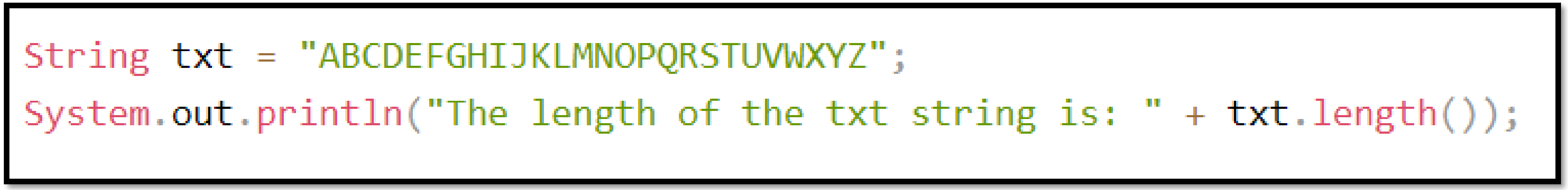
- toString: define a **string representation of the instances of the class**



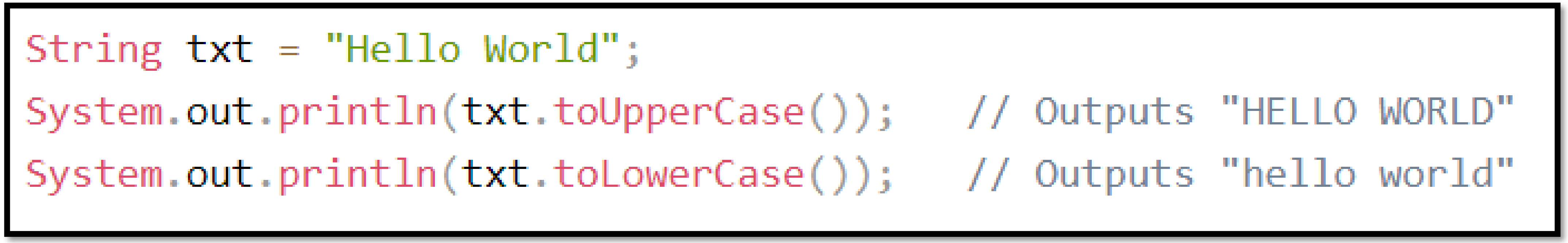
# Java Strings

Strings are used for storing text.

A `String` variable contains a collection of characters surrounded by double quotes:



```java
String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println("The length of the txt string is: " + txt.length());
```



```java
String txt = "Hello World";
System.out.println(txt.toUpperCase());   // Outputs "HELLO WORLD"
System.out.println(txt.toLowerCase());   // Outputs "hello world"
```

## Finding 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):

```
String txt = "Please locate where 'locate' occurs!";
System.out.println(txt.indexOf("locate")); // Outputs 7
```

```
String firstName = "John";
String lastName = "Doe";
System.out.println(firstName + " " + lastName);
```

```
String firstName = "John ";
String lastName = "Doe";
System.out.println(firstName.concat(lastName));
```

| Escape character | Result | Description |
|---|---|---|
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

```
String txt = "We are the so-called \"Vikings\" from the north.";
```

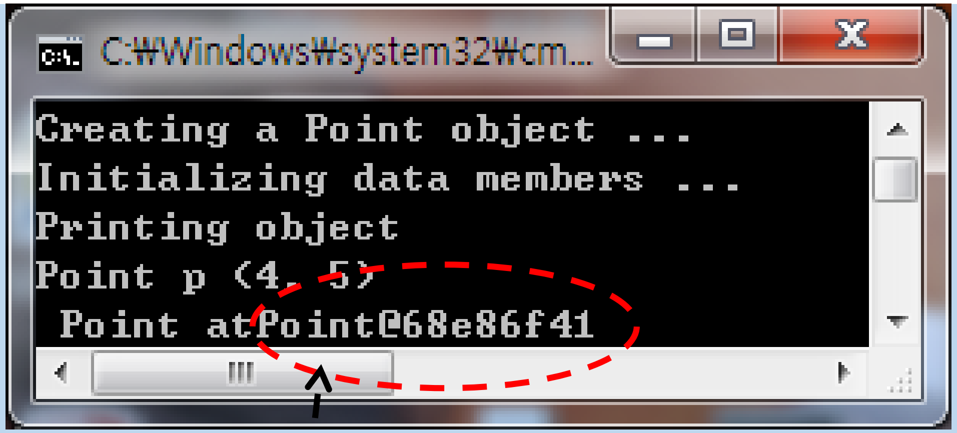
```
String txt = "It\'s alright.";
```

```
String txt = "The character \\ is called backslash.";
```

| Code | Result |
|---|---|
| \n | New Line |
| \r | Carriage Return |
| \t | Tab |
| \b | Backspace |
| \f | Form Feed |

```
class Point {

    int x;
    int y;
}
class TestPoint {
    public static void main(String[] args) {
        System.out.println("Creating a Point object ... ");
        Point p = new Point();
        System.out.println("Initializing data members ...");
        p.x = 4;
        p.y = 5;
        System.out.println("Printing object");
        System.out.println("Point p (" + p.x + ", " + p.y + ")");
        System.out.println(" Point at" + p);
    }
}
```

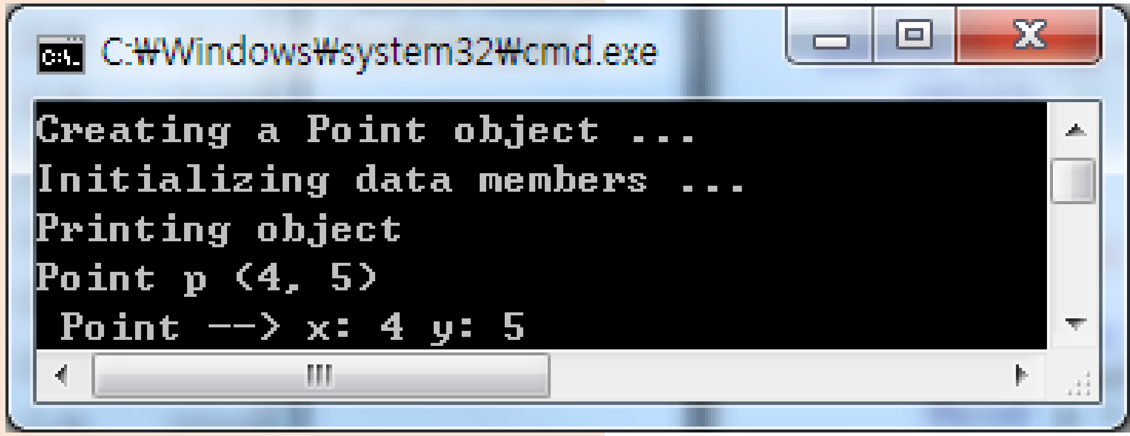


```
Creating a Point object ...
Initializing data members ...
Printing object
Point p (4, 5)
 Point atPoint@68e86f41
```

**No User-Defined toSting()**

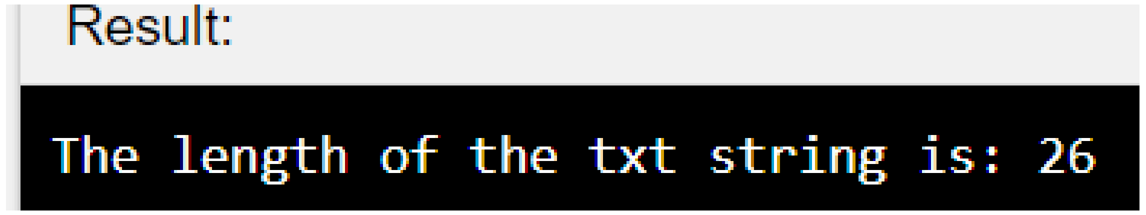
Should define a user-defined toString()

```
class Point {
    int x;
    int y;
   public String toString() {
           return "x: " + x +  " y: " + y;
   }
}
class TestPoint {
    public static void main(String[] args) {
        System.out.println("Creating a Point object ... ");
        Point p = new Point();
        System.out.println("Initializing data members ...");
        p.x = 4;
        p.y = 5;
        System.out.println("Printing object");
        System.out.println("Point p (" + p.x + ", " + p.y + ")");
        System.out.println(" Point --> " + p);
    }
}
```



```
Creating a Point object ...
Initializing data members ...
Printing object
Point p (4, 5)
 Point --> x: 4 y: 5
```

```
String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println("The length of the txt string is: " + txt.length());
```



Result:

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

```
String txt = "Hello World";
System.out.println(txt.toUpperCase());
// Outputs "HELLO WORLD"
System.out.println(txt.toLowerCase());
// Outputs "hello world"
```

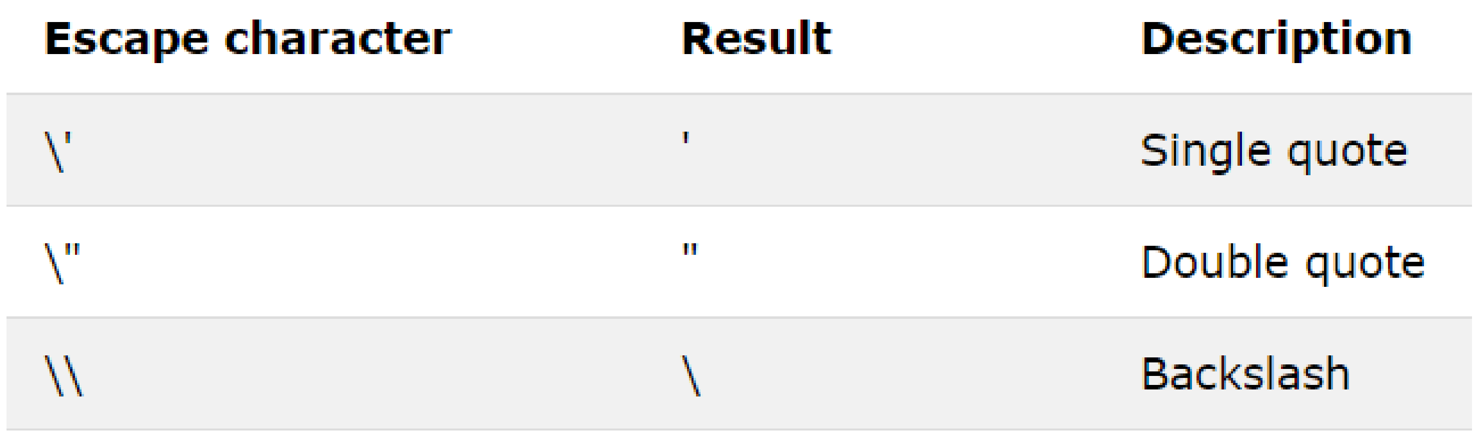
```
//Finding a Character in a String

String txt = "Please locate where 'locate' occurs!";
System.out.println(txt.indexOf("locate")); // Outputs 7
```

```
//String Concatenation
String firstName = "John";
String lastName = "Doe";
System.out.println(firstName + " " + lastName); OR

String firstName = "John "; String lastName = "Doe";
System.out.println(firstName.concat(lastName));
```

```
String txt = "We are the so-called "Vikings" from the north.";
```



| Escape character | Result | Description |
|---|---|---|
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

```
String txt = "We are the so-called \"Vikings\" from the north.";
```

```
String x = "10";
int y = 20;
String z = x + y;   // z will be 1020 (a String)
```

| Code | Result |
|---|---|
| \n | New Line |
| \r | Carriage Return |
| \t | Tab |
| \b | Backspace |
| \f | Form Feed |