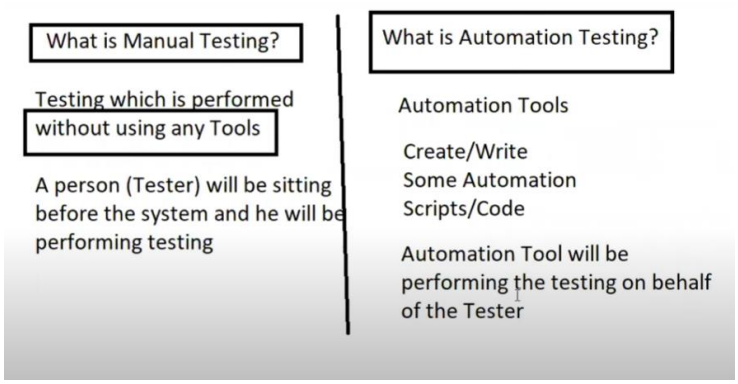
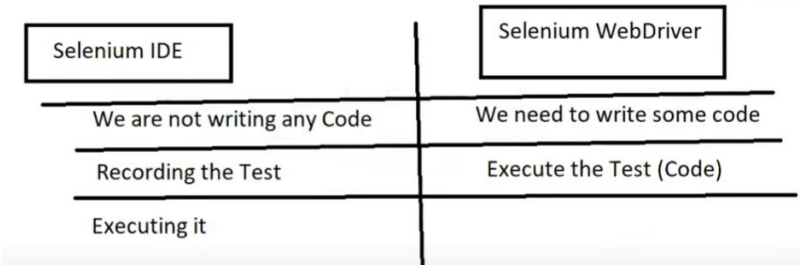
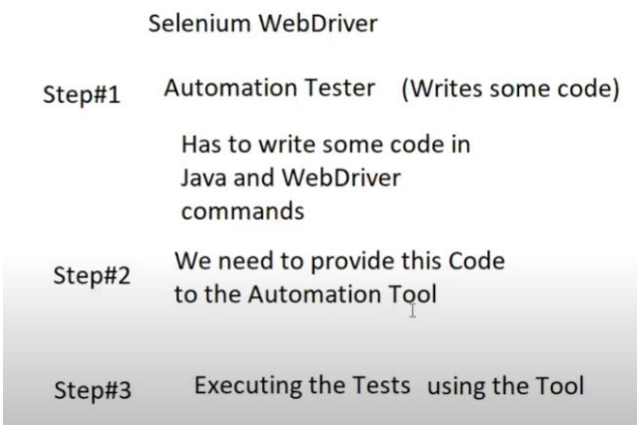
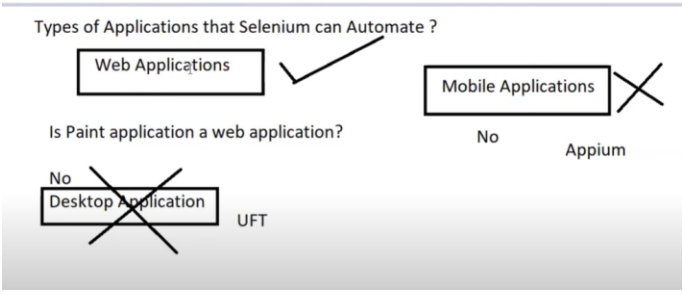


Selenium IDE is record and playback tool



How automation tool perform testing on behalf of tester





Selenium Supported Browsers

Test Environment

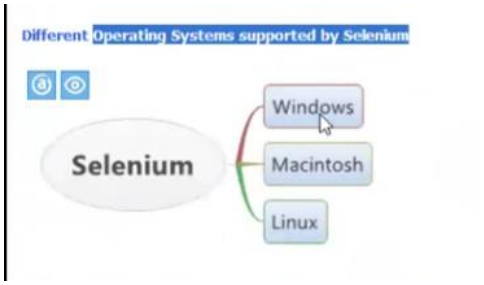
Executing the Tests

```
if(browser='Chrome'){

}
```

Chrome Browser
Firefox
Internet Explorer
Edge
Safari
Operara

Do we have to write seperate code for each and every browser?

No



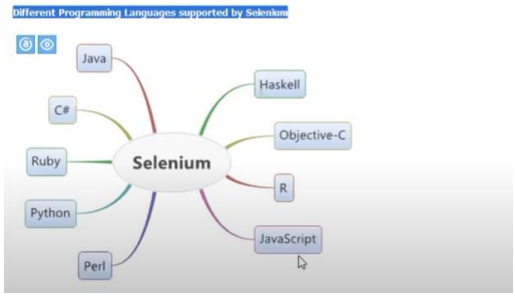
C# Programming Language

Is it compulsary for me to automate using C#?

No

Java

The Complete team can undestand the automation code and can work together



Different Components in Selenium?

Selenium is an Automation Tool

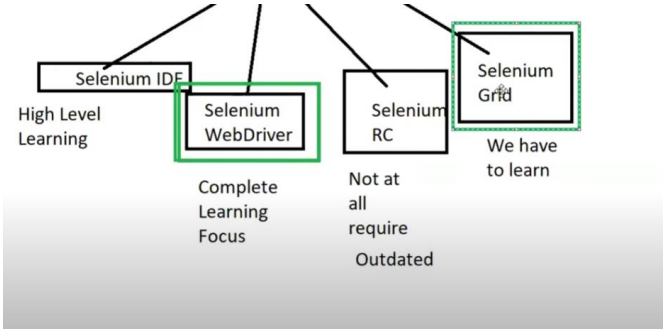
Selenium is not a single tool
It is a collection of serveral componets

Selenium IDE

Selenium WebDriver

Selenium RC

Selenium Grid



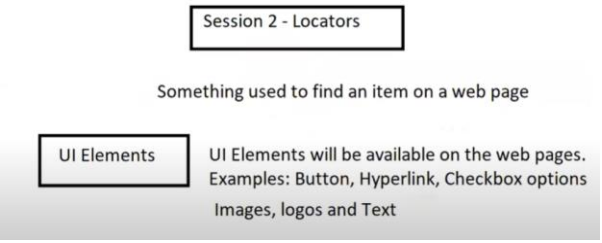
Selenium versions

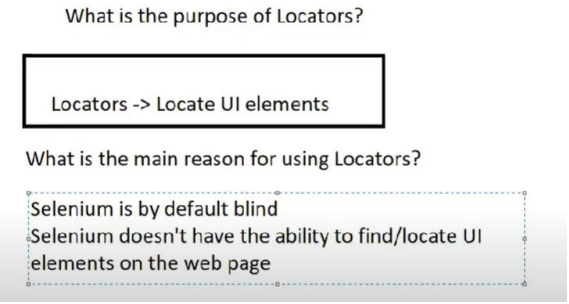
Selenium 1, selenium 2, selenium 3 , selenium 4

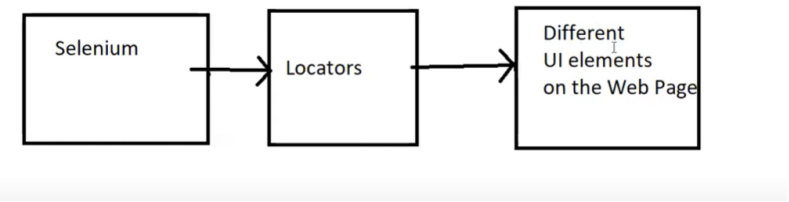
Official website:

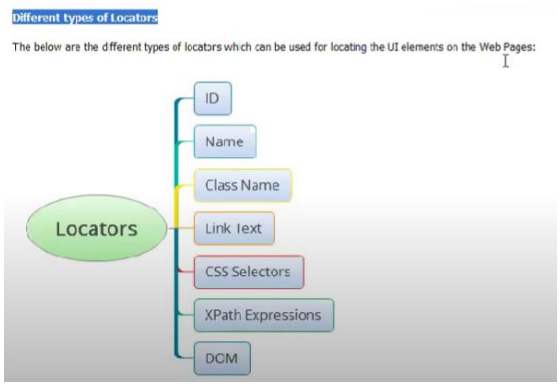
https://www.selenium.dev

Locators:









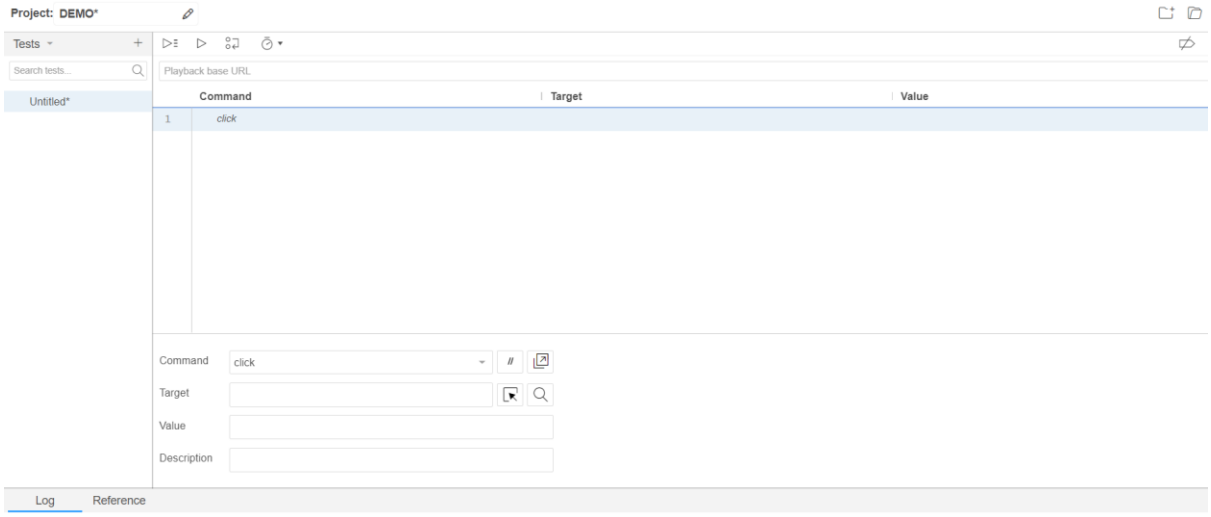
Selenium IDE:
search selenium IDE , click on first link , add chrome extension[supported in chrome and firedfox only]

Now go to https://omayo.blogspot.com/ this link

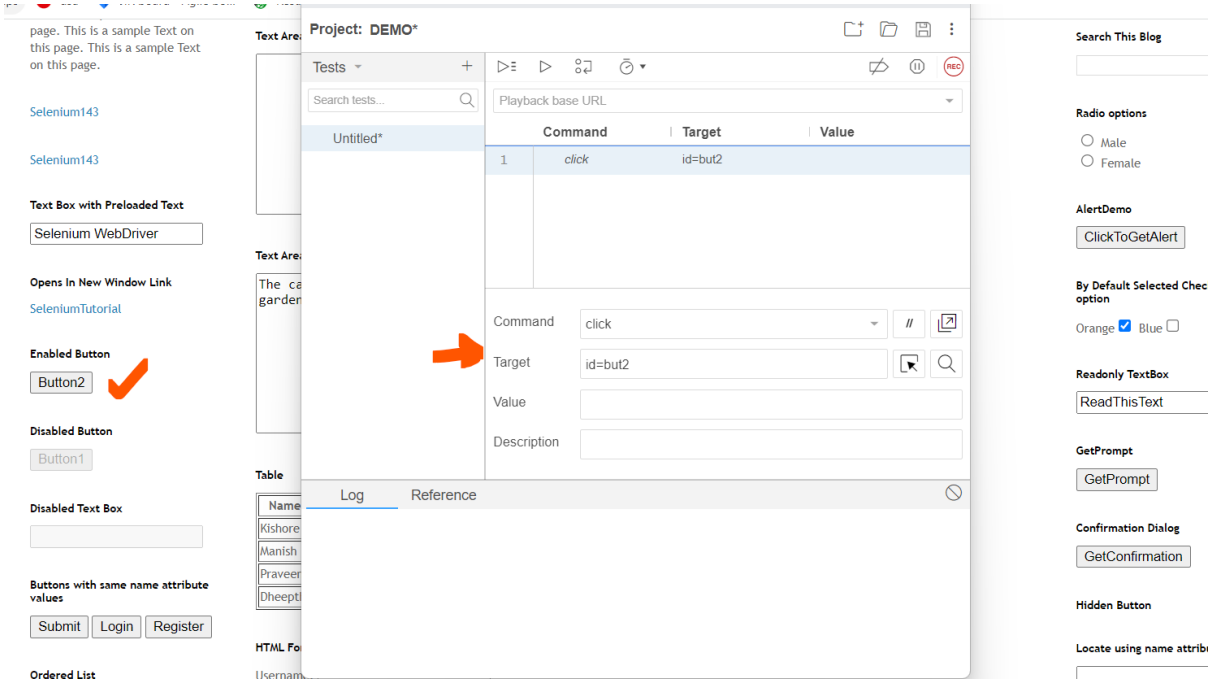
Then click on selenium IDE Extension

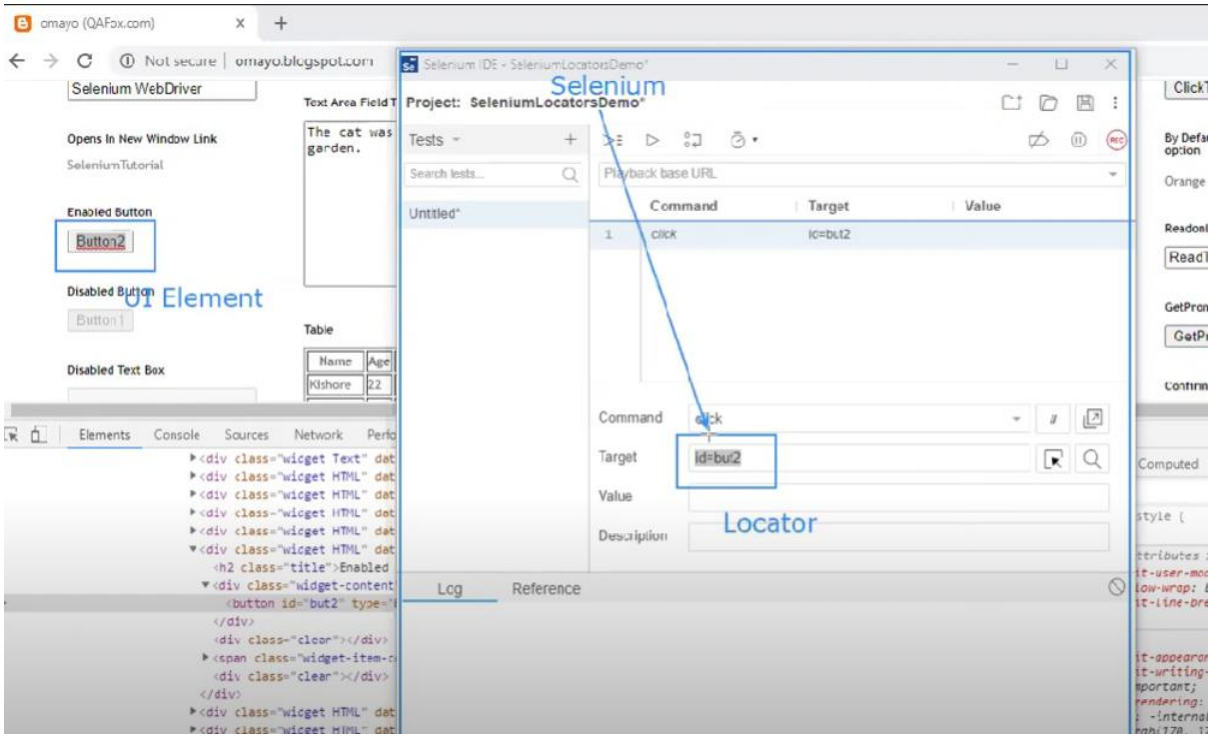
Select create a new project , give a name

Now we will get below screen , select a command from drop down



By passing attribute and value in target it will get highlighted





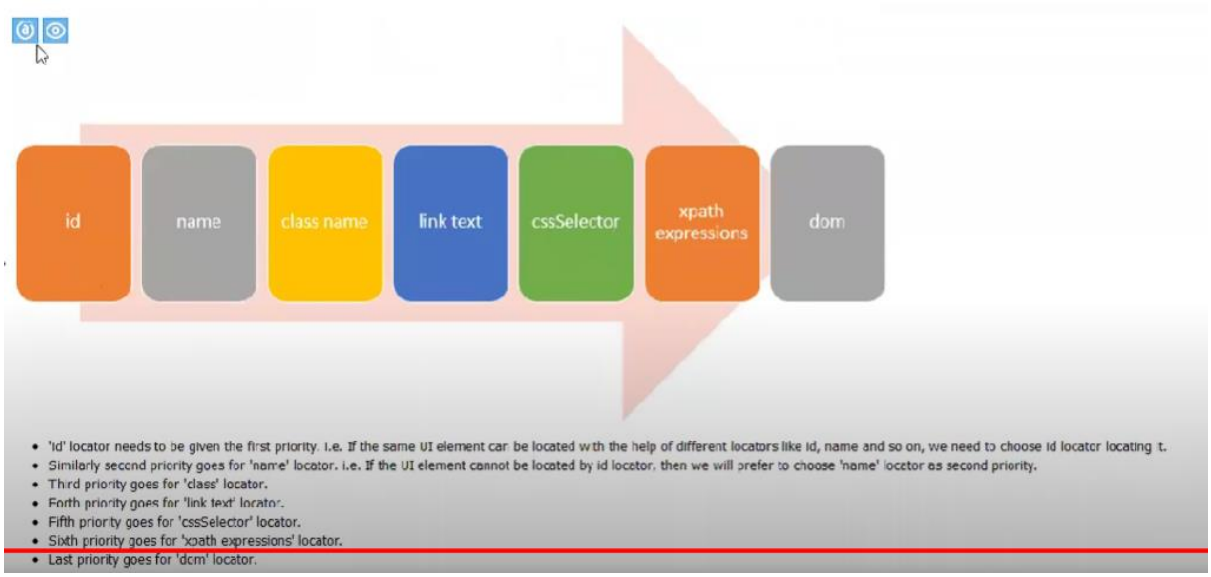
Selenium IDE supports locators: id,name

Doesnot support class locator

Note: class and dom locators won't work in Selenium IDE.

- id locator - Example: 'Button2' button in omayo blog - Syntax: id=but2
- name locator - Example: 'Locate using name attribute' text box field in omayo blog - Syntax: name=textboxn
- class locator - Example: text box "locate using class" in omayo blog - Syntax: class=classone
  - I will demonstrate using this locator during Selenium WebDriver sessions.
- link locator - Example: link 'compendiumdev' in omayo blog - Syntax: link=compendiumdev
- css locator - Example: 'Button2' button in omayo blog - Syntax: css=#but2
- xpath locator - Example: 'Button2' button in omayo blog - Syntax: xpath=//*[@id='but2']
- dom locator - Example: 'Button2' button in omayo blog - Syntax: dom=document.getElementById("but2")

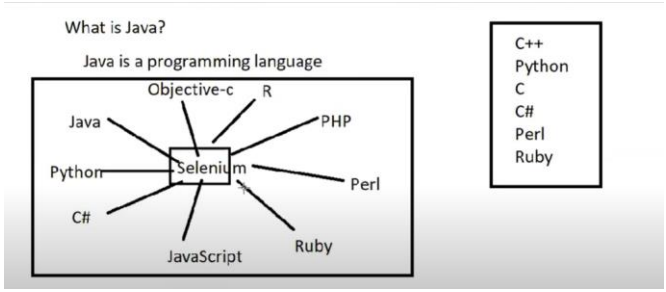
Priority of Locators

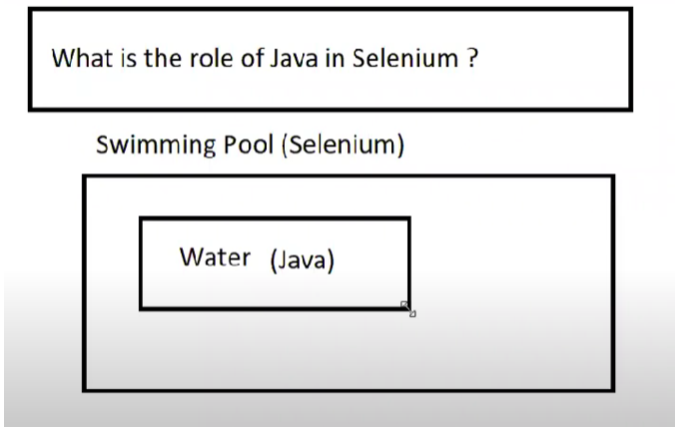


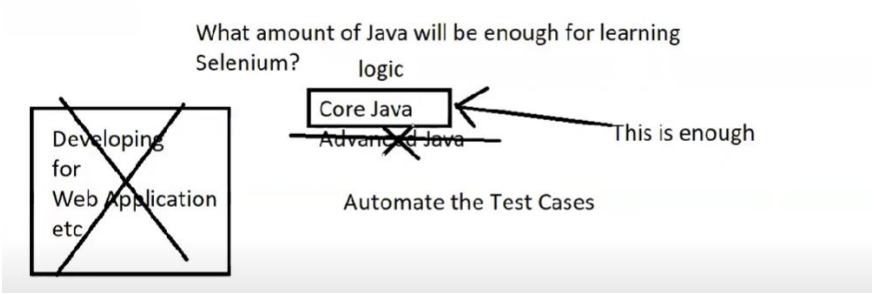
https://autonomiq.io/deviq-chropath.html

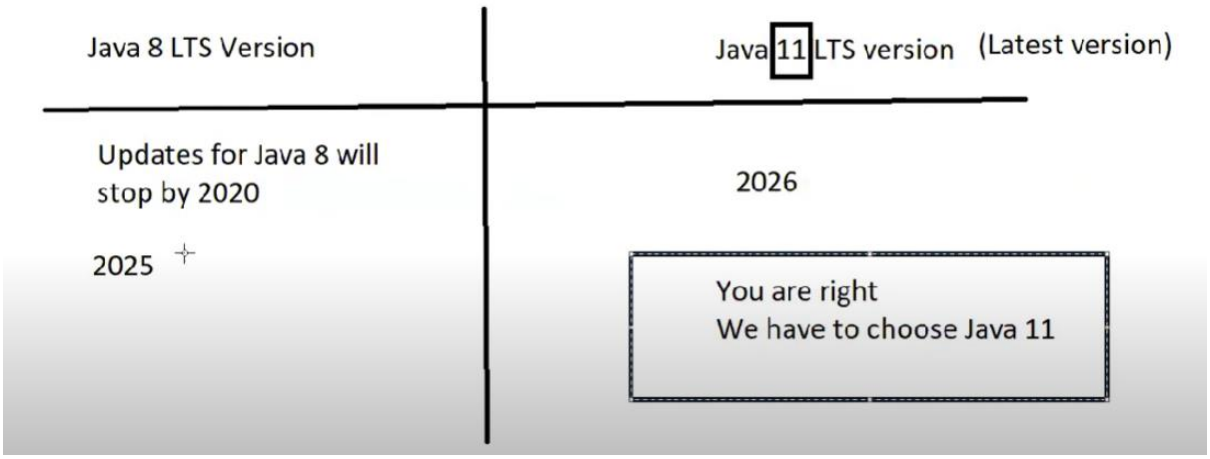
and download for chrome

Getting started with java:









Installation:

Go to download link search java jdk 11 download

https://www.oracle.com/in/java/technologies/javase/jdk11-archive-downloads.html

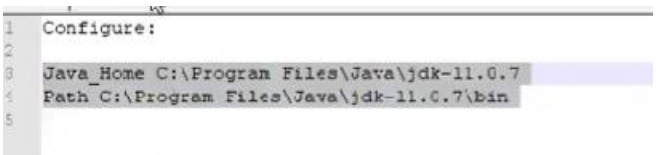
download .exe file of jdk depending on OS

create oracle account and download

double click on the downloaded exe file

install-next-next

Jdk path and jdk bin path



Windows search - advanced system settings- control panel

Click on advanced tab – click on environmental variables

System variables – new – give name as JAVA_HOME and give path of jdk

Then – path – edit – new – bin folder path

Then go to command prompt

Check **java -version**

C:\Users\MOULALI>java -version

java version "11.0.12" 2021-07-20 LTS

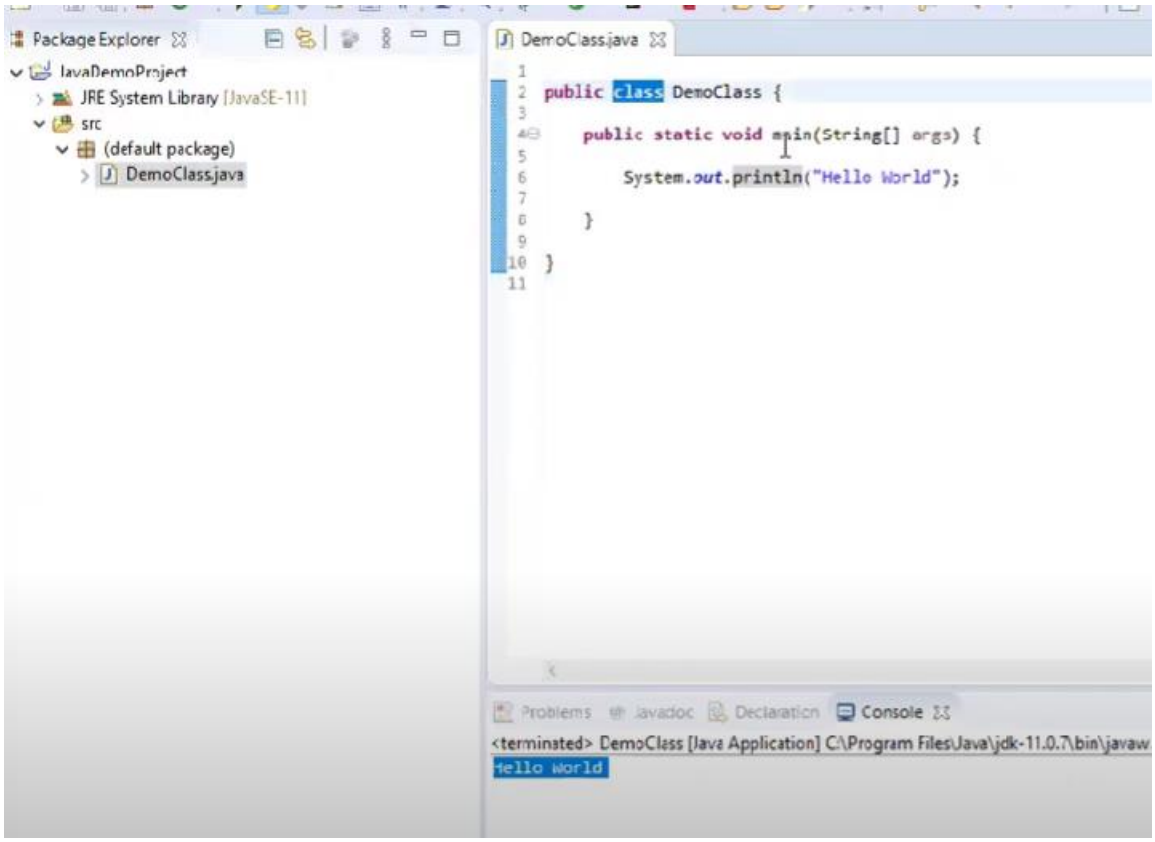
Java(TM) SE Runtime Environment 18.9 (build 11.0.12+8-LTS-237)

Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.12+8-LTS-237, mixed mode)

This shows java version in our machine

Install Eclipse:

Download eclipse of 64 bit , to overcome conflicts between 64bit java jdk and eclipse version



Eclipse IDE will automatically compiles our .java file and creates a .class file immediately

Part 2 of Java:

Understand the Java programs
Variables
Data Types
Operators
Literals

Java (Part 2) - Understanding Java Programs, Variables, Data Types, Literals and Operators

Understanding Java Programs

- In Java programs, we have to enclose everything inside a Class.
  - Syntax: public class ClassName { }
  - This *class* term is used as syntax to create/define a Class in Java
- In Java programs, execution starts from the main method
  - Syntax of main() method - public static void **main**(String args[]){ }
- All the Java statements in Java should end with ';' symbol
  - Example for a Java Statement is nothing but print statement - System.out.println("Hello World");
- All the Java statements should be written inside the methods
  - We generally write code which is nothing but a set of statements insice the methods
- Keywords like public, static, void and String args[] will be explained later

Compiler Errors

Java Complier Errors will be displayed when we make syntax mistakes in the Java Code:

- Example: All the Java statements in Java should end with ';' symbol
  - Remove the ; from the end of Java Statement
- Example: Java is case sensitive
  - Replace 'S' with 's' in the statement
- Example: Remove any of the closing brace

Understanding the Java programs

In Java programs, Everything will be enclosed inside a Class

How to create a Class in Java, and what is the keyword we have to use for creating a class in Java?

class

Java Keyword

```
public class Demo{

}
```

Demo.java

```
1
2
3  public class Demo{
4
5      //Can I run this code - No
6      //There is no main method inside this class
7
8
9  }
```

Conscle

```
<terminated> Demo [Java Application] C:\Program Files\Java\jdk-11.0.7\bin\javaw.exe (10-Jun-2020, 5:46:52 an
Error: Main method not found in class Demo, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

Demo.java

```
1
2
3  public class Demo{
4
5      public static void main(String[] args) {
6
7          //This thing is empty
8
9
10     }
11
12
13 }
```

In Java programs, all the Java Statements (Example: println statement) should be written inside the main method

```
1
2
3  public class Demo{
4
5      public static void main(String[] args) {
6
7          System.out.println("Arun Motoori");
8
9
10     }
11
12
13 }
```

Compiler Errors

Strict

If you break
the Java Rules

Not providing semicolon
at the end of the Java
statements

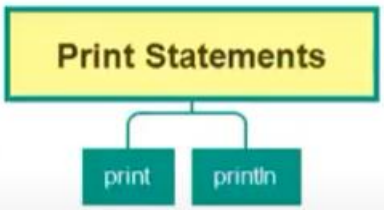
While writing the code
if you are getting some
syntax errors

Compiler Errors

Print Statements

Print statements in Java are used to print the program output to the console.

- The below are the two types of print statements in Java:

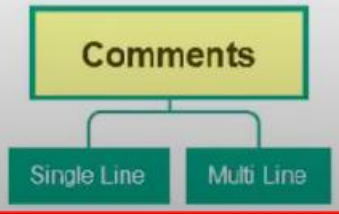


- Demonstrate print statements
- Demonstrate println statements
- Demonstrate printing a number and text
- Sysout + Ctrl + Space
- Syso + Ctrl + Space

Comments

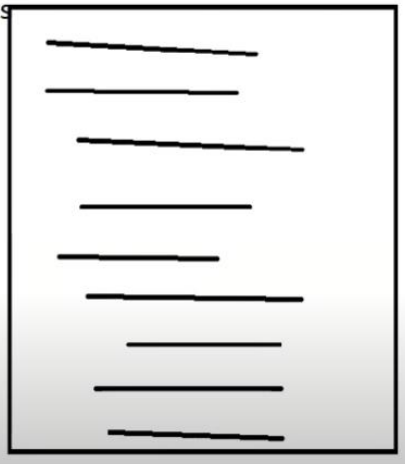
Comments provided in a Java program won't be executed and are generally used to explain the underlined code.

- The below are the two types of comments in Java:



Comments

30 days after



Why we have to use comments?

One - We can explain the underlying code

second - Without deleting the code, we can comment it and stop it from execution

Third - Documentation

- Demonstrate single line comments
  - Syntax: **// Sample Comment Text**
- Demonstrate multi line comments
  - Syntax: **/* Sample Comment Text */**

Variables, Data Types, Operators and Literals

In order to **store the data** in Java programs, we need to use Variables, Data Types, operators and Literals.

- Example: **int a = 5;**
  - Refer more details here
- Demonstrate a program which stores the data into a variable and prints it

Variables, Data Types, Operators and Literals

In order to **store the data** in Java programs, we need to use Variables, Data Types, operators and Literals.

- Example: **int a = 5;**
  - Refer more details here
- Demonstrate a program which stores the data into a variable and prints it

Variables

Variable is a name provided to a reserved memory location.

- Refer more details here

Data Types

We can define the variables with different Data types, based on the type of data to be stored.

- The below are the different data types in Java:
  - byte
  - short
  - int
    - Assigning another value to the same variable
    - Creating multiple variables of the same data type
  - long
  - double
  - float
  - char
  - boolean

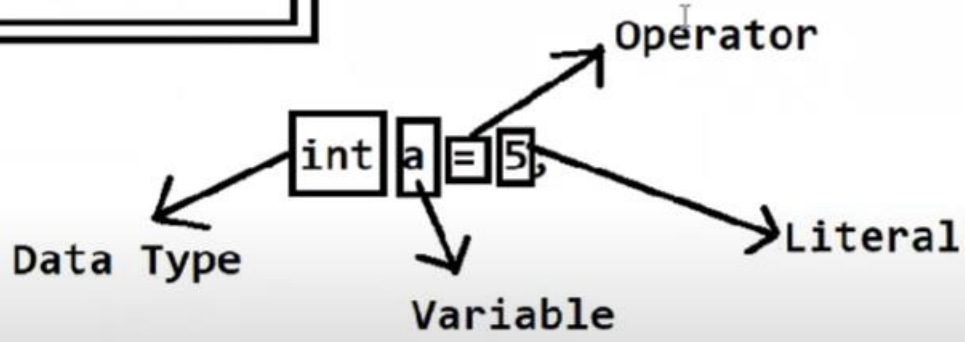
String is not a data type in Java

- String is a predefined class in Java

Variables
Data Types
Operators
Literals

Data

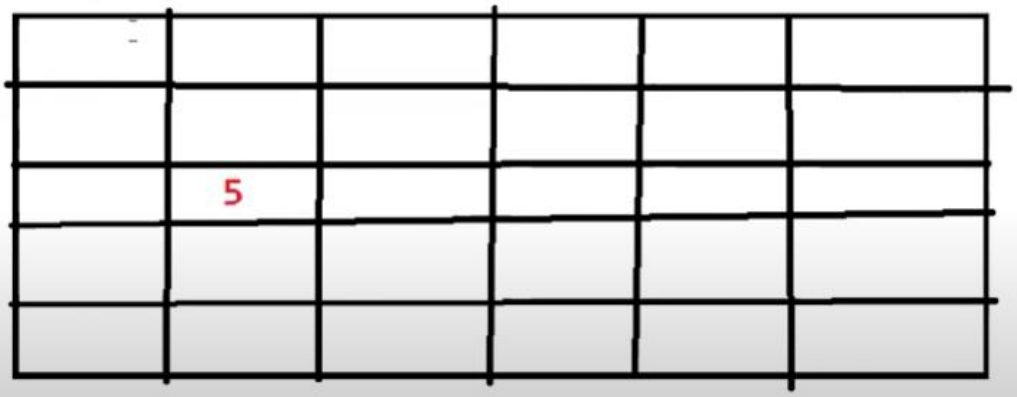
Storing data in the Java programs



Variables

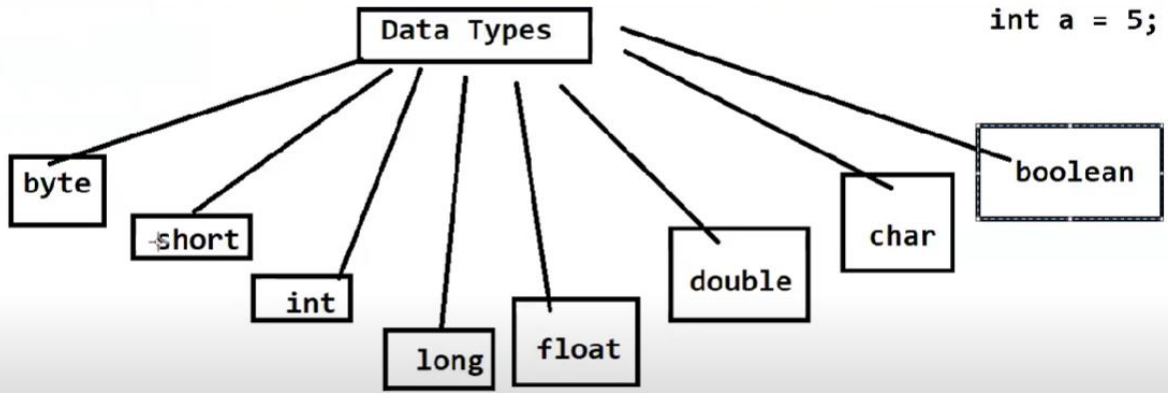
```
int a = 5;
```

Computer Memory



Variable is nothing but a name

```
int a = 5;
```



```java
public class Demo{

    public static void main(String[] args) {

        // int
        byte a = 123;
        short b = 12345;
        int c = 1234567891;
        long d = 12345678912345L;


        // double
        double e = 123343434343434343434343433334343434343434443.4563434343433434343434343433434343434343433434343443
        float f = 123343434343433334343434343434343443.4563434343433434343434343433434343434343433434343443F;

        //char
        char g = '$';


        // boolean
        boolean h = false;  // true or false


        //Do we have any data types in Java for storing string values
        // No

        String i = "Arun Motoori";
    }
}
```

java.lang.String

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

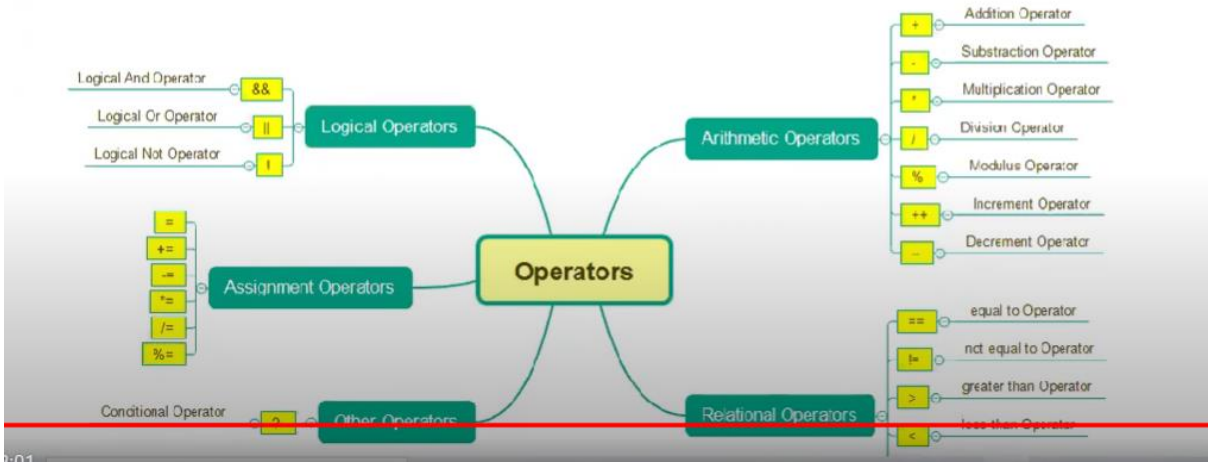
Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

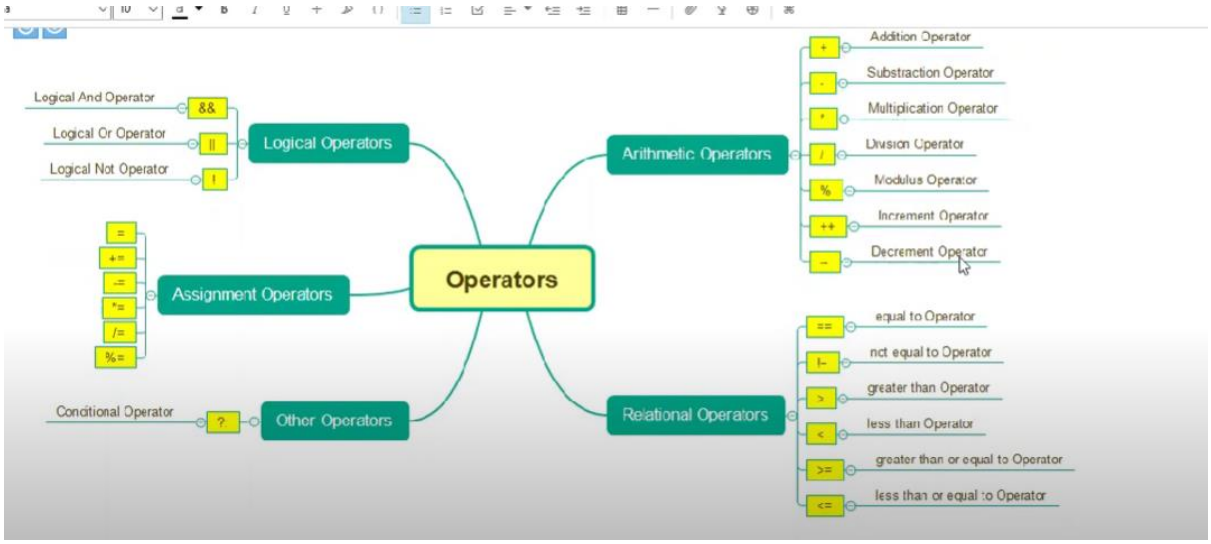
```
String str = "abc";
```

Operators

Operators are just symbols used to perform operations on the provided data.

- The below are the different types of Operators in Java:






```java
public class Demo{


    public static void main(String[] args) {

        int a = 9, b= 5,c;



            //Arithmetic Operators
            System.out.println();


    }

}
```

```java
public class Demo{


    public static void main(String[] args) {

        int a = 5, b = 6;

        boolean c = (a>b) ? true : false;



    }

}
```

Literals

Literals in Java are the representation of numeric, boolean, string and character data

- Integer Literals - Numerical values in the range of -2147483648 to 2147483648
- Long Literals - Numerical values in the range of -9223372036854775808 to 9223372036854775807
- Floating Point Literals - Example: 123.456F
- Double Literals - Example: 123.456
- Boolean Literals - true or false
- Character Literals - Example: 'S'
- String Literals - Example: "Hello World"

Printing Numerical and String literals using print statements and the combination of literals.

Miscellaneous

- Warning messages will be displayed for unused variables to represent wastage of memory.
- Trying to print the variable which is not assigned with any value
  - Default initialization values - 0, nul, false
- Trying to print the variables before declaration and assignment of values

Concatenation

"Arun"+" "+"Motoori"

Arun Motoori

Addition

5+6 11

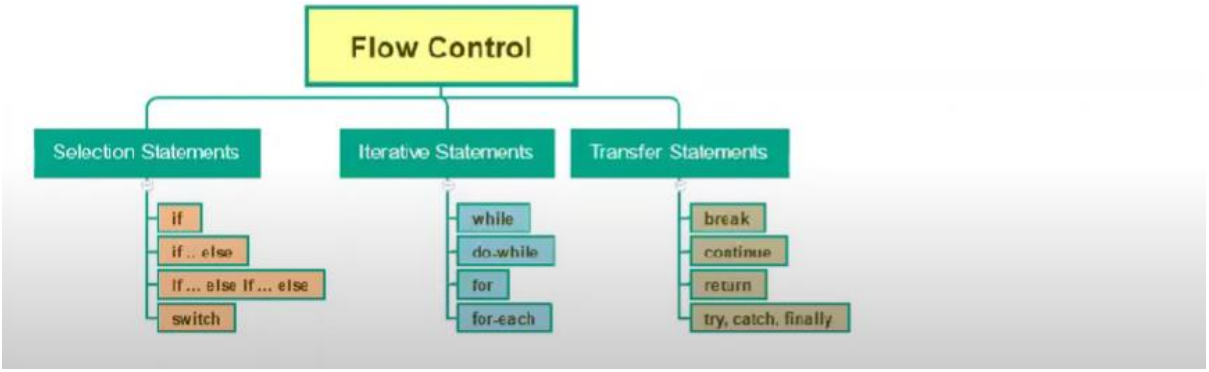
Addition operator

Java (Part 3) - Flow Control Statements

Flow Control

Flow Control describes the order in which statements will be executed at run time.

- There are different types of flow control statements and can be categorized as below:



Selection Statements

Selection Statements have one or more conditions which return either true or false when evaluated.

- Based on the returned value, the set of code will be executed.
  - When condition is true, the set of code will be executed.
  - When condition is false, the set of code wont be executed.
- Different types of Java Selection Statements:

Selection Statements

a =5,b = 3
(a>b)

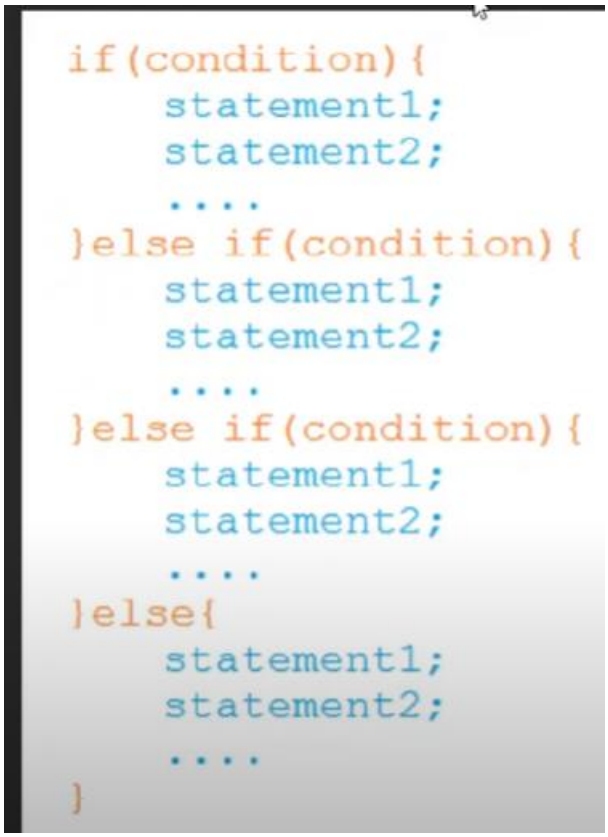
if
if .. else
if .. else if .. else
switch

- One or more conditions

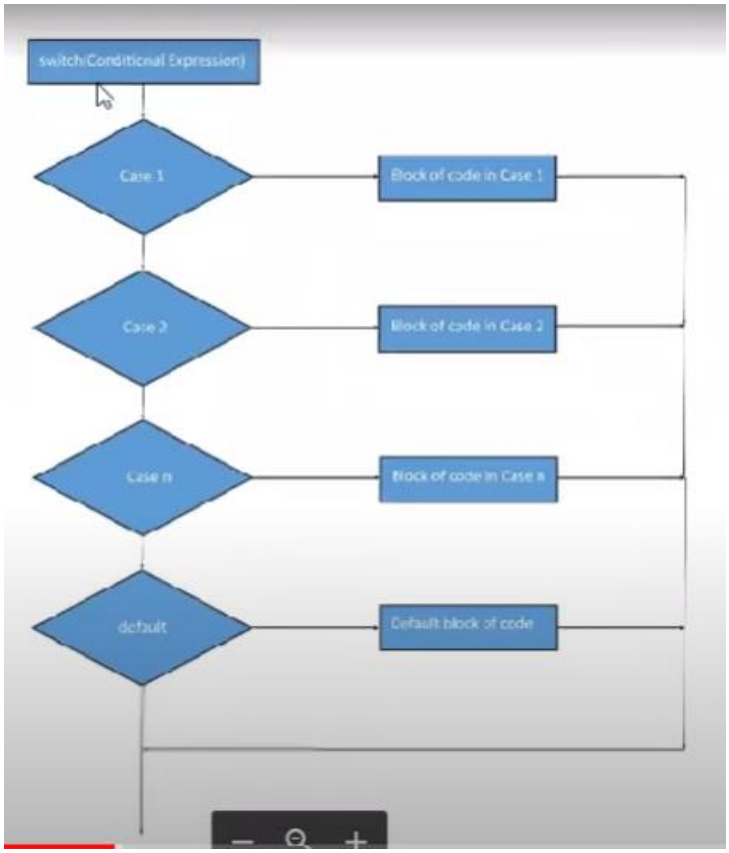
if - one condition
if ..else - one condition
if ..else if .. else - two or more conditions
switch - one condition

- **if statements**
  - Code inside the **if** decision making statement executed only when the condition provided inside if decision making statement returns true. (View screenshot here)
  - Syntax (View screenshot here)
  - Demonstrate **if** decision making statement
    - Demonstrate the execution of the statements inside if block when the if condition is true. (Demonstrate here)
    - Demonstrate that statements inside if block are skipped from execution when the if condition is false. (Demonstrate here)
- **if ... else statements**
  - While using **if ... else** decision making statement, code inside if block gets executed when the if condition is true and code inside else block gets executed when the if condition is false. (View screenshot here)
  - Syntax (View screenshot here)
  - Demonstrate if else decision making statement
    - Demonstrate the execution of statements inside if block when the if condition is true (Demonstrate here)
    - Demonstrate the execution of statements inside else block when the if condition is false (Demonstrate here)
- **if ... else if .. else statements**
  - **if .. else if .. else** statements contains more than one conditions. If the first if condition returns false, the code inside the if block will be skipped and control will be taken to the next conditions i.e. **else if** conditions. all the **else if** conditions return false, the code inside else if blocks will be skipped and the code inside the **else** block without condition will be executed. (View screenshot here)
  - Syntax (View screenshot here)
    - Demonstrate the program when all the if and else if conditions have returned false and code inside the else block is executed (Demonstrate here)
    - Demonstrate the program when the if condition has returned true and code inside the if block got executed skipping all the remaining else if and else blocks (Demonstrate here)
    - Demonstrate the program when one of the else if condition has returned true and code inside that else if block got executed skipping all the remaining if, else if and else blocks (Demonstrate here)
- **switch statements**
  - Based on result of a condition expression, switch case chooses one of many possibilities. (View screenshot here)
    - Syntax (View screenshot here)
    - The result of a condition expression needs to result a int or character or a string value.
    - Demonstrate switch case (Demonstrate here)
    - Demonstrate switch case which dont match any case and executes the code in the default section. (Demonstrate here)
    - Demonstrate what happens when we dont provide break; statements inside the cases. (Demonstrate here)

```java
public class Demo{

    public static void main(String[] args) {

        int a = 5, b = 3;

        if(a>b) {

            System.out.println("Inside if block as a is greater than b");

        }else {

            System.out.println("Inside else block as a is less than b");

        }


    }

}
```



```java
public class Demo{

    public static void main(String[] args) {

        int a = 5, b = 4,c =3, d=2,e =1;

        if(a<b) {

            System.out.println("Inside if block - a is less than b");

        }else if(b<c) {

            System.out.println("Inside first else if block - b is less than c");

        }else if(c<d) {

            System.out.println("Inside seccnd else if block - c is less than d");

        }else if(d<e){

            System.out.println("Inside third else if block - d is les than e");

        }else {

            System.out.println("Inside else block - e is the least number");

        }



    }

}
```

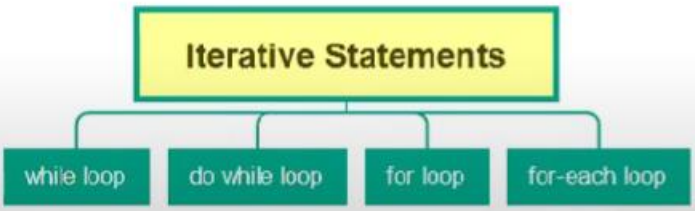



```java
public class Demo{

    public static void main(String[] args) {

        int a = 0;

        switch(a) {

        case 0:
            System.out.println("Inside case 0");
            System.out.println("Inside case 0");
            System.out.println("Inside case 0");
            break;
        case 1:
            System.out.println("Inside case 1");
            System.out.println("Inside case 1");
            break;
        case 2:
            System.out.println("Inside case 2");
            break;
        default:
            System.out.println("Inside default ... as no case is matching");

        }


    }

}
```

Iterative Statements

Iterative Statements helps us in executing the same block of code multiple times

- Iterative Statements executes the same set of code until the loop condition is satisfied. (View screen-shot here)
- Different types of Iterative Statements:



*Demo.java

```
1  public class Demo{
2
3      public static void main(String[] args) {
4
5
6          for(int i=0;i<150;i+-) {
7           System.out.println("This is a sample line");
8          }
9
10
11
12      }
13
14  }
```

- **while loop**
  - while loop executes the same block of code multiple times i.e. until the boolean condition turns false.
  - while loop tests the condition before executing the code in loop body. (View here)
  - Syntax: View here
  - Demonstrate while loop
    - Demonstrate the while loop when the condition is always true (Demonstrate here)
      - Loop will be iterated infinite times as the condition is always true
    - Demonstrate the while loop when the condition is false (Demonstrate here)
      - Loop will be executed at-least once as the condition is false.
    - Demonstrate the while loop where the condition is initially true and after few iterations is turned false (Demonstrate here)
      - Loop will be executed until the condition turns false.
- **do while loop**
  - do-while loop works similar to while loop, but the block of code will be executed at-least once even after the condition is false
  - Unlike while loop, do-while loop tests the condition after executing the code in loop body. (View here)
  - Syntax: View here
  - Demonstrate the do-while loop where the condition is initially true and after few iterations has turned false (Demonstrate here)
    - Loop will be iterated multiple times until the condition becomes false
- **for loop**
  - for loop is the most commonly used loop in Java.
  - for loop executes the same block of code multiple times, until the boolean condition turns false (View here)
  - Syntax: View here
  - Demonstrate the for loop when the condition is initially true and after few iteration has turned false (Demonstrate here)
    - Loop will be iterated multiple times until the condition becomes false
- **for-each loop**
  - Will be explained later, as it is generally used with Arrays and Collections.

```java
public class Demo{

    public static void main(String[] args) {

        int a=5,b=4;

        while(a<b) {

            System.out.println("Inside while block");

        }

        do {

            System.out.println("Inside while block");

        }while(a<b);

    }

}
```

while

condition is first before executing the while block

Condition - false

o times

do while

condition is last after executing the block

Condition - false

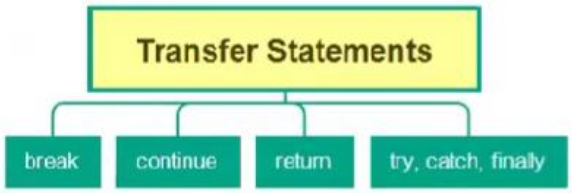
1

```
for(initialization-section;condition-section;increment/decrement-section){

    Statement1;
    Statement2;
    Statement3;
    .....

}
```

Transfer Statements

Transfer statements are used to transfer the flow of execution from one block of code to a different block of code.

- Different types of Transfer Statements:



- **break;**
  - The purpose of the break; statement is to come out of the statements based on some condition.
  - Demonstrate the usage of break; statements inside any loop statement (Demonstrate here)
- **continue;**
  - The purpose of the continue statement is to skip the current iteration of a loop based on some condition and continue with the next iteration.
  - Demonstrate on how to use continue; statements inside any loop statement (Demonstrate here)
- **return**
  - Will be explained later, as it is used with methods.
- **try, catch, finally**
  - Will be explained later, as it is used in Exception Handling

*Demo.java

```
1  public class Demo{
2
3      public static void main(String[] args) {
4
5          int a = 1;
6
7
8          switch(a) {
9
10         case 0:
11             System.out.println("Inside case 0");
12             break;
13         case 1:
14             System.out.println("Inside case 1");
15             break;
16         case 2:
17             System.out.println("Inside case 2");
18             break;
19         default:
20             System.out.println("Inside default cases");
21
22         }
23
24         System.out.println("After switch case");
25
26
27     }
28
29 }
```

```java
public class Demo{

    public static void main(String[] args) {

        int i=0;

        while(i<5) {

            if(i==3) {
                break;
            }

            System.out.println("Inside while loop having value of i as: "+i);

            i++;
        }

    }

}
```

Demo.java

```java
public class Demo{

    public static void main(String[] args) {

        int i=0;

        while(i<5) {

            if(i==3) {
                continue;
            }

            System.out.println("Inside while loop having value of i as: "+i);

            i++;
        }

    }

}
```

break

Take you out of loops and switch case

continue

Only skip the current iteration

i

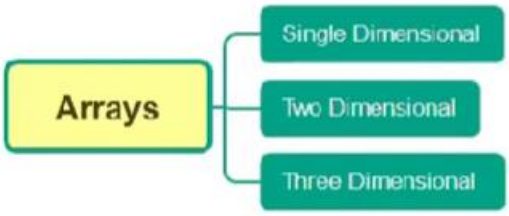
0,1,2,3,4,5,6,7,8,9

Skipped

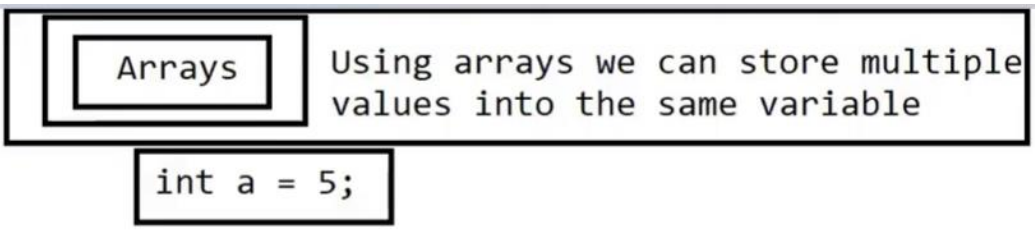
Java (Part 4) - Arrays, Methods and Variables

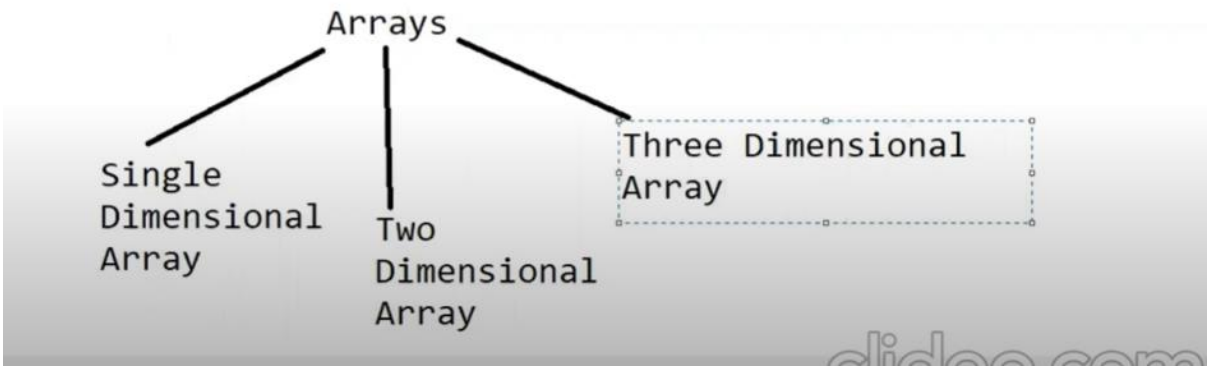
Arrays

- Using Arrays, multiple values of same data type can be stored into a single variable.
- Arrays can be categorized as below:

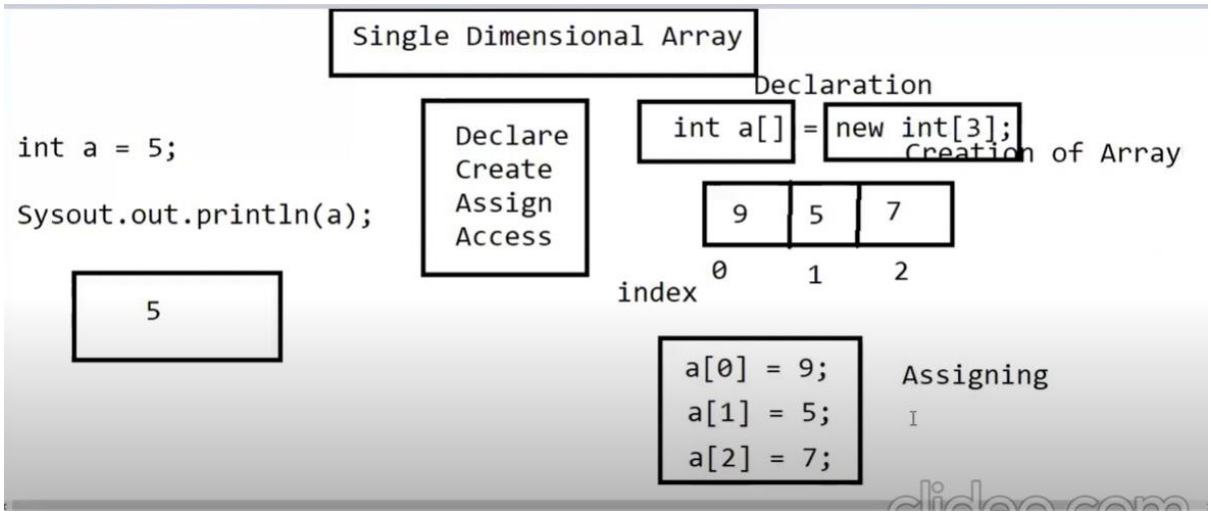


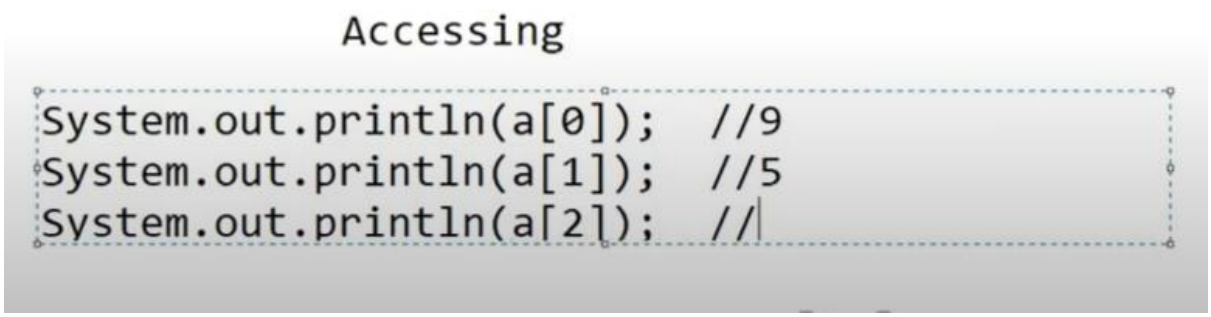
- **Single Dimensional Array**
  - Example: **int[] a = new int[3];**
  - Demonstrate Declaring, Creating, Assigning and Accessing the single dimensional Array - Demonstrate here
  - View the diagrammatic representation of single dimensional array here
  - Shortcut representation of single dimensional array - Demonstrate here
  - 'length' predefined variable of Arrays - Demonstrate here
  - Using for loop with single dimensional arrays - Demonstrate here
  - Using for-each loop with single dimensional arrays - Demonstrate here
  - ArrayIndexOutOfBoundsException





- **Single Dimensional Array**
  - Example: **int[] a = new int[3];**
  - Demonstrate Declaring, Creating, Assigning and Accessing the single dimensional Array - Demonstrate here
  - View the diagrammatic representation of single dimensional array here
  - Shortcut representation of single dimensional array - Demonstrate here
  - 'length' predefined variable of Arrays - Demonstrate here
  - Using for loop with single dimensional arrays - Demonstrate here
  - Using for-each loop with single dimensional arrays - Demonstrate here
  - ArrayIndexOutOfBoundsException
- **Two Dimensional Array**
  - Example: int[][] a = new int[2][3];
  - Two dimensional Array is nothing but array of single dimensional arrays.
  - View the diagrammatic representation of two dimensional array here
  - Demonstrate Declaring, Creating, Initializing and Accessing the two dimensional Array - Demonstrate here
  - Shortcut representation of two dimensional array - Demonstrate here
  - 'length' predefined variable of Arrays - Demonstrate here
  - Using for loop with two dimensional arrays - Demonstrate here






```
int a[] = new int[3];

int[] a = new int[3];
```

```java
public class Demo{

    public static void main(String[] args) {

        int[] a = new int[3];   //Declaration and creation of Array

        //Assigning the values to the Array Elements
        a[0] = 9;
        a[1] = 5;
        a[2] = 7;

        //Accessing the Array Elements
        System.out.println(a[0]);  //9
        System.out.println(a[1]);  //5
        System.out.println(a[2]);  //7


    }

}
```

```java
public class Demo{

    public static void main(String[] args) {

        int[] a = {9,5,7};  //Declaring, Creating and Assignment

        //Accessing the Array Elements
        System.out.println(a[0]);  //9
        System.out.println(a[1]);  //5
        System.out.println(a[2]);  //7


    }

}
```

Finding the size of the Array

`length` - Predefined variable of Arrays

Predefined/inbuilt

```
int a = 5;
```

user defined

```java
public class Demo{

    public static void main(String[] args) {

        int[] a = {9,5,7,1,3,8,0,2,4};

        System.out.println("The size of the array is "+a.length);  //3


    }

}
```

```java
public class Demo{

    public static void main(String[] args) {

        int[] a = {9,5,7,1,3,8,0,2,4};

        for(int i=0;i<a.length;i++) {

            System.out.println(a[i]);  //9 5 7 1 3 8 0 2 4

        }


    }
}
```

```java
public class Demo{

    public static void main(String[] args) {

        int[] a = {9,5,7,1,3,8,0,2,4};

        //for-each loop

        for(int i:a) {

            System.out.println(i);

        }


    }
}
```

```java
public class Demo{

    public static void main(String[] args) {

        int[] a = {9,5,7,1,3,8,0,2,4};

        //for-each loop

        for(int i:a) {

            if(i==3) {
                break;
            }

            System.out.println(i);

        }


    }
}
```

Arrays

for

for-each

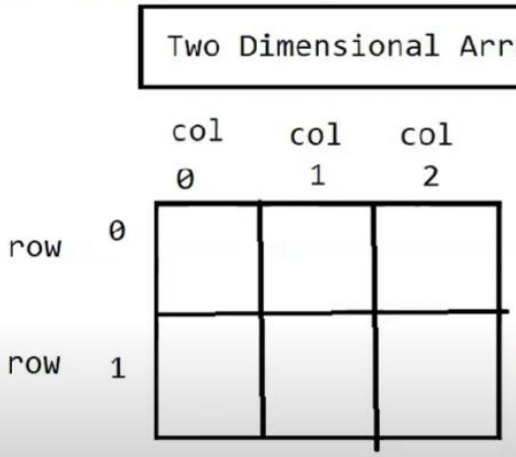
This is for Arrays

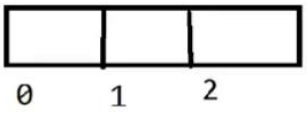
Error

ArrayIndexOutOfBoundsException

When we exceed the size of the Array

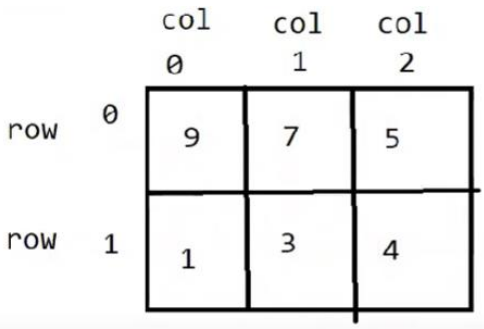
Two Dimensional Array






```
int[] a = new int[3];
```

```
int[][] a = new int[row][col];
```




```
int[] a = new int[3];
```

```
int[][] a = new int[2][3];
a[0][0] = 9;
a[0][1] = 7;
a[0][2] = 5;
a[1][0] = 1;
a[1][1] = 3;
a[1][2] = 4;
```

```java
public class Demo{

    public static void main(String[] args) {

        int[][] a = new int[2][3];

        a[0][0] = 9;
        a[0][1] = 7;
        a[0][2] = 5;

        a[1][0] = 1;
        a[1][1] = 3;
        a[1][2] = 4;

        System.out.println(a[0][0]);
        System.out.println(a[0][1]);
        System.out.println(a[0][2]);
        System.out.println(a[1][0]);
        System.out.println(a[1][1]);
        System.out.println(a[1][2]);


    }
}
```

```java
public class Demo{

    public static void main(String[] args) {

        int[][] a = {{9,7,5},{1,3,4}};


        System.out.println(a[0][0]);
        System.out.println(a[0][1]);
        System.out.println(a[0][2]);
        System.out.println(a[1][0]);
        System.out.println(a[1][1]);
        System.out.println(a[1][2]);


    }

}
```

a.length - Number of rows - 2

a[0].length - Number of cols

a[1].length - 3

```java
public class Demo{

    public static void main(String[] args) {

        int[][] a = {{9,7,5},{1,3,4}};

        for(int row=0;row<a.length;row++) {

            for(int col=0;col<a[0].length;col++) {

                System.out.println(a[row][col]);

            }

        }

    }

}
```

```java
public class Demo{

    public static void main(String[] args) {

        char[] a = {'s','a','m'};



    }

}
```

```java
public class Demo{

    public static void main(String[] args) {

        String[] a = {"Arun","Motoori","Selenium","Java"};



    }

}
```

```java
public class Demo{

    public static void main(String[] args) {

        boolean[] a = {true,false,false,false,true,true};



    }

}
```

Demo.java

```java
public class Demo{

    public static void main(String[] args) {

        //Arrays are fixed in size

        int[] a = new int[3];

         a[0] = 9;
         a[1] = 5;
         a[2] = 7;
         a[3] = 8;  // ArrayIndexOutOfBoundsException


    }

}
```

Console

```
<terminated> Demo [Java Application] C:\Program Files\Java\jdk-11.0.7\bin\javaw.exe (15-Jun-2020, 7:11:53 an
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out o
        at Demo.main(Demo.java:12)
```

Disadvantages of Array

Arrays are fixed in size
ArrayList is the solution

We cannot store mulitple types of values into a single array

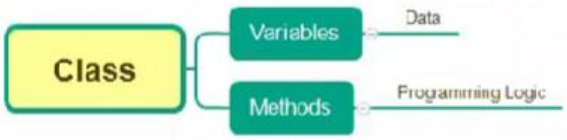
Object Arrays

Object
- Predefined/Inbuilt Class
- Parent of all the Classes in Java

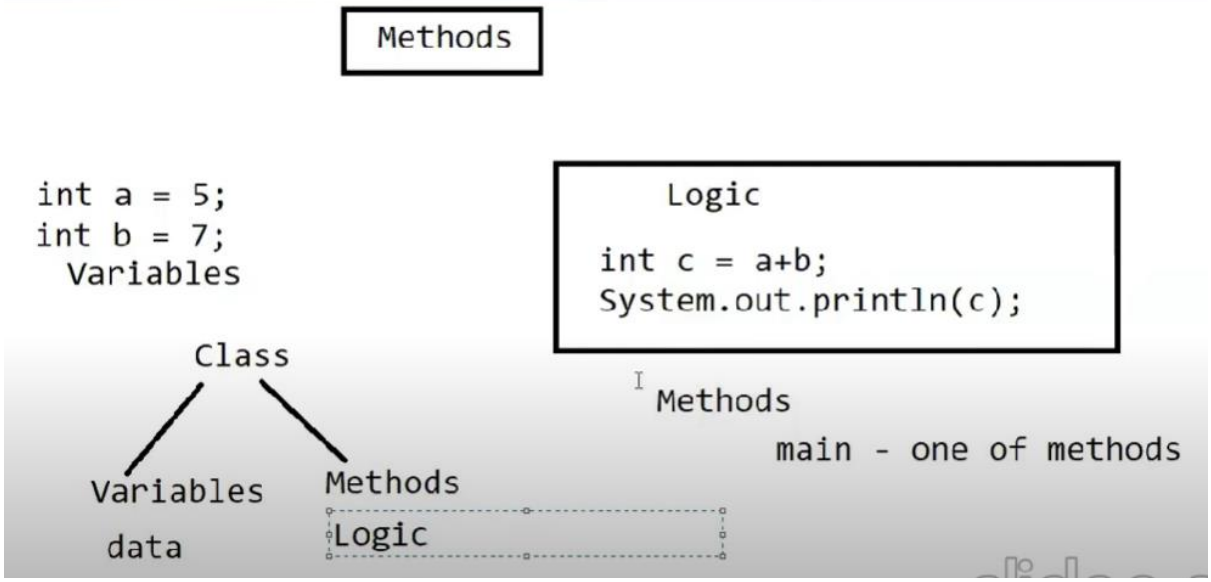
- **Others topics on Arrays**
  - Arrays and different Data type declaration
  - Disadvantages of Arrays
    - Fixed in Size
      - Solution: Collections Framework - ArrayList
    - Cannot store different types of literals into a single variable
      - Solution: Object Arrays

Methods

In Java programming, programming logic needs to be written inside methods:



- **main()** method is a method where the program execution starts and we can write programming logic inside the main() method - Demonstrate here
- Demonstrate creating multiple methods along with the below - Demonstrate here
  - Creating multiple methods along with main() method
    - All the method should reside inside the Class
  - main() method calling other method
  - non-main() method calling other method
  - method() calling other method multiple times
- Demonstrate single, multiple parameterized methods and passing arguments to those methods - Demonstrate here
  - Create a single parameterized method
  - Create a multiple parameterized method
  - Call the single and multiple parameterized methods by passing the arguments while calling
- Demonstrate returning the values back to the calling methods - Demonstrate here
  - Return nothing from a method
  - Return int value from a method
  - Return String value from a method




```java
public class Demo{

    public static void main(String[] args) {

        methodOne(); // Method Calling statement

        methodTwo(); // Method calling statement

    }

    public static void methodOne() {

        System.out.println("Inside methodOne");

    }

    public static void methodTwo() {


        System.out.println("Inside methodTwo");

    }


}
```

```java
public class Demo{

    public static void main(String[] args) {

        methodOne();

    }

    public static void methodOne() {

        System.out.println("Inside methodOne");

        methodTwo();

    }

    public static void methodTwo() {


        System.out.println("Inside methodTwo");

    }


}
```

```java
public class Demo{

    public static void main(String[] args) {

        methodOne(9);

    }

    public static void methodZero() {

        System.out.println("Inside methodZero");

    }

    public static void methodOne(int a) {

        System.out.println("Inside methodOne - Parameter value as "+a);

    }

    public static void methodTwo(int a,int b) {

        System.out.println("Inside methodOne - Parameter value as "+a);

    }

    public static void methodFive(int a,int b,int c,int d,int e) {

     System.out.println("Inside methodOne - Parameter value as "+a);

    }


}
```

*Demo.java

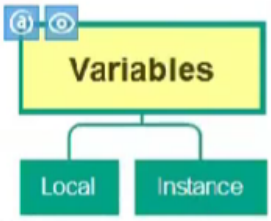
```java
public class Demo{

    public static void main(String[] args) {

        add(9.5);

    }

    public static int add(int a,int b) {

        int sum = a+b;

        return sum;

    }

}
```

```java
public class Demo{

    public static void main(String[] args) {

        String result = methodSample();

        System.out.println(result);

    }

    public static int add(int a,int b) {

        int sum = a+b;

        return sum;

    }

    public static String methodSample() {

        return "Arun Motoori";

    }


}
```

Variables

Variable is a name provided to a reserved memory location.

- Refer more details here
- There are three types of Variables:

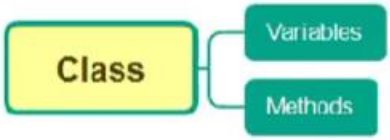


- Local Variables
  - A variable which is declared inside the method is called local variable (Demonstrate here)
- Instance and Static Variables
  - A variable which is declared inside the class but outside the method is called Instance variable (Demonstrate here)
  - We have to specify the static text before the instance variable as the method which is going to use this variable is a static method
    - This concept will be explained in upcoming sessions.
- Scope of the variables

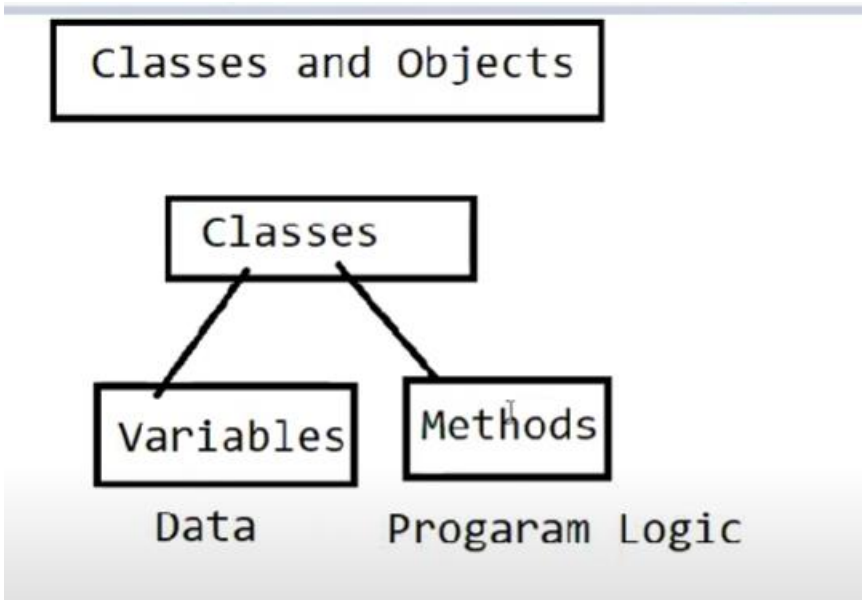
Java (Part 5) - Classes, Objects, Strings, Wrapper Classes and Constructors

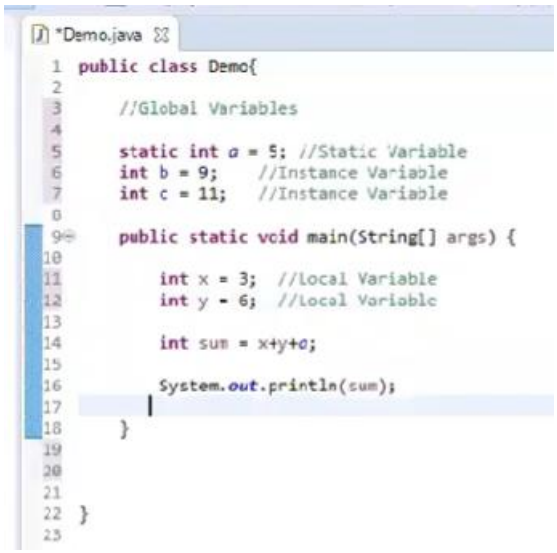
Classes and Objects

- Class encloses variables and methods - Demonstrate here



- **Class** is a template used for creating **Objects**
- Demonstrate creating a Class and use the Class as a template for creating Objects - Demonstrate here
  - Create a Class having a main method say Demo
  - Inside the same Java file create another Class named Car
  - Create any variables inside the Car class say model, cost, color
  - Create any methods inside the Car class say startCar(), stopCar(), carDetails()
  - Create any objects using Car class
  - Initialize and Access the variables & methods of Objects
- Object Creation Statement
  - **Car benz = new Car();** - View here





Instance and Static Variables which are outside the methods can be called as

Global Variables

Static methods can only access the static stuff

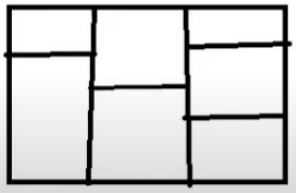
*Demo.java

```java
public class Demo{

    //Global Variables

    static int a = 5; //Instance Variable
    int b = 9;    //Instance Variable
    int c = 11;   //Instance Variable

    public static void main(String[] args) {

        int x = 3;  //Local Variable
        int y = 6;  //Local Variable

        int sum = x+y+a;

        System.out.println(sum);

        sample();

    }


    //non-static method
    public void sample() {

        System.out.println("Inside sample method");


    }


}
```

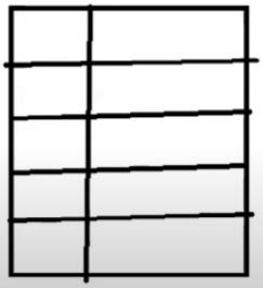
Class is a template for creating Objects

Objects mean real world entities

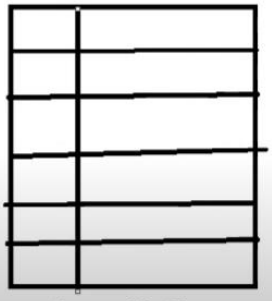
Plan for building a house



Plan



Real Building



Real Building

Class

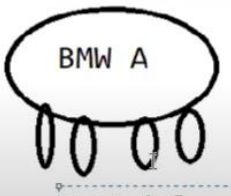
Car

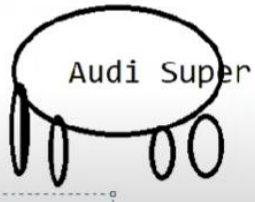
4 wheels
Color
Model
Milage
Cost

Real world enties

Objects

Black





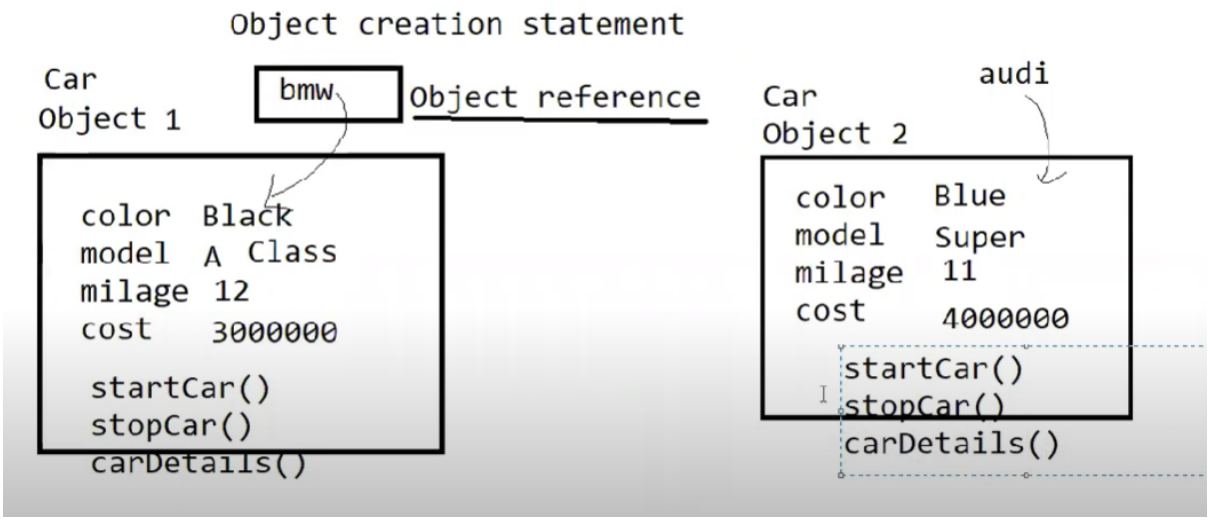
A Model

Static methods can only access static stuff

- static variables
- static methods

Non-static methods can access both static and non-static stuff

- Instance and Static variables
- Call static and non-static methods



*Car.java

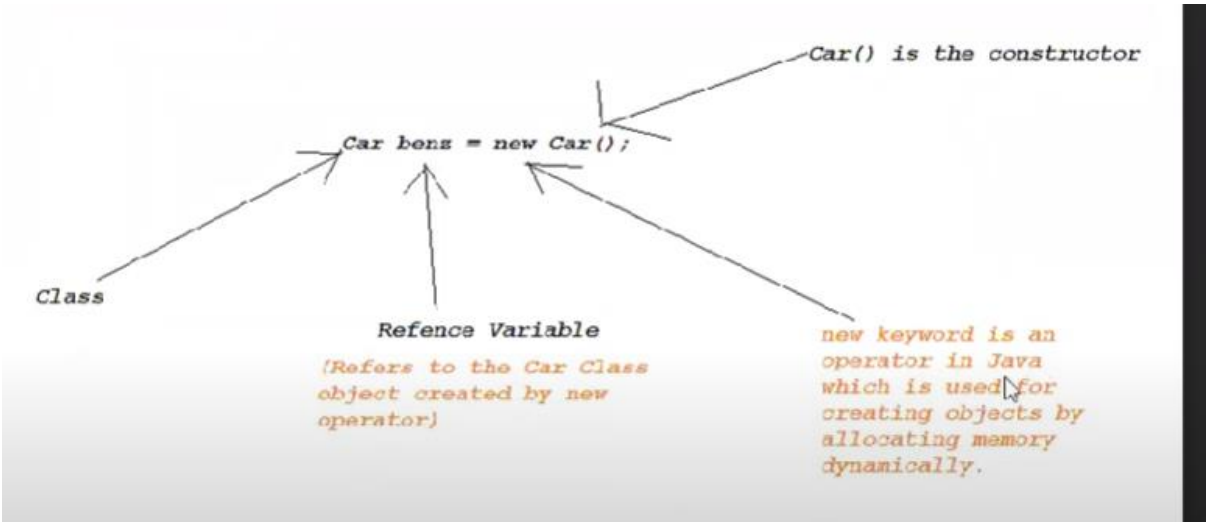
```
1  //Class is a template for creating objects
2  public class Car {
3
4      String color;
5      String model;
6      int milage;
7      int cost;
8
9      public void startCar() {
10
11         System.out.println(model+" car started");
12
13     }
14
15     public void stopCar() {
16
17         System.out.println(model+" car stopped");
18
19     }
20
21     public void carDetails() {
22
23         System.out.println("Color of car is "+color);
24         System.out.println("Model of car is "+model);
25         System.out.println("Milage of car is "+milage);
26         System.out.println("Cost of car is "+cost);
27
28     }
29
30
31 }
32
```

*Demo.java

```
1  public class Demo {
2
3      //Objects should not created outside the methods
4
5      public static void main(String[] args) {
6
7          //Object Creation Statement
8
9          Car bmw = new Car();
10         Car audi = new Car();
11
12         //Initializing the object
13
14         bmw.color = "Black";
15         bmw.model = "A Class";
16         bmw.milage = 12;
17         bmw.cost = 3000000;
18
19         audi.color = "Blue";
20         audi.model = "Super";
21         audi.milage = 11;
22         audi.cost = 4000000;
23
24         //Access the methods
25
26         bmw.startCar();
27         bmw.stopCar();
28         bmw.carDetails();
29
30         audi.startCar();
31         audi.stopCar();
32         audi.carDetails();
33
34     }
35 }
36
37
```

*Demo.java

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

        //Object Creation Statement

        Car bmw = new Car();
        Car audi = new Car();

        //Initializing the object

        bmw.color = "Black";
        bmw.model = "A Class";
        bmw.milage = 12;
        bmw.cost = 3000000;



        audi.color = "Blue";
        audi.model = "Super";
        audi.milage = 11;
        audi.cost = 4000000;

        //Access the methods

        bmw.startCar();
        bmw.stopCar();
        bmw.carDetails();

        audi.startCar();
        audi.stopCar();
        audi.carDetails();

        //Accessing the variables
        System.out.println(bmw.color);

    }
}
```



String

String is not a data type, instead it is a predefined class in Java Class Library

- Google "Java 11 API" and find the **String** class in the Java Class Library
- Actual Representation of String - String s = new String("Sample Text"); - Demonstrate here
- Shortcut Representation of String - String s = "Sample Text"; - Demonstrate here
- Concatenate two strings
  - using '+' operator - Demonstrate here
- Predefined methods of String class - Out of all the predefined methods of Strings, the below are the methods which are useful as part of Selenium Automation:
  - Using **equals()** method to compare two strings - Demonstrate here
  - Using **length()** method to find the length of String literal text - Demonstrate here
  - Using **substring()** method to retrieve the portion from the actual String text - Demonstrate here
  - Using **trim()** to remove the spaces before and after the string text - Demonstrate here
  - Using **indexOf()** to check whether the provided text is in the provided paragraph. - Demonstrate here
    - Returns -1 in case the provided text is not available
  - Using **split()** method to split the text into different parts based on the provided text, symbol or space.

Primitive Data Types	Non-Primitive Data Types
Non-Object	Object
int short byte long double, float char boolean	Arrays int[] a = new int[3];

```java
public class Demo {

    public static void main(String[] args) {

        String firstname = "Arun";
        String lastname = "motoori";

        System.out.println("Arun".equals(firstname));  // true


    }

}
```

```java
public class Demo {

    public static void main(String[] args) {

        String firstname = "Arun";
        String lastname = "motoori";

        System.out.println("Arun".equals(lastname));  // We should not use == for String comparision



    }

}
```

Array	String
• lenth predefined variable	length() method

Demo.java

```java
public class Demo {

    public static void main(String[] args) {

        String name = "Arun Motoori";

        System.out.println(name.substring(5));



    }

}
```

Conscle

```
<terminated> Demo [Java Application] C:\Program
Motoori
```

```java
public class Demo {

    public static void main(String[] args) {

        String name = "Arun Motoori";

        System.out.println(name.substring(5,9));



    }

}
```

Demo.java

```java
public class Demo {

    public static void main(String[] args) {

        String name = "      Arun Motoori        ";

        System.out.println(name.length()); //26

        String trimmedname = name.trim();

        System.out.println(trimmedname.length());  //12


    }

}
```

Conscle

```
<terminated> Demo [Ja
26
12
```

Demo.java

```java
public class Demo {

    public static void main(String[] args) {

        String name = "  Arun Motoori  ";  // With spaces - 16
        String withoutbeforeandafterspaces = name.trim();  //Arun Motoori //12

        System.out.println(name.length());

        System.out.println(withoutbeforeandafterspaces.length());


    }

}
```

```java
public class Demo {

    public static void main(String[] args) {

        String para = "This is a sample text written by Arun Motoori";

        System.out.println(para.indexOf("Arun")); //33

    }

}
```

```java
public class Demo {

    public static void main(String[] args) {

        String para = "This is a sample text written by Arun Motoori";

        System.out.println(para.indexOf("Karan")); // -1

    }

}
```

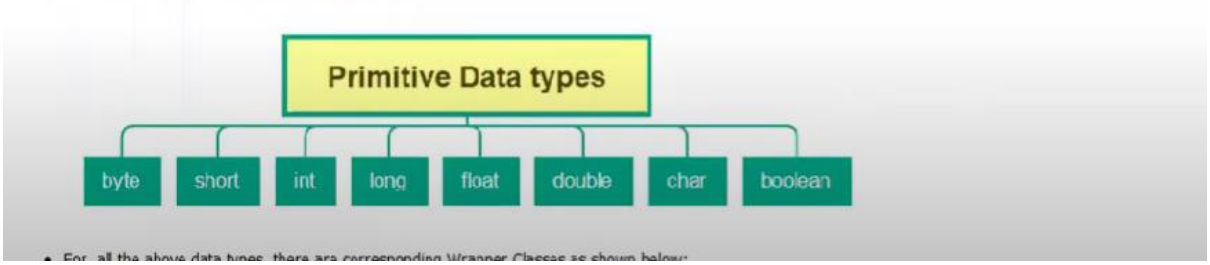
Demo.java

```java
public class Demo {

    public static void main(String[] args) {

        String para = "This-is-a-sample-text-written-by-Arun-Motoori";

        String[] a = para.split("-");

        for(String i:a) {

            System.out.println(i);

        }

    }

}
```

Wrapper Classes and Primitive Data types

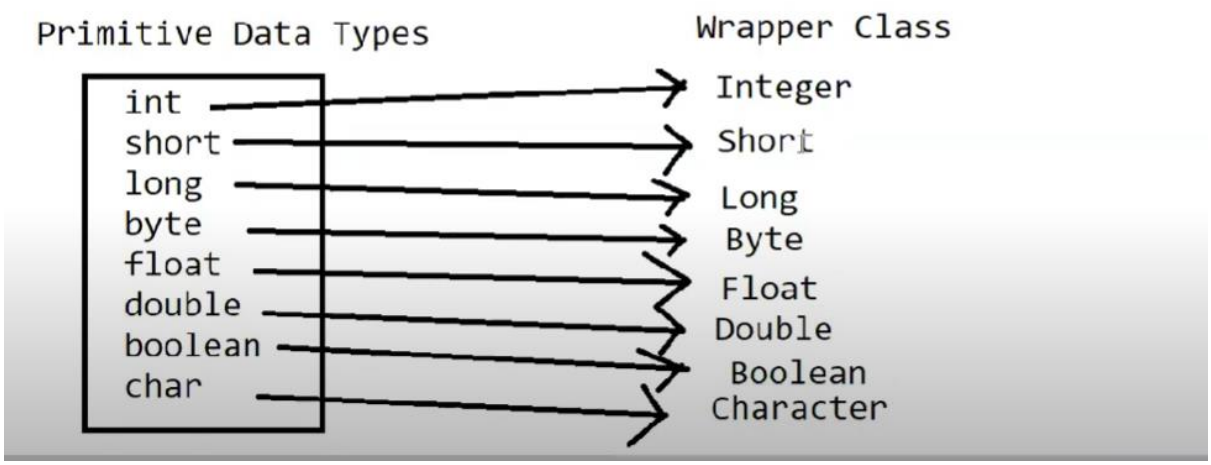
In order to use primitive data types as Objects, we have to use Wrapper Classes which help us in converting the primitve data types into objects.

- The below are the different primitive data types:



- For all the above data types, there are corresponding Wrapper Classes as shown below:

Wrapper Classes and Primitive Data Types



Is Java 100% Object Oriented Programmng langauge?

No

Why it is not 100%?

Primitive Data Types

How to convert primtive data type values to Wrapper class objects?

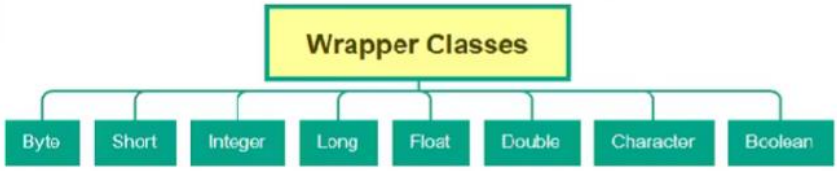
There may be some situations where
we have to use primitive data types also as objects

Solution: Wrapper Classes

```java
public class Demo {

    public static void main(String[] args) {

        int a = 5;

        Integer aobj;

        aobj = a;

        System.out.println(a);
        System.out.println(aobj);

    }

}
```

```java
public class Demo {

    public static void main(String[] args) {

        char a = 's';

        Character aobj = a;

        System.out.println(a);
        System.out.println(aobj);

    }

}
```

- For, all the above data types, there are corresponding Wrapper Classes as shown below:



- Demonstrate a program which uses Wrapper Classes for converting the primitive data types into Objects and Objects into primitive data types - Demonstrate here
- String and using Wrapper classes for conversion to appropriate data type

```java
public class Demo {

    public static void main(String[] args) {

        String a = "100"; //text

        System.out.println(a-15); //10015


    }

}
```

```java
public class Demo {

    public static void main(String[] args) {

        String a = "100"; //text

        //Converting string text into number text
        int anum = Integer.parseInt(a);

        System.out.println(anum+15); //115


    }

}
```

```java
public class Demo {

    public static void main(String[] args) {

        String a = "123.456";  //String text

        double b = Double.parseDouble(a);

        System.out.println(b+20);  //143.456


    }

}
```

Demo.java

```
1  public class Demo {
2
3      public static void main(String[] args) {
4
5          String a = "123ABC";  //String text
6
7          String b = a.substring(0,3);  //"123"
8
9          int num = Integer.parseInt(b);
10
11         System.out.println(a+15);
12
13
14     }
15
16 }
17
```

Conscle

```
<terminated> Demo [Java Applicat
123ABC15
```

Demo.java

```
1  public class Demo {
2
3      public static void main(String[] args) {
4
5          String a = "123ABC";  //String text
6
7          String b = a.substring(0,3);  //"123"
8
9
10         int num = Integer.parseInt(b);
11
12         System.out.println(num 15);
13
14
15     }
16
17 }
18
```

Conscle

```
<terminated> Demo [Java Applicatior
138
```

Java (Part 6) - Constructors, this keyword, Overloading, Packages, Inheritance and Overriding

Constructors

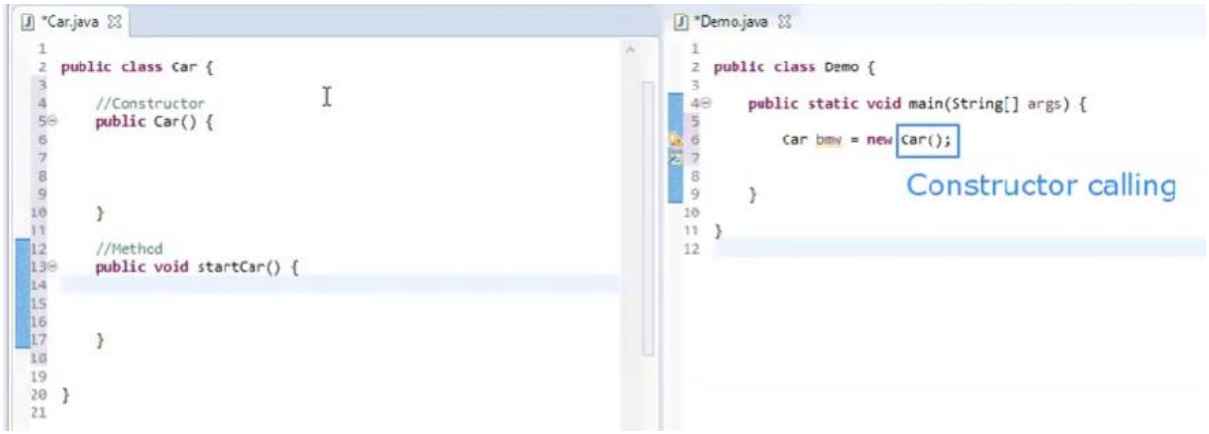
Constructors are similar to methods, but have the below differences:

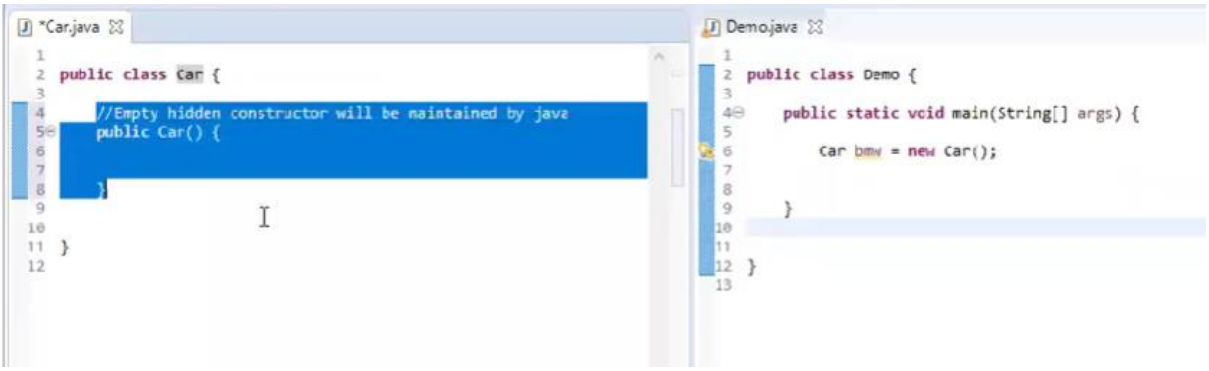
- Demonstrate the constructors having the below qualities here
  - Constructors have the same name as Class name
  - Constructors are automatically called when an object is created for the Class
  - Constructors won't have any return type - Return types like void, int etc won't be available for constructors
  - Empty hidden Constructor will be called, when an object is created for the Class not specified with explicit constructors
- Constructors simplify the initialization of variables
  - Demonstrate initialization of variables without using constructors - Demonstrate here
  - Demonstrate initialization of variables with constructors - Demonstrate here

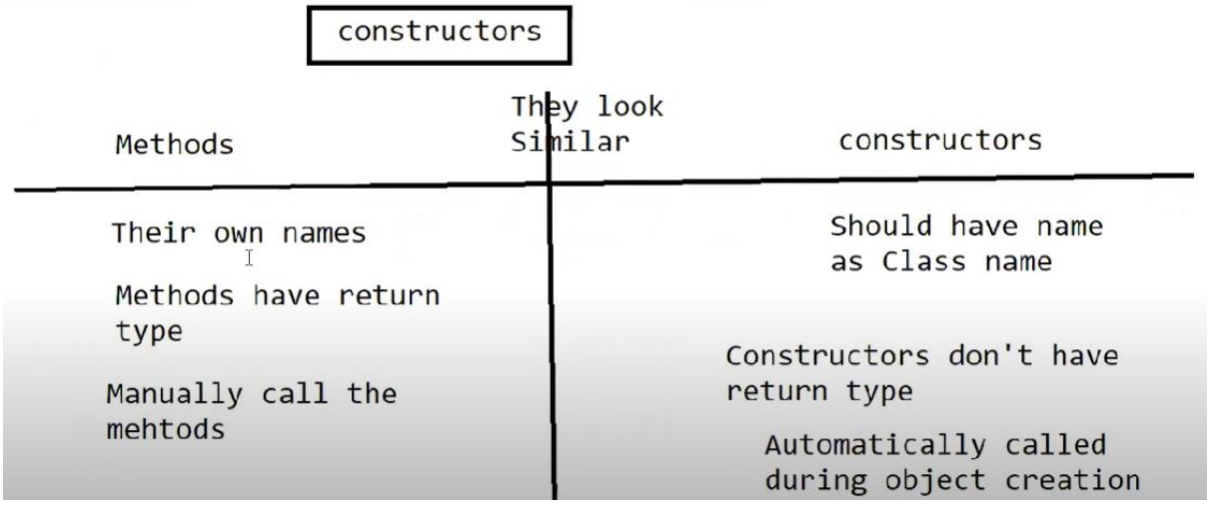
this keyword

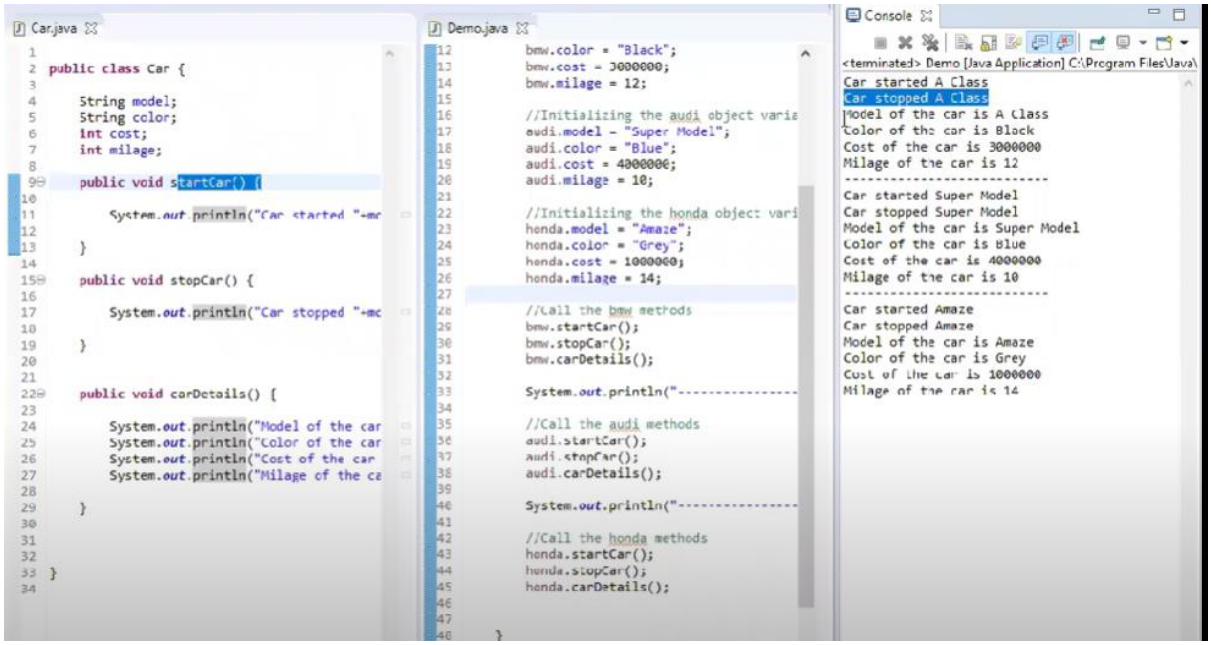
The purpose of the this keyword is to differentiate the instance variable with the parameterized variables of methods/constructors.

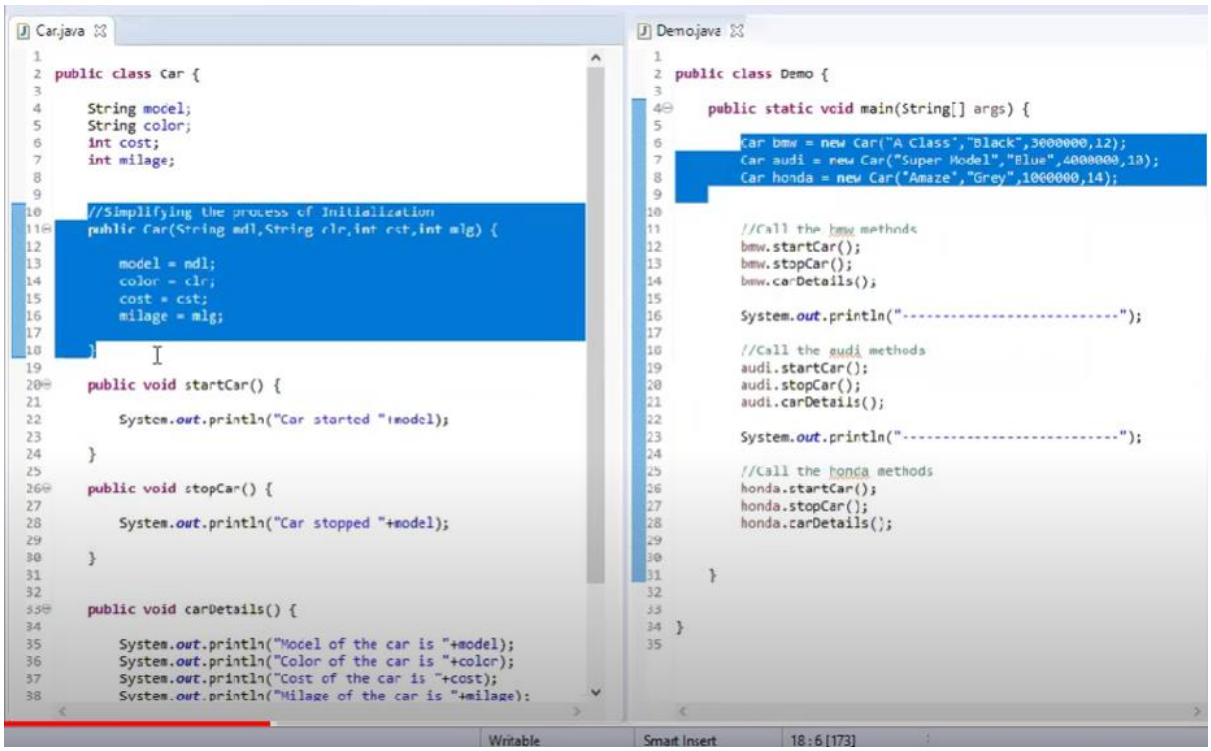
- Using this keyword with **Methods**
  - Demonstrate the program which dont use this keyword - Demonstrate here
  - Demonstrate the advantage of using this keyword with methods - Demonstrate here
- Using this keyword with **Constructors**
  - Generally required for constructors, as constructors are automatically called when objects are created and there by all the required variables will be initialized automatically
  - Similar to methods.


*Car.java
public class Car {
//Constructor
public Car() {
}
//Method
public void startCar() {
}
}
*Demo.java
public class Demo {
public static void main(String[] args) {
Car bmw = new Car();
}
}
Constructor calling


*Car.java
public class Car {
//Empty hidden constructor will be maintained by java
public Car() {
}
}
Demo.java
public class Demo {
public static void main(String[] args) {
Car bmw = new Car();
}
}


constructors
They look Similar
Methods
Their own names
Methods have return type
Manually call the mehtods
constructors
Should have name as Class name
Constructors don't have return type
Automatically called during object creation





this keyword:

Car.java

```java
public class Car {

    //Instance variables
    String model;
    String color;
    int cost;
    int milage;


    //Simplifying the process of Initialization
    public Car(String model,String color,int cost,int milage) {

        this.model = model;
        this.color = color;
        this.cost = cost;
        this.milage = milage;

    }

    public void startCar() {

        System.out.println("Car started "+model);

    }

    public void stopCar() {

        System.out.println("Car stopped "+model);

    }


    public void carDetails() {

        System.out.println("Model of the car is "+model);
        System.out.println("Color of the car is "+color);
        System.out.println("Cost of the car is "+cost);
```

Demo.java

```java
public class Demo {

    public static void main(String[] args) {

        Car bmw = new Car("A Class","Black",3000000,12);
        Car audi = new Car("Super Model","Blue",4000000,10);
        Car honda = new Car("Amaze","Grey",1000000,14);


        //Call the bmw methods
        bmw.startCar();
        bmw.stopCar();
        bmw.carDetails();

        System.out.println("--------------------------");

        //Call the audi methods
        audi.startCar();
        audi.stopCar();
        audi.carDetails();

        System.out.println("--------------------------");

        //Call the honda methods
        honda.startCar();
        honda.stopCar();
        honda.carDetails();


    }

}
```

*Car.java

```java
public class Car {

    //Instance variables
    String model;
    String color;
    int cost;
    int milage;

    public void sample(String model,String color,int cost,int milage) {

        this.model = model;
        this.color = color;
        this.cost = cost;
        this.milage = milage;

    }


    //Simplifying the process of Initialization
    public Car(String model,String color,int cost,int milage) {

        this.model = model;
        this.color = color;
        this.cost = cost;
        this.milage = milage;

    }

    public void startCar() {

        System.out.println("Car started "+model);

    }

    public void stopCar() {

        System.out.println("Car stopped "+model);
```

Overloading

Can we create multiple methods having same name inside the same class?

Yes, it is possible

methodA

methodA

We can multiple mehtods having same name inside the same class

methodA

We can multiple mehtods having same name inside the same class
When these methods differ in terms of

- Number of paramters
- Different type of parameter
- Order of parameters

*Car.java

```
1
2  public class Car {
3
4      public void methodA() {
5
6
7
8      }
9
10     public void methodA(int a) {
11
12
13
14     }
15
16     public void methodA(double a) {
17
18
19
20     }
21
22     public void methodA(int a,int b) {
23
24
25
26     }
27
28     public void methodA(int a,char b) {
29
30
31
32     }
33
34     public void methodA(char b,int a) {
35
36
37
38     }
39
```

*Car.java

```
1
2  public class Car {
3
4      public Car() {
5
6
7      }
8
9      public Car(int a) {
10
11
12     }
13
14     public Car(double a) {
15
16
17     }
18
19     public Car(int a,double b) {
20
21
22     }
23
24
25     public Car(double b,int a) {
26
27
28     }
29
30 }
31
```

Packages

Packages are created to group related classes/interfaces/other files.

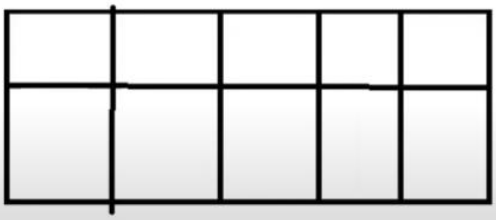
- We generally group things to organize them better for locating them easily.
- Default package - Create a new Java project say Facebook and Create a new Java Class say 'FacebookLogin' and observed that a default package will be created. - view here
- Package creation - Create a new Java project say Facebook and group the Classes under various packages - view here
- Demonstrate - Accessing instances variables and methods from other class which is under the same package
- Demonstrate - Importing the Classes in the other packages while accessing the instance variables and methods created in the Classes which are under other packages
- Demonstrate - Using * in the import statements to import all the classes in the package instead of importing a single class every time

Tools Shapes Colors

Packages

Organize the related things

Find the books easily



Naming Convention

Project Name

FacebookDemoProject

Class Name

DemoOne

Method Name

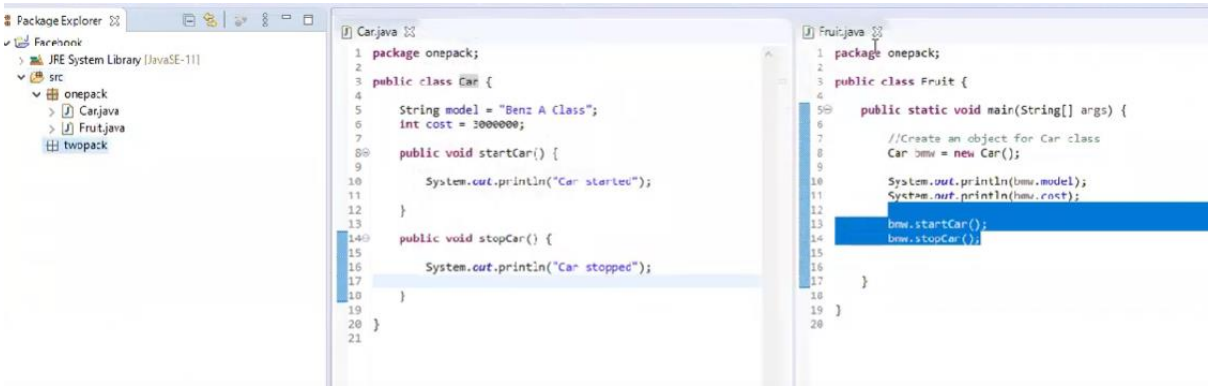
methodSumTwo

Packages

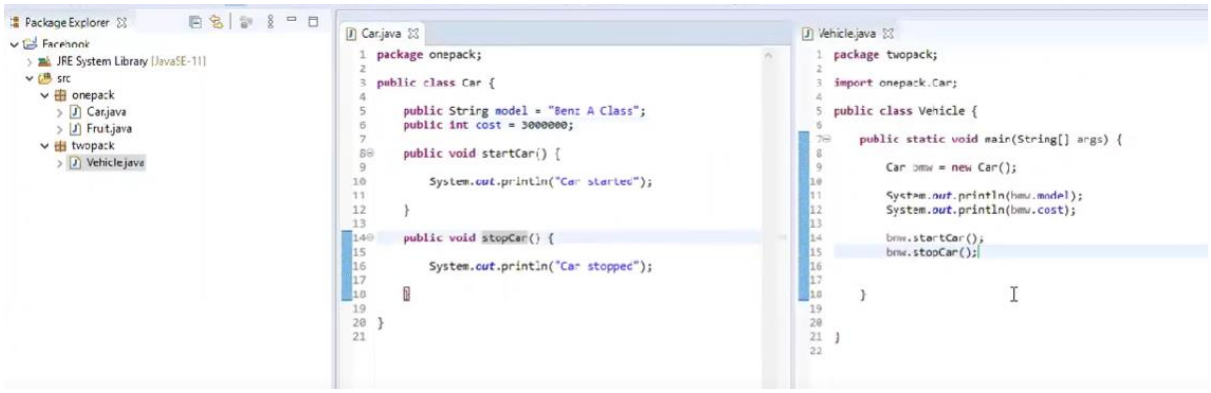
lower case

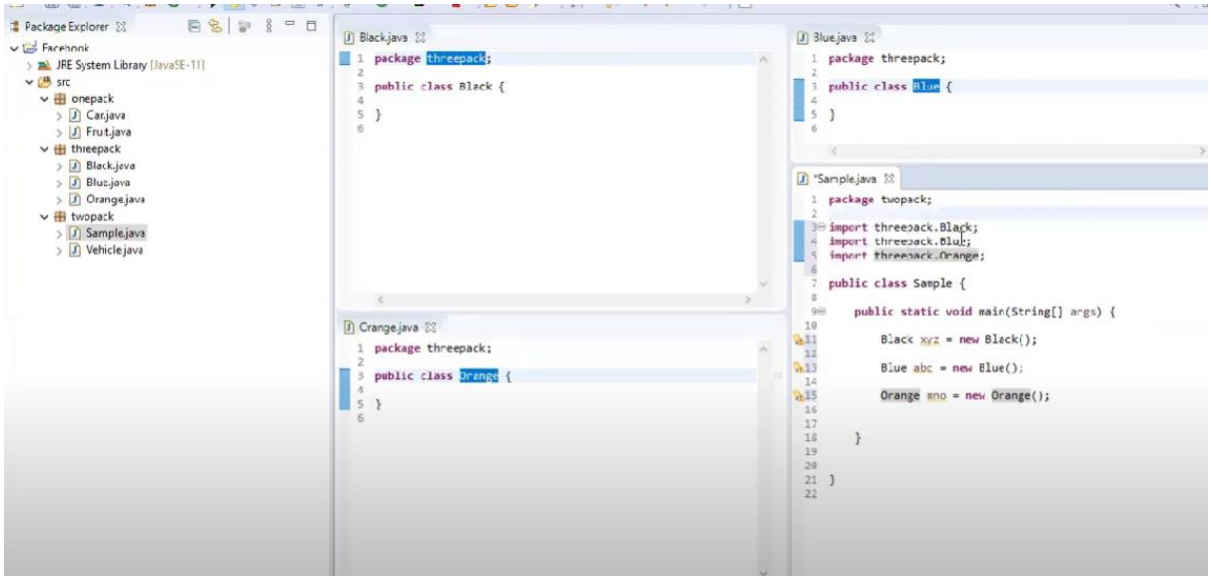
demopack

Classes in same package



Classes in 2 packages





Java (Part 7) - Inheritance

Inheritance

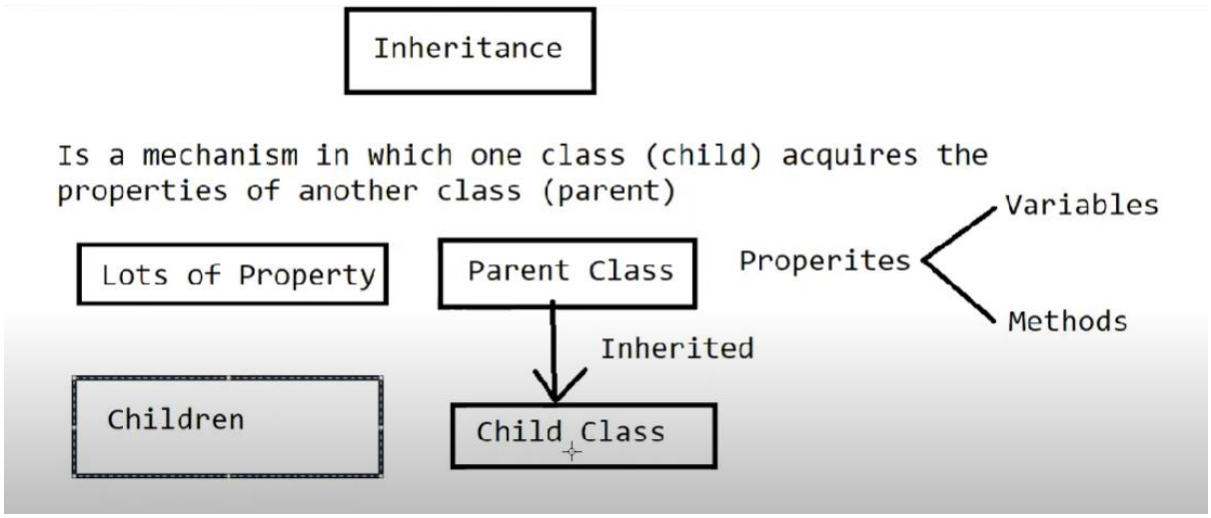
Inheritance is a mechanism in which one class acquires the properties (i.e. variables and methods) of another class

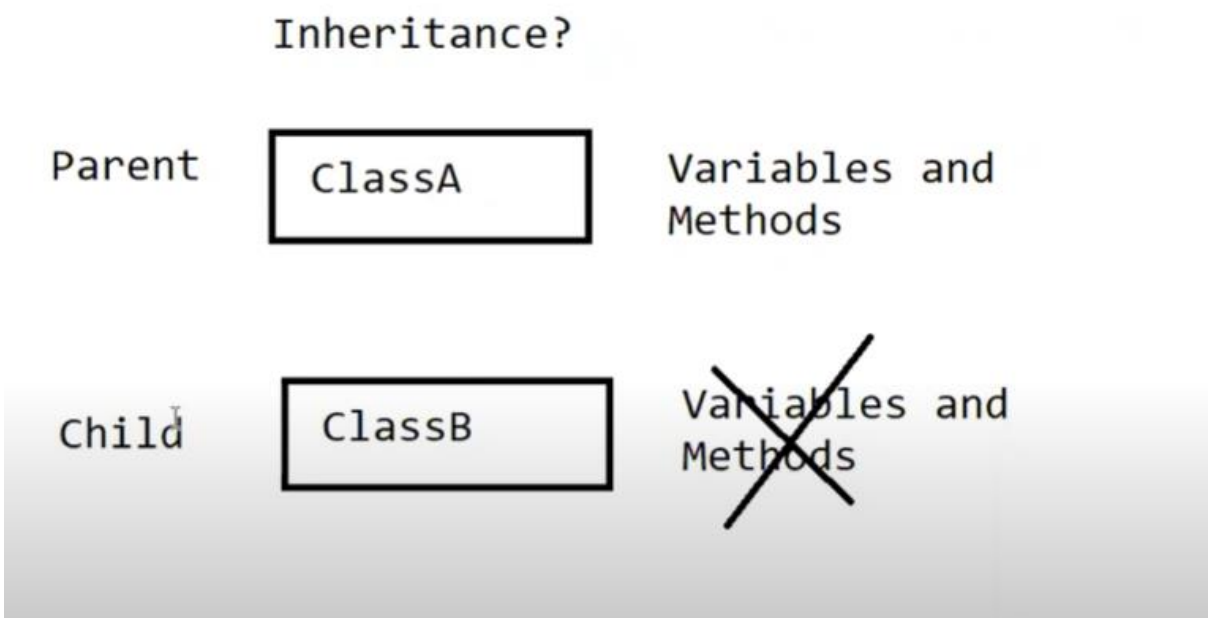
- The purpose of this Inheritance is to use the properties (i.e. methods and variables) inside a class instead of recreating the same properties again in new class.
- Child class acquires the properties (i.e. variables and methods) of Parent Class.
- Child class uses **extends** keyword to inherit the properties from parent class
- Demonstrate a child class which inherits the properties from Parent Class - Demonstrate here
  - Child class can have specific properties (i.e. variables and methods) which are not available in the parent class
  - Object created for parent class can access the variables and methods that are created in parent class only. It cannot access the child class properties.
  - Object created for child class which is inheriting the parent class can access the variables and methods of both parent class and child class.

Overriding

When a method in the Child class (i.e. sub-class) is duplicate of a method in Parent class (i.e. super-class) , then the method in the sub-class is said to override the method in super-class.

- When we create an Object for Sub-class and call the overridden method, the method in the sub-class will be called - Demonstrate here
- Even though the name of the method in the sub-class has the same name as a method in super-class, if the type of parameters or number of parameters, then the method in the sub-class will overload the method in super-class instead of overriding
- Constructors cannot be overridden as the name of the constructor needs to be same as the name of the Class.





Dog.java

```java
//Parent Class - Dog
public class Dog {

    String breed;
    String color;
    String size;

    public void eat() {

        System.out.println(breed+" Dog is eating");

    }

    public void sleep() {

        System.out.println(breed+" Dog is sleeping");

    }

    public void bark() {

        System.out.println(breed+" Dog is barking");

    }

}
```

*Demo.java

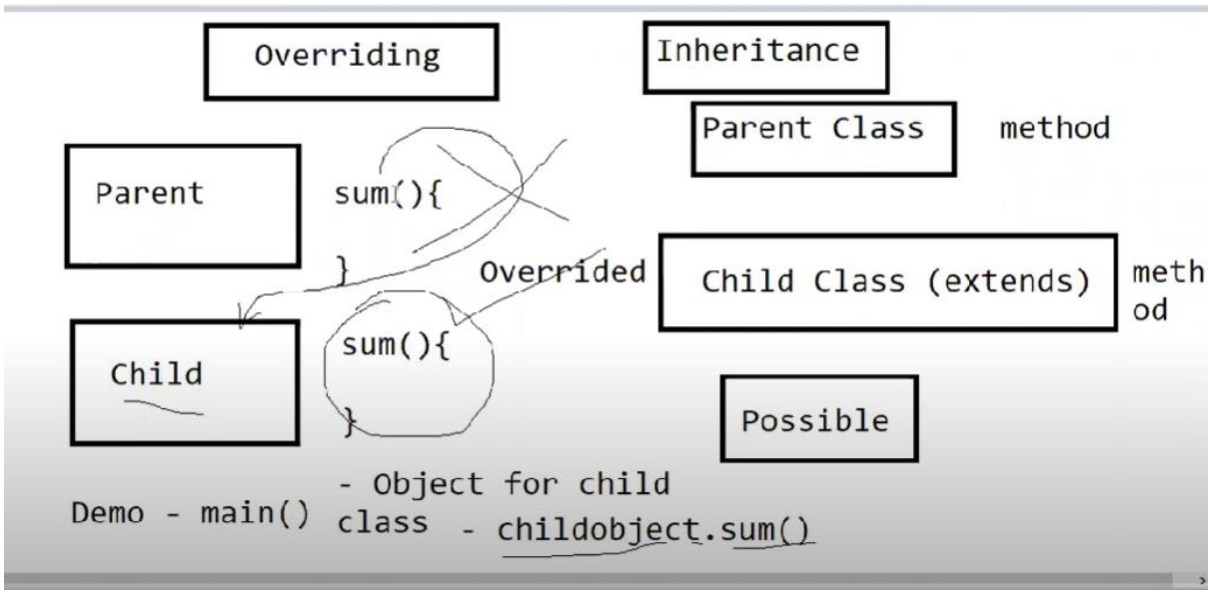
```java
public class Demo {

    public static void main(String[] args) {

        //Create Objects
        Dog d = new Dog();

        d.breed = "Shepard";
        d.color = "Black";
        d.size = "Big";

        d.eat();
        d.sleep();
        d.bark();


    }

}
```

Pug.java

```java
//Child Class - Pug
public class Pug extends Dog {

    String cutenesslevel;

    public void bite() {

        System.out.println("Pug dog bites");

    }


}
```

Demo.java

```java
public class Demo {

    public static void main(String[] args) {

        //Create Object for Parent Class
        Dog d = new Dog();

        d.breed = "Shepard";
        d.color = "Black";
        d.size = "Big";


        d.eat();
        d.sleep();
        d.bark();

        System.out.println("--------------------");

        //Create Object for child class
        Pug p = new Pug();

        p.breed = "Pug";
        p.color = "Brown";
        p.size = "Small";
        p.cutenesslevel = "High";

        p.eat();
        p.sleep();
        p.bark();
        p.bite();


    }

}
```



Dog.java

```
1
2  public class Dog {
3
4      public void eat() {
5
6          System.out.println("Shepard dog is eating");
7
8      }
9
10 }
11
```

*Pug.java

```
1
2  public class Pug extends Dog{
3
4      //This overrides the method of parent class
5      //Method Overriding
6
7      public void eat() {
8
9          System.out.println("Pug dog is eating");
10
11     }
12
13 }
14
```

Demo.java

```
1
2  public class Demo {
3
4      public static void main(String[] args) {
5
6          Pug p = new Pug();
7
8          p.eat();
9
10         Dog d = new Dog();
11
12         d.eat();
13
14     }
15
16 }
17
```

Method Overriding

Constructors?
No

Create a duplicate method in child class

duplicate method
in child class
will override the
method in parent
class

replace

Dog.java

```
1
2  public class Dog {
3
4      public Dog() {
5
6          System.out.println("Shepard dog");
7
8      }
9
10 }
11
```

Pug.java

```
1
2  public class Pug extends Dog{
3
4      public Pug() {
5
6          System.out.println("Pug dog");
7
8      }
9
10
11
12 }
13
```

Constructor
Overriding is
not possible

*Dog.java

```
1
2  public class Dog {
3
4
5      public void sum() {
6
7          System.out.println("Shepard Sum is 0");
8
9      }
10
11     //Overloading
12     public void sum(int a) {
13
14         System.out.println("Sum is: "+a);
15
16     }
17
18
19     //Overloading
20     public void sum(int a,int b) {
21
22         System.out.println("Sum is: "+(a+b));
23
24     }
25
26
27
28
```

*Pug.java

```
1
2  public class Pug extends Dog{
3
4      //This method is overriding the same parent class method
5      public void sum() {
6
7          System.out.println("Pug Sum is 0");
8
9      }
10
11
12
13 }
14
```

Dog.java

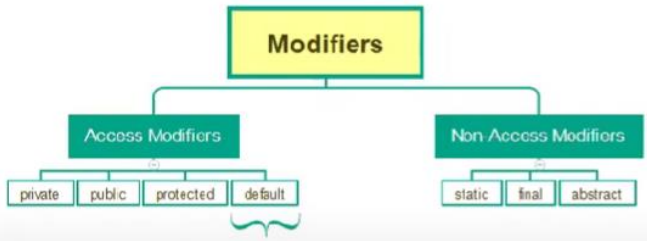
```java
public class Dog {

    public void sum() {

        System.out.println("Dog sum");

    }



}
```

*Pug.java

```java
public class Pug extends Dog{

    //Overloading
    public void sum(int a) {

        System.out.println("Pug sum");

    }

}
```

Modifiers

- Modifiers in Java can be categorized as below:



When we don't specify any modifier

public Access Modifier

- Classes/variables/methods specified with 'public' access modifier can be accessed directly by the classes which are in the same package - Demonstrate
- Classes/variables/methods specified with 'public' access modifier can be accessed by the classes outside the package after importing the classes - Demonstrate

Modifiers

main()
Method
From where the program execution starts

class
Keyword for creating the classes in Java

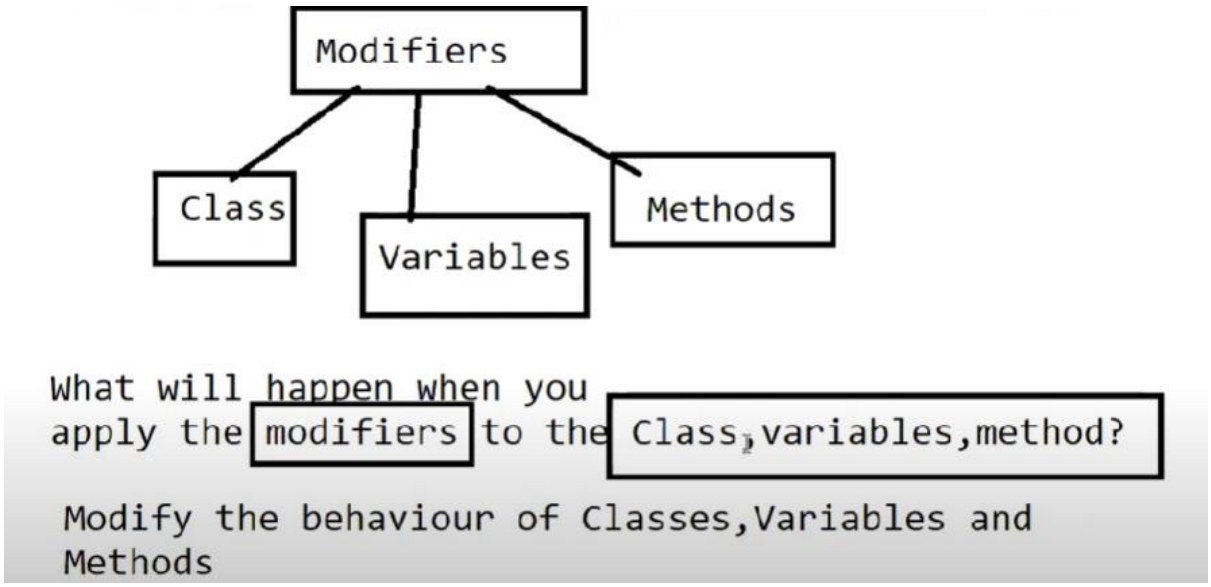
Demo
Class Name

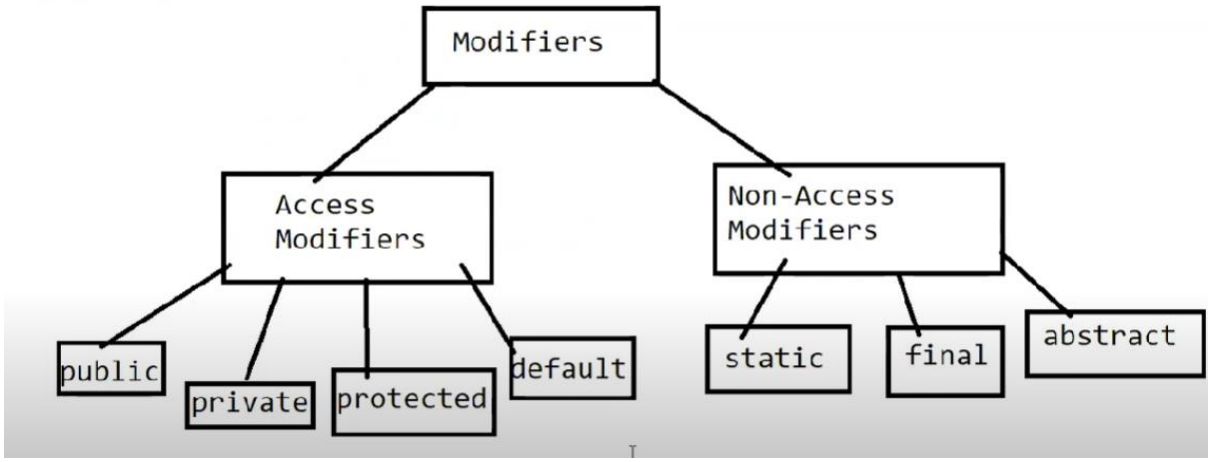
public ?

static ?

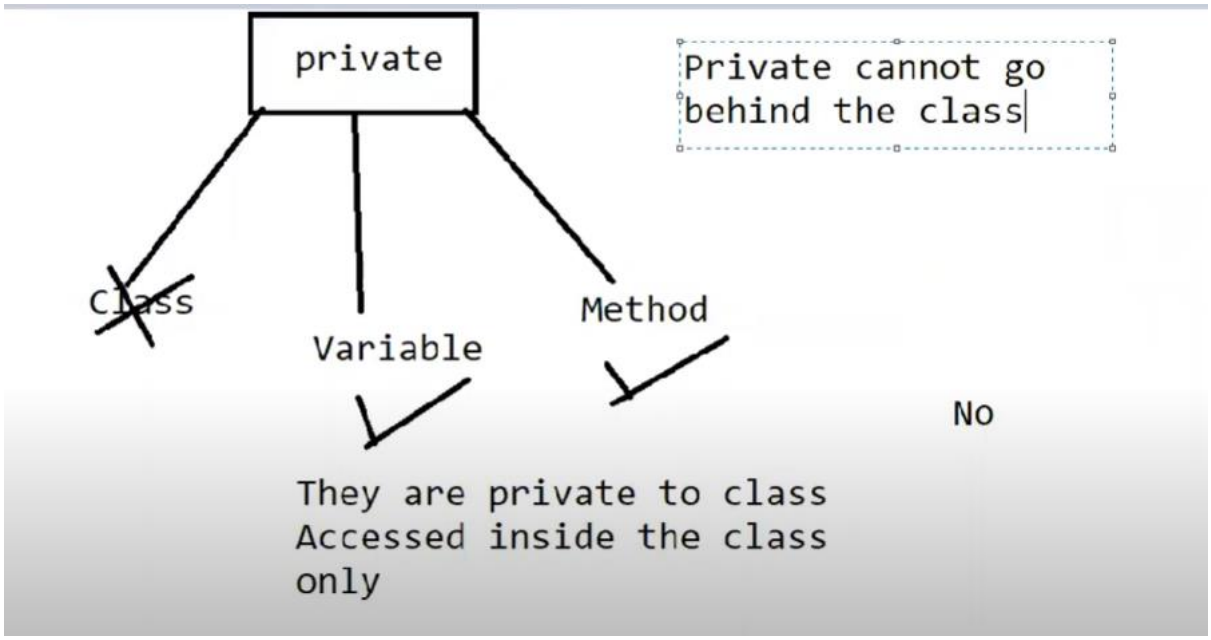
Modifiers

void
return type





We cannot use private with class



```java
package login;

public class Login {

    //private variables
    private static String username = "arunmotoori";
    private static String password = "1234";

    //private method
    private static void loginToApplication() {

        System.out.println("Logged into the application using username: " +username+" and password: "+password);

    }

    public static void main(String[] args) {

        loginToApplication();

    }

}
```

Login.java

```java
package login;

public class Login {

    //public variables
    public String username = "arunmotoori";
    public String password = "1234";

    //public method
    public void loginToApplication() {

        System.out.println("Logged into the application using usernam

    }


}
```

Logout.java

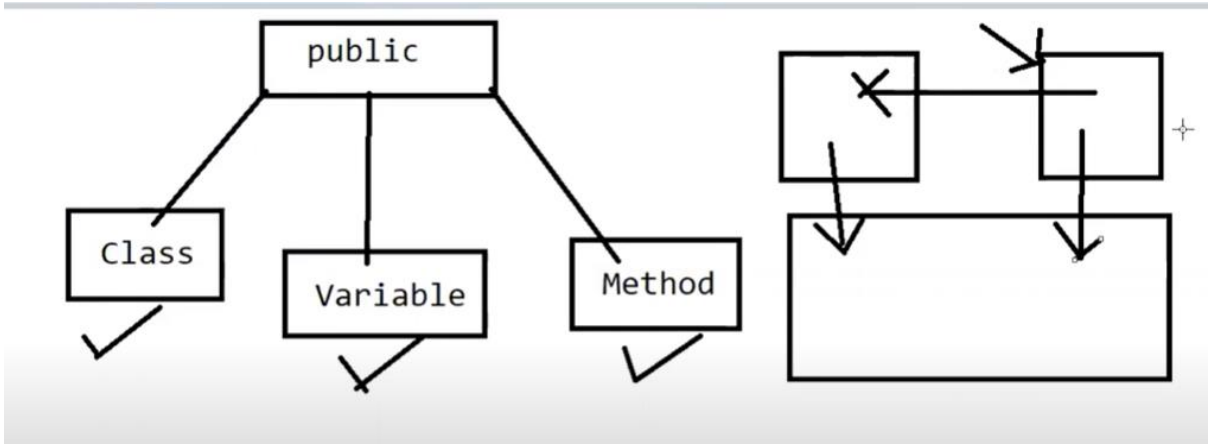
```java
package login;

public class Logout {

    public void logoutFromApplication() {

        Login l = new Login();

        System.out.println(l.username);
        System.out.println(l.password);

        l.loginToApplication();


    }


}
```

Login.java

```java
package login;

public class Login {

    //public variables
    public String username = "arunmotoori";
    public String password = "1234";

    //public method
    public void loginToApplication() {

        System.out.println("Logged into the application using usernam

    }


}
```

AddPhotos.java

```java
package photos;

import login.Login;


public class AddPhotos {

    public void loginAndAddPhotos() {

        Login l = new Login();

        System.out.println(l.username);
        System.out.println(l.password);

        l.loginToApplication();

    }

}
```



private Access Modifier

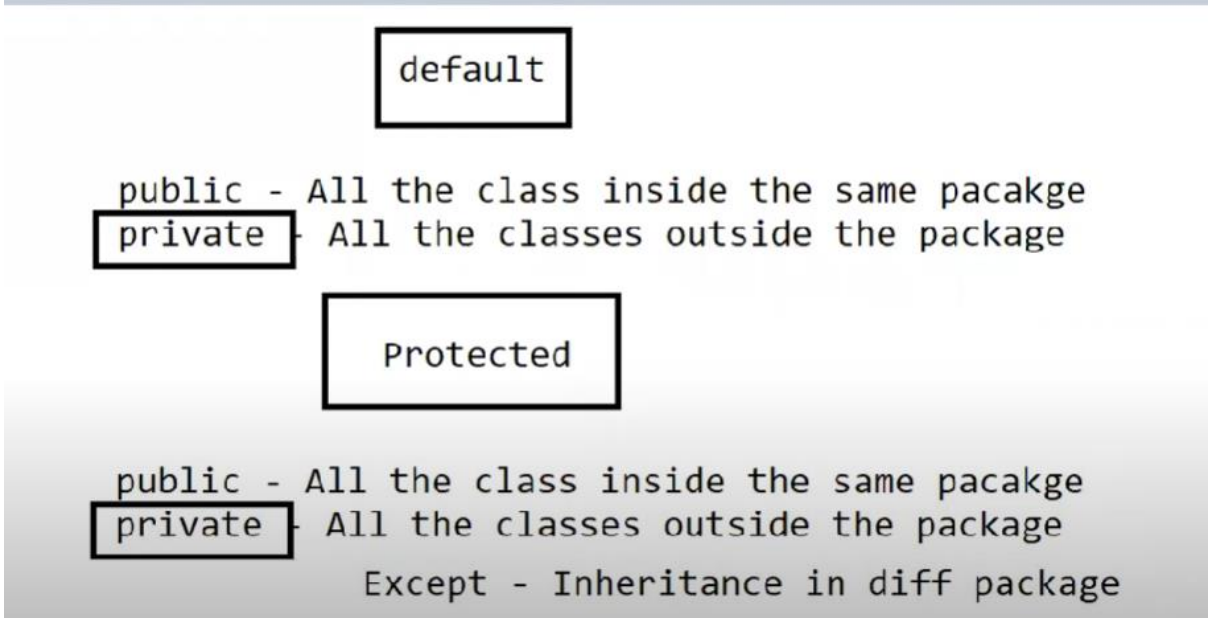
- Java classes cannot be specified with 'private' access modifier - Demonstrate
- variables/methods specified with 'private' access modifier can be accessed only with in the same class - Demonstrate

default Access Modifier

- When no modifier is specified before classes/variables/methods, then we name it as default modifier - Demonstrate
- default means public to all the classes inside the same package and private to the classes which are outside the package - Demonstrate

protected Access Modifier

- protected means public to all the classes inside the same package and private to all the classes which are outside the package except child classes
- Java classes cannot be specified with 'protected' access modifier - Demonstrate
- While accessing the protected variables/methods outside the packages using sub-classes, we don't have to create an object to access them as they are inherited variables and methods - Demonstrate

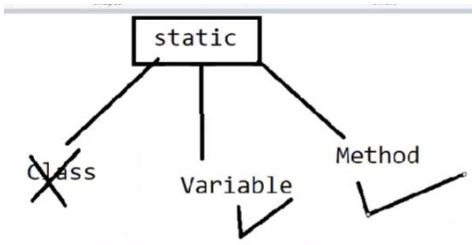


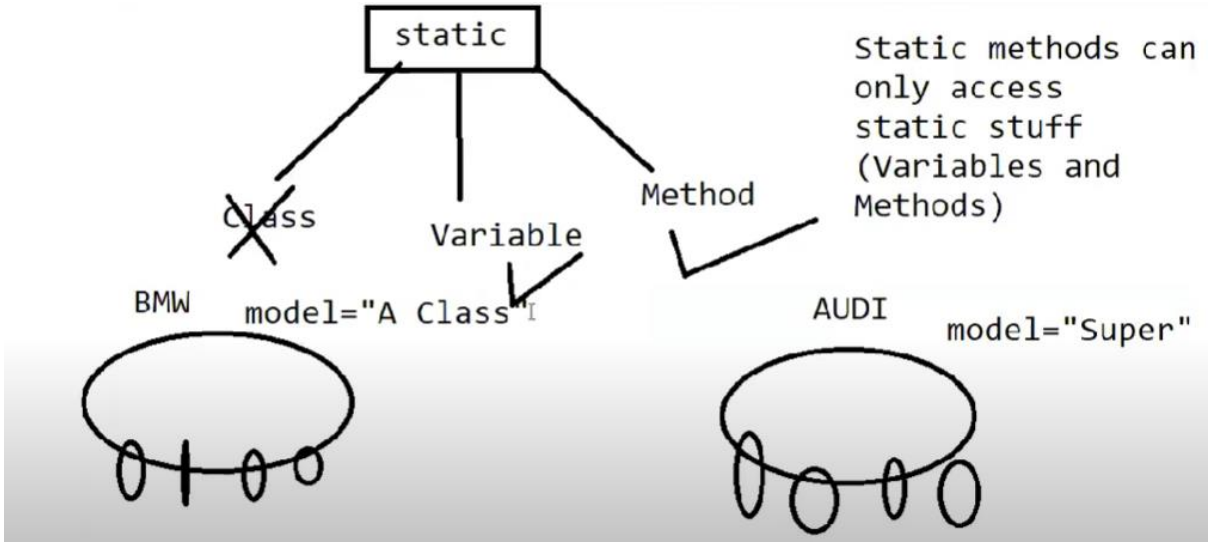
static Non Access Modifier

- Java classes cannot be specified with 'static' non-access modifier - Demonstrate
- Variables declared directly inside the class but outside the methods and are specified with 'static' modifier are known as static variables
- Memory allocated to the static variables is different from the memory allocated to the instance variables - view here
- static variables needs to be accessed with the help of Class name, as they belong to the Class memory - View here
- static variables are generally used to store common data, where as Object variables/Instance variables are used to store Object specific data.
  - wheels variable can be used as a static variable/class level variable as it has common data i.e. wheels count is 4 for all the cars in the market
  - Where as cost variable cannot be used as a static variable as its value changes from car to car, hence we use it as an Object variable/Instance variable.
- static can also be used with methods
- static can only access static stuff
  - You have to create object to overcome this

final Non-Access Modifier

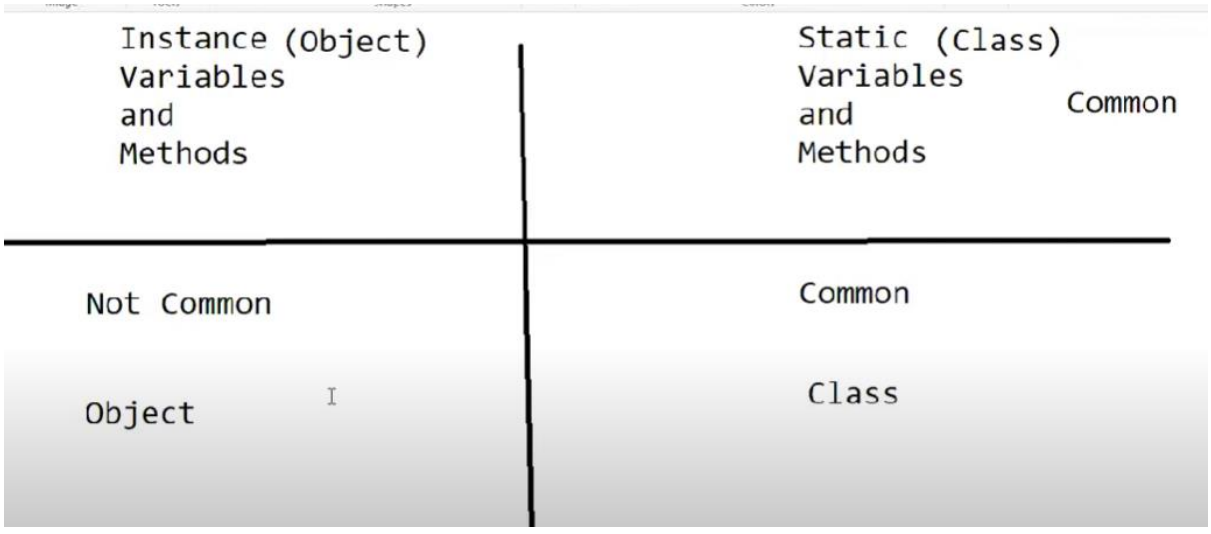
- The value of the variable cannot be changed on specifying it with final non-access modifier - Demonstrate here
- final modifier specified classes cannot be inherited/extended by other classes - Demonstrate here
- final modifier specified methods in a class cannot be overridden by its sub-classes - Demonstrate here





*Car.java

```
1
2  public class Car {
3
4      String model;
5      String color;
6      int cost;
7      double milage;
8      static int wheels;  //Common Variable
9
10
11
12
13 }
14
```



Car.java

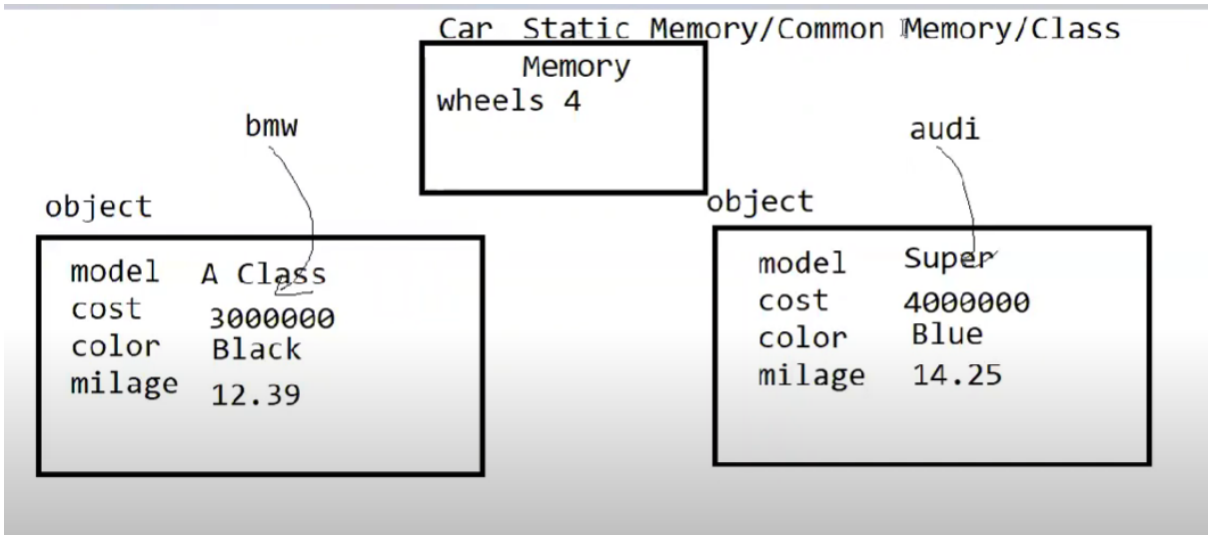
```java

public class Car {

    //Instance variables
    String model;
    String color;
    int cost;
    double milage;
    //Static variables
    static int wheels = 4;  //Common Variable

    //Instance methods
    public void startCar() {

        System.out.println(model+" car is starting");

    }

    public void stopCar() {

        System.out.println(model+" car is stopping");

    }

    //Static Methods
    public static void carWheels() {

      System.out.println("Number of wheels a car has is "+wheels);

    }


}

```

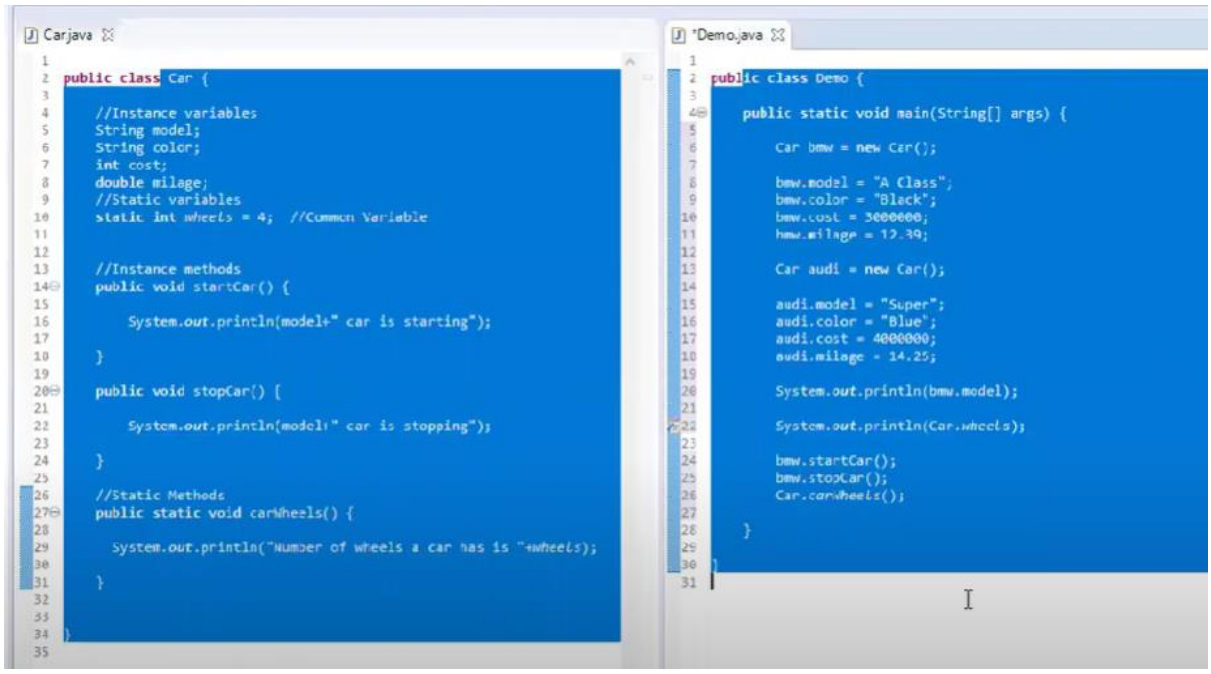
Car.java

```java

public class Car {

    //Instance variables
    String model;
    String color;
    int cost;
    double milage;
    //Static variables
    static int wheels = 4;  //Common Variable


    //Instance methods
    public void startCar() {

        System.out.println(model+" car is starting");

    }

    public void stopCar() {

        System.out.println(model+" car is stopping");

    }

    //Static Methods
    public static void carWheels() {

      System.out.println("Number of wheels a car has is "+wheels);

    }


}

```

*Demo.java

```java

public class Demo {

    public static void main(String[] args) {

        Car bmw = new Car();

        bmw.model = "A Class";
        bmw.color = "Black";
        bmw.cost = 3000000;
        bmw.milage = 12.39;

        Car audi = new Car();

        audi.model = "Super";
        audi.color = "Blue";
        audi.cost = 4000000;
        audi.milage = 14.25;



    }
}

```





Final:

Final class cannot be inherited

We cannot change value of final variable

```java
public class Dog {

    final String breed = "Shepard";

    public void sample() {

        breed = "Pug";

    }

}
```

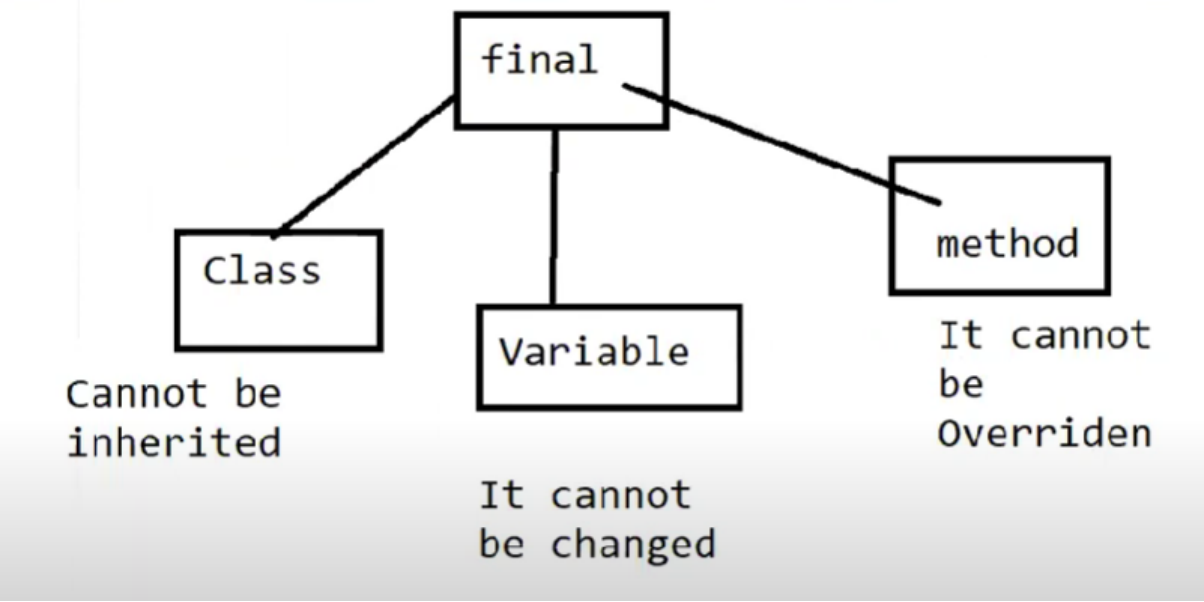
Cannot override a final method

Dog.java

```java
public class Dog {

    String breed = "Shepard";

    public final void sample() {

        System.out.println("Inside sample method of Dog Class");

    }

}
```

Pug.java

```java
public class Pug extends Dog {

    public void sample() {

        System.out.println("Inside sample method of Pug Class");

    }

}
```



abstract Non-Access Modifier

- variables cannot be specified with 'abstract' non-access modifier - Demonstrate
- On specifying a method with abstract modifier, we can just declare the method without implementing it - Demonstrate here
- Classes having at-least one abstract specified method must be specified as abstract
- Sub-Class inheriting the Super-Class needs to implement the abstract specified methods in Super-Class - Demonstrate here
  - Purpose of abstract methods - Used when the super-class dont have to implement everything, and when the sub-classes inheriting the super-class needs to implement them.
- Objects cant be created for abstract classes, we have to create a Sub-Class and access its variables/methods using Sub-Class object reference - Demonstrate here

Abstract methods cannot have method body

If a method is abstract , class must be abstract

Car.java

```java
public abstract class Car {

    public abstract void startCar();

}
```

Car.java

```java
public abstract class Car {


    //abstract method
    public abstract void startCar();

    //non-abstract method
    public void stopCar() {

        System.out.println("Car is stopping");

    }

}
```

Demo.java

```java
public class Demo {

    public static void main(String[] args) {



    }

}
```

Bmw.java

```java
public class Bmw extends Car{

    @Override
    public void startCar() {

        System.out.println("Car started in Bmw class");

    }


}
```

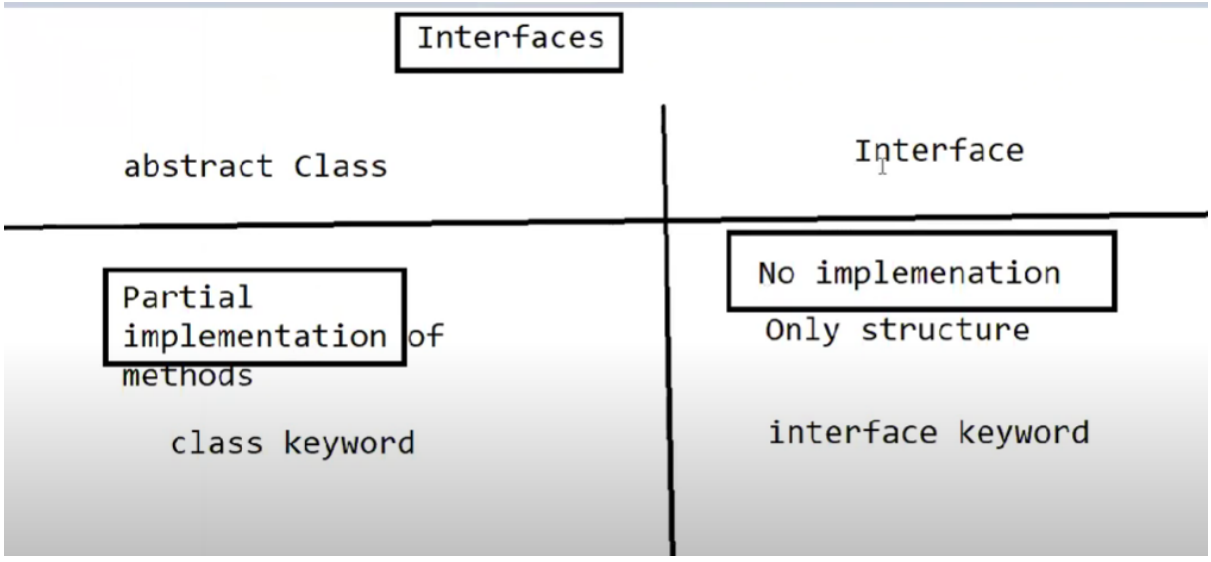
Java (Part 8) - Interfaces, Exception Handling and Handling Files

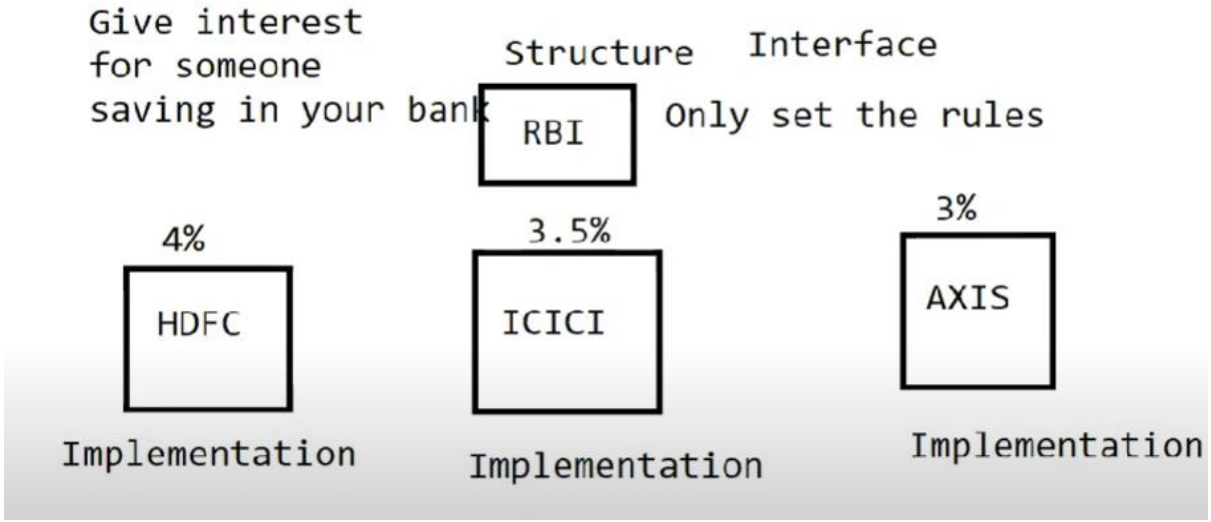
Interfaces

The purpose of an interface is to just to declare all the functionalities required before actually implementing them.

- Interfaces looks similar to Classes and are extensions of abstract classes
- Create an interface say 'Bank' in Eclipse IDE and create variables & methods inside it as shown here
- Variables in the interfaces are of static and final type
- In abstract classes, we can have both methods (i.e. implemented and non-implemented), where as in interfaces, we cannot implement any methods.
- Classes use **implements** keyword to implement any interface - Demonstrate here
- Classes implementing an interface can have their own specific methods apart from methods which are acquired from an interface - Demonstrate here
- Objects cannot be created for an interface - Demonstrate
- Object can be created for the Classes which are implementing the interfaces, for accessing interface defined methods and class specific methods - Demonstrate
- Follow the below steps to provide the access the interface specific methods and not to access the class specific methods
    - Create an object for the Class which is implementing the interface
    - Assign the object of the class to the interface reference variable
    - Using the interface reference variables, we can now access only the methods which are declared in the interface - Demonstrate here

```java
public abstract class Car {

    //abstract methods

    public abstract void startCar();

    public abstract void stopCar();

    //non-abstract method

    public void carDetails() {

        System.out.println("Inside car details method");

    }

}
```





```java
public interface Bank {

    String ACCOUNTTYPEONE = "Savings";  //By default static and final type
    String ACCOUNTTYPETWO = "Current";  //By default static and final type

    public void viewAcccuntBalance(); //By default abstract methods

    public void tranferFuncers();

    public void openFixedDeposit();

}
```

Bank.java | Demo.java | Car.java

```java
public abstract class Car {

    //abstract methods

    public abstract void startCar();

    public abstract void stopCar();

    //non-abstract method

    public void carDetails() {

        System.out.println("Inside car details method");

    }


}
```

BMW.java

```java
public class BMW extends Car {

    @Override
    public void startCar() {

        System.out.println("Inside Start car method");

    }

    @Override
    public void stopCar() {

        System.out.println("Inside stop car method");

    }

}
```

Bank.java

```java
public interface Bank {

    String ACCOUNTTYPEONE = "Savings";  //By default static and final
    String ACCOUNTTYPETWO = "Current";  //By default static and final

    public void viewAccountBalance(); //By default abstract methods

    public void tranferFunders();

    public void openFixedDeposit();


}
```

HDFC.java

```java
public class HDFC implements Bank{

    @Override
    public void viewAccountBalance() {

        System.out.println("Viewing Account Balance");

    }

    @Override
    public void tranferFunders() {

        System.out.println("Transferring Funds");

    }

    @Override
    public void openFixedDeposit() {

        System.out.println("Opening Fixed Deposit");

    }


}
```

Bank.java

```java
public interface Bank {

    String ACCOUNTTYPEONE = "Savings";  //By default static and final
    String ACCOUNTTYPETWO = "Current";  //By default static and final

    public void viewAccountBalance(); //By default abstract methods

    public void tranferFunders();

    public void openFixedDeposit();


}
```

*HDFC.java

```java
public abstract class HDFC implements Bank{

    @Override
    public void viewAccountBalance() {

        System.out.println("Viewing Account Balance");

    }

    @Override
    public void tranferFunders() {

        System.out.println("Transferring Funds");

    }


}
```

*HDFChild.java

```java
public class HDFChild extends HDFC{

    @Override
    public void openFixedDeposit() {
        // TODO Auto-generated method stub

    }

}
```

Dog.java

```java
//Parent Class
public class Dog {

    String breed;
    String color;
    String size;

    public void eat() {

        System.out.println(breed+" dog is eating");

    }

    public void sleep() {

        System.out.println(breed+" dog is sleeping");

    }

    public void bark() {

        System.out.println(breed+" dog is barking");

    }
}
```

*Pug.java

```java
public class Pug extends Dog {

    int pugCost;

     public void eat() {

        System.out.println("pug is eating");

    }

    public void feedPug() {

        System.out.println("Pug dog is feeded");

    }
```

*Demo.java

```java
public class Demo {

    public static void main(String[] args) {

        //Object of the Child class
        //Can be assigned to Parent Class
        Dog e = new Pug();

        e.eat();


    }
```

What will happen when you assign the child class object to the parent class ?

Only parent class variables and methods can be accessed

The child class methods will be called

Bank.java

```java
public interface Bank {

    String ACCOUNTTYPEONE = "SAVINGS";
    String ACCOUNTTYPETWO = "CURRENT";

    public void viewAccountBalance();
    public void transferFunds();
    public void openFixedDeposit();

}
```

HDFC.java

```java
public class HDFC implements Bank {

    @Override
    public void viewAccountBalance() {
        // TODO Auto-generated method stub

    }

    @Override
    public void transferFunds() {
        // TODO Auto-generated method stub

    }

    @Override
    public void openFixedDeposit() {
        // TODO Auto-generated method stub

    }

    public void calculateHDFCInterest() {


    }

}
```

Demo.java

```java
public class Demo {

    public static void main(String[] args) {

        Bank b = new HDFC();

        b.viewAccountBalance();
        b.transferFunds();
        b.openFixedDeposit();


    }

}
```

Exception Handling

Exception is nothing but an error which is occurred during runtime i.e. during program execution

- If an exception has occurred during program execution at any step, the steps which are after the exception wont be executed - Demonstrate here

try catch blocks

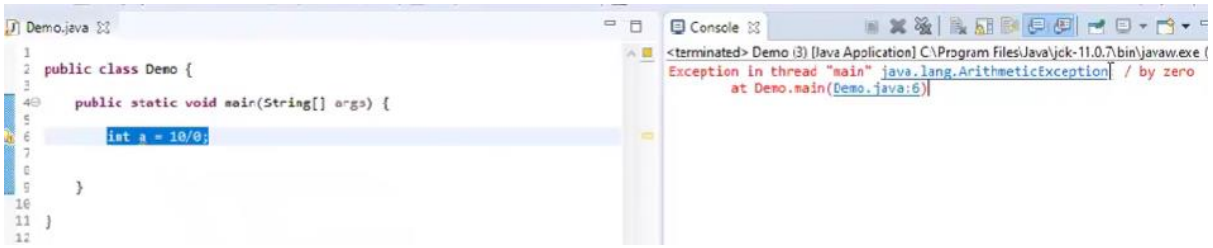
- We can handle the exceptions using the **try catch** blocks
  - Handling the exceptions is known as Exception Handling
  - Syntax: View here
  - Explain the flow of try catch block - view here
  - Demonstrate a program having code to handle the exception using try catch blocks - Demonstrate here
  - In the above Syntax image, 'Exception' is the Class name and 'e' is the object reference which can catch the exception (i.e. object) thrown from try block

Exception Handling

Exception? Erros whi

Compile Time Errors	Exceptions
Before Execution	Runtime (During Execution)

Error which are not identified during compile time and occur while the code is beging executing.



What is the problem of Exception?

Program will stop on getting an Exception

All the code which come after the Exception wont be executed

What if
I want the program to contine even after Exception?

Exception Handling

```
try{
    //Code having exception
} catch(Exception e){
    //Code to handle the exception
}
```

Exception Handling

try catch blocks

new ArithmeticException();

Java
Stop the program

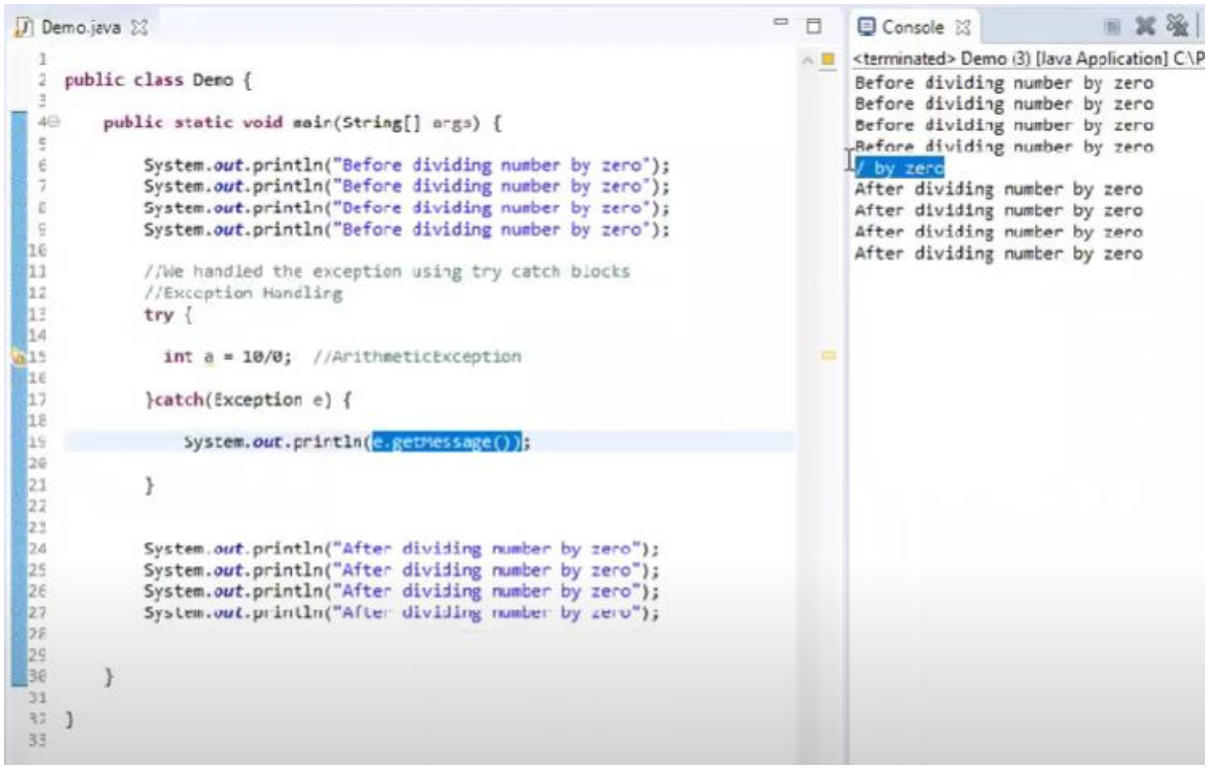
```
Exception e = new ArithmeticException();
```

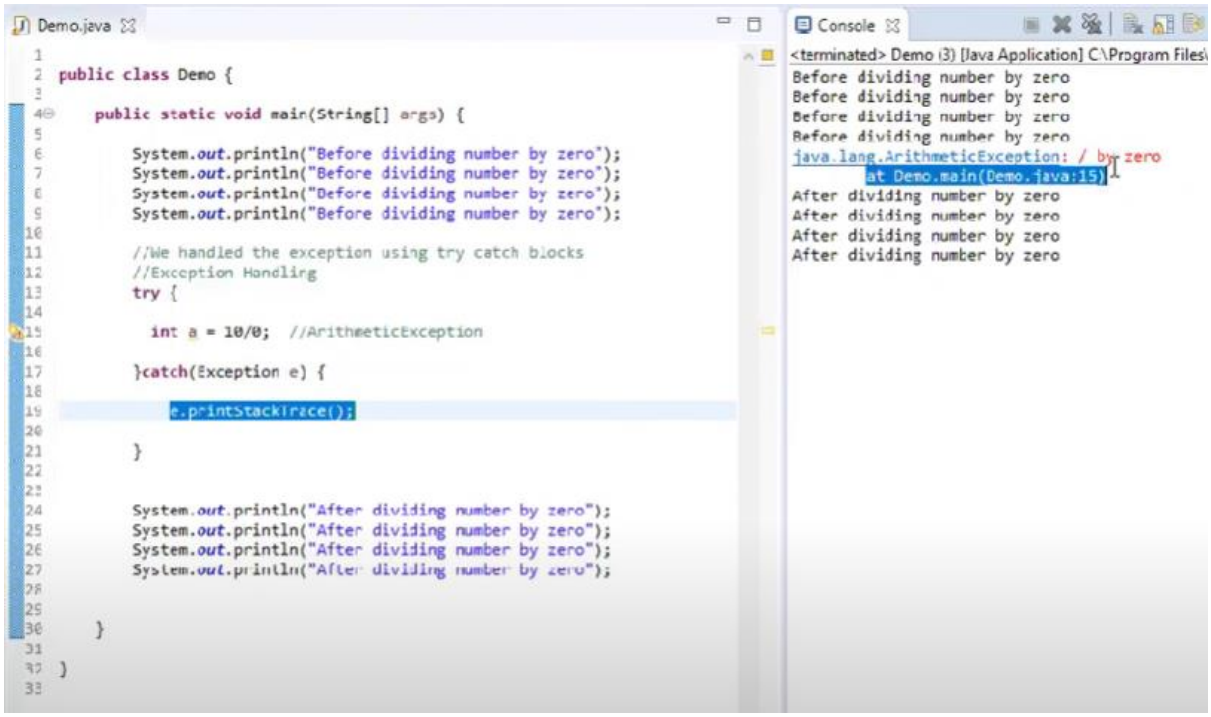
Statement2 has Exception

```
try{
    Statement1;
    Statement2;
    Statement3;
} catch(Exception e){
    //Code to handle the exception
}
```

Demo.java

```
1
2  public class Demo {
3
4      public static void main(String[] args) {
5
6          System.out.println("Before dividing number by zero");
7          System.out.println("Before dividing number by zero");
8          System.out.println("Before dividing number by zero");
9          System.out.println("Before dividing number by zero");
10
11         try {
12
13             int a = 10/0;  //ArithmeticException
14
15         }catch(Exception e) {
16
17             System.out.println("Some Exception got handled here");
18
19         }
20
21
22         System.out.println("After dividing number by zero");
23         System.out.println("After dividing number by zero");
24         System.out.println("After dividing number by zero");
25         System.out.println("After dividing number by zero");
26
27
28     }
29
30 }
31
```


Demo.java
public class Demo {
public static void main(String[] args) {
System.out.println("Before dividing number by zero");
System.out.println("Before dividing number by zero");
System.out.println("Before dividing number by zero");
System.out.println("Before dividing number by zero");
//We handled the exception using try catch blocks
//Exception Handling
try {
int a = 10/0; //ArithmeticException
}catch(Exception e) {
System.out.println(e.getMessage());
}
System.out.println("After dividing number by zero");
System.out.println("After dividing number by zero");
System.out.println("After dividing number by zero");
System.out.println("After dividing number by zero");
}
}
Console
<terminated> Demo (3) [Java Application] C:\P
Before dividing number by zero
Before dividing number by zero
Before dividing number by zero
Before dividing number by zero
/ by zero
After dividing number by zero
After dividing number by zero
After dividing number by zero
After dividing number by zero


Demo.java
public class Demo {
public static void main(String[] args) {
System.out.println("Before dividing number by zero");
System.out.println("Before dividing number by zero");
System.out.println("Before dividing number by zero");
System.out.println("Before dividing number by zero");
//We handled the exception using try catch blocks
//Exception Handling
try {
int a = 10/0; //ArithmeticException
}catch(Exception e) {
e.printStackTrace();
}
System.out.println("After dividing number by zero");
System.out.println("After dividing number by zero");
System.out.println("After dividing number by zero");
System.out.println("After dividing number by zero");
}
}
Console
<terminated> Demo (3) [Java Application] C:\Program Files
Before dividing number by zero
Before dividing number by zero
Before dividing number by zero
Before dividing number by zero
java.lang.ArithmeticException: / by zero
at Demo.main(Demo.java:15)
After dividing number by zero
After dividing number by zero
After dividing number by zero
After dividing number by zero

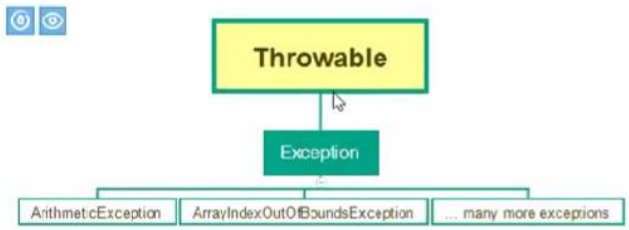
Demo.java

```
public class Demo {

    public static void main(String[] args) {

        System.out.println("Before dividing number by zero");
        System.out.println("Before dividing number by zero");
        System.out.println("Before dividing number by zero");
        System.out.println("Before dividing number by zero");

        sample();


        System.out.println("After dividing number by zero");
        System.out.println("After dividing number by zero");
        System.out.println("After dividing number by zero");
        System.out.println("After dividing number by zero");


    }

    public static void sample() {

        //We handled the exception using try catch blocks
                //Exception Handling
                try {

                  int a = 10/0;  //ArithmeticException

                }catch(Exception e) {

                    e.printStackTrace();

                }

    }

}
```

Console

```
<terminated> Demo (3) [Java Application] C:\Program Files\Java\jck-
Before dividing number by zero
Before dividing number by zero
Before dividing number by zero
Before dividing number by zero
java.lang.ArithmeticException: / by zero
        at Demo.sample(Demo.java:28)
        at Demo.main(Demo.java:11)
After dividing number by zero
After dividing number by zero
After dividing number by zero
After dividing number by zero
```

Exceptions Hierarchy



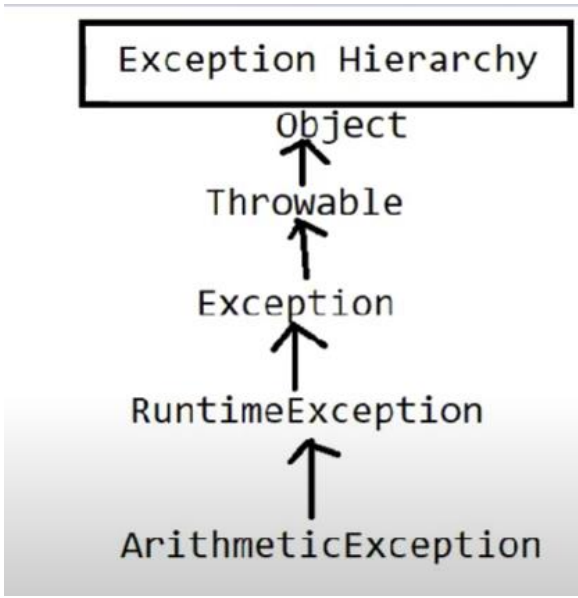
- Demonstrate ArithmeticException and handle it using 'ArithmeticException' class in catch block - Demonstrate here
- Demonstrate ArrayIndexOutOfBourdsException and handle it using 'ArrayIndexOutOfBoundsException' class in catch block - Demonstrate here
- Exception class is the parent class of all the Exception Classes like ArithmeticException and ArrayIndexOutOfBoundsException classes and can handle them
- Throwable class is the grant parent class of all the Exception Classes like ArithmeticException and ArrayIndexOutOfBoundsException classes and can handle them

Demo.java

```
public class Demo {

    public static void main(String[] args) {

        //ArrayIndexOutOfBounds Exception

        int[] a = new int[3];

        a[0]= 9;
        a[1] = 6;
        a[2] = 4;
        a[3] = 8;

    }

}
```

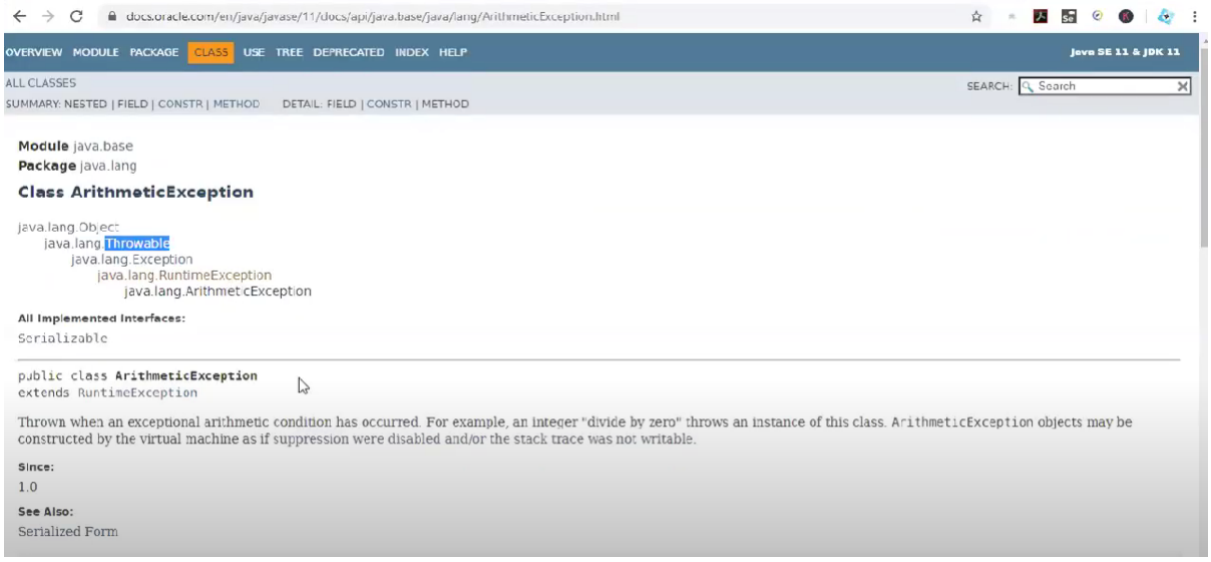
Console

```
<terminated> Demo (3) [Java Application] C:\Program Files\Java\jck-11.0.7\bin\javaw.exe (24-Jun-
Exception in thread "main" java.lang.ArrayIncexOutOfBoundsException: Ind
        at Demo.main(Demo.java:13)
```



In Java, what is the parent class of all the classes?

Object



Module java.base
Package java.lang

Class IndexOutOfBoundsException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.IndexOutOfBoundsException

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException

---

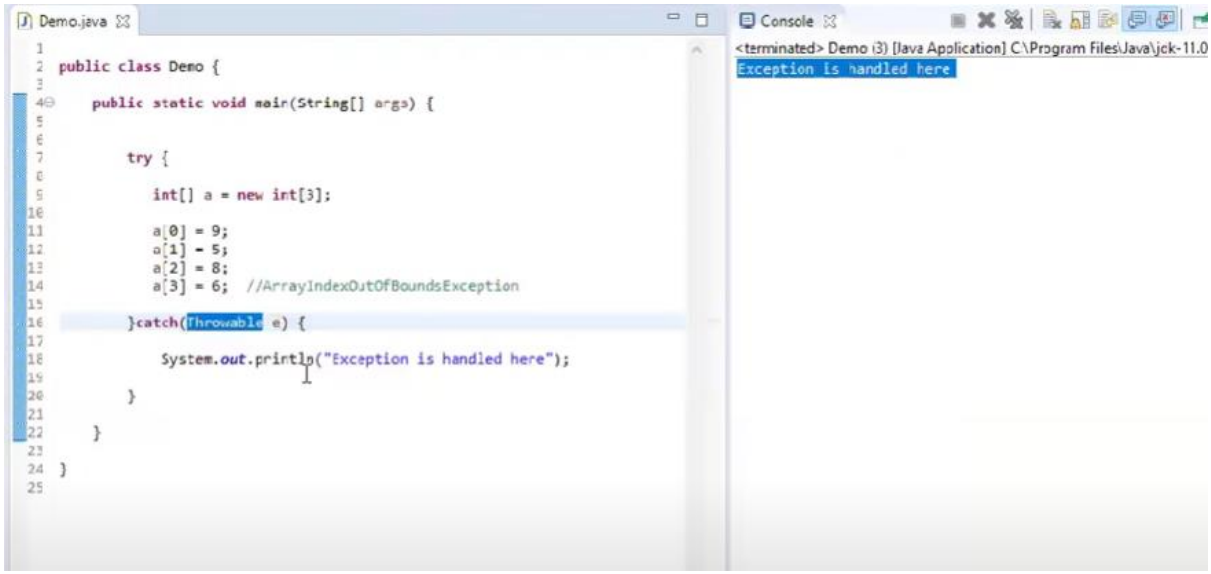
public class **IndexOutOfBoundsException**
extends RuntimeException

Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

Applications can subclass this class to indicate similar exceptions.

Since:

1.0



Demo.java

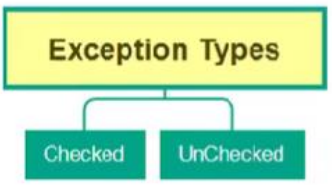
```
1
2  public class Demo {
3
4      public static void main(String[] args) {
5
6
7          try {
8
9             System.out.println("first line in try block");
10            int a = 10/0;
11            System.out.println("second line in try block");
12
13         }catch(Exception e) {
14
15             System.out.println("exception is handled here");
16
17         }
18
19
20         System.out.println("After");
21     }
22
23 }
24
```

Console

```
<terminated> Demo (3) [Java Application] C
first line in try block
Exception is handled here
After
```

Exception Types

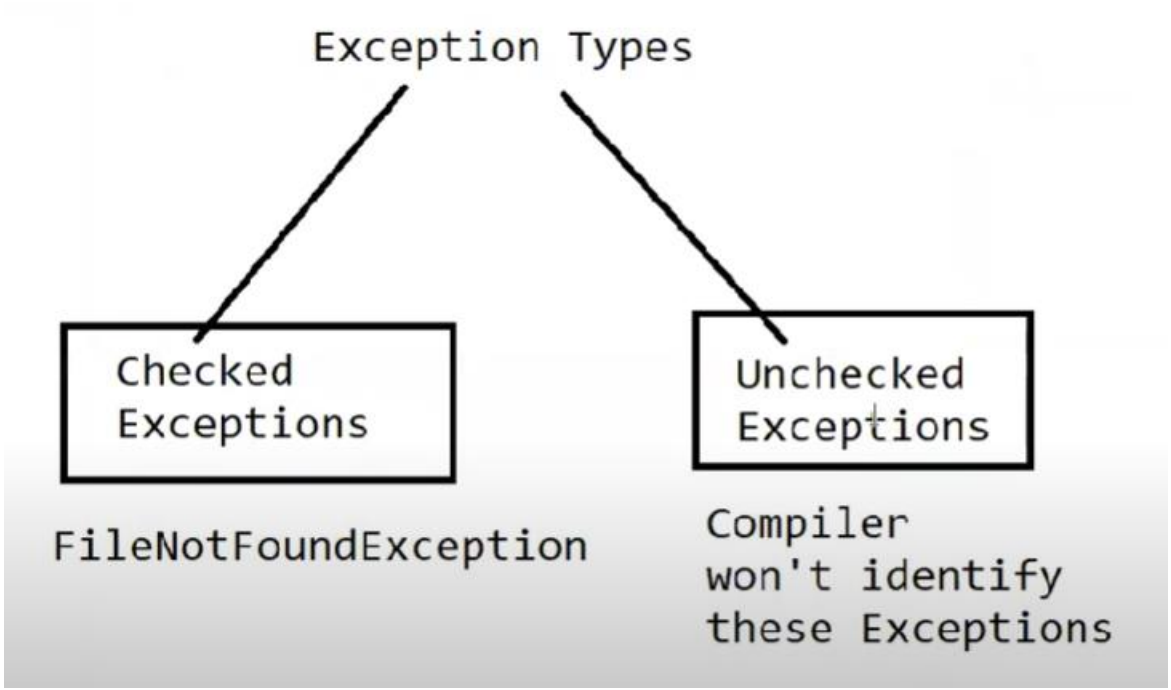
- Exceptions can be categorized as below



- Unchecked exceptions are the exceptions that are not checked by compiler and will occur only during execution - Demonstrate AirhmeticException
- Checked Exceptions are the exceptions that are checked by the compiler - Demonstrate FileNotFoundException
- Handling Checked Exceptions using try .. catch block
- Ignoring Checked Exceptions using throws keyword

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;

public class Demo {

    public static void main(String[] args) {

        //Representing a file
        File file = new File("D:\\ABC\\xyz.txt");

        //Read this file
        try {

            FileReader fr = new FileReader(file);

        } catch (FileNotFoundException e) {

            // TODO Auto-generated catch block
            e.printStackTrace();

        }



    }

}
```

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;

public class Demo {

    public static void main(String[] args) throws FileNotFoundException {

        //Representing a file
        File file = new File("D:\\ABC\\xyz.txt");

        //Compiler is insiting you to handle the exception
            FileReader fr = new FileReader(file);


        System.out.println("End of the program");



    }

}
```



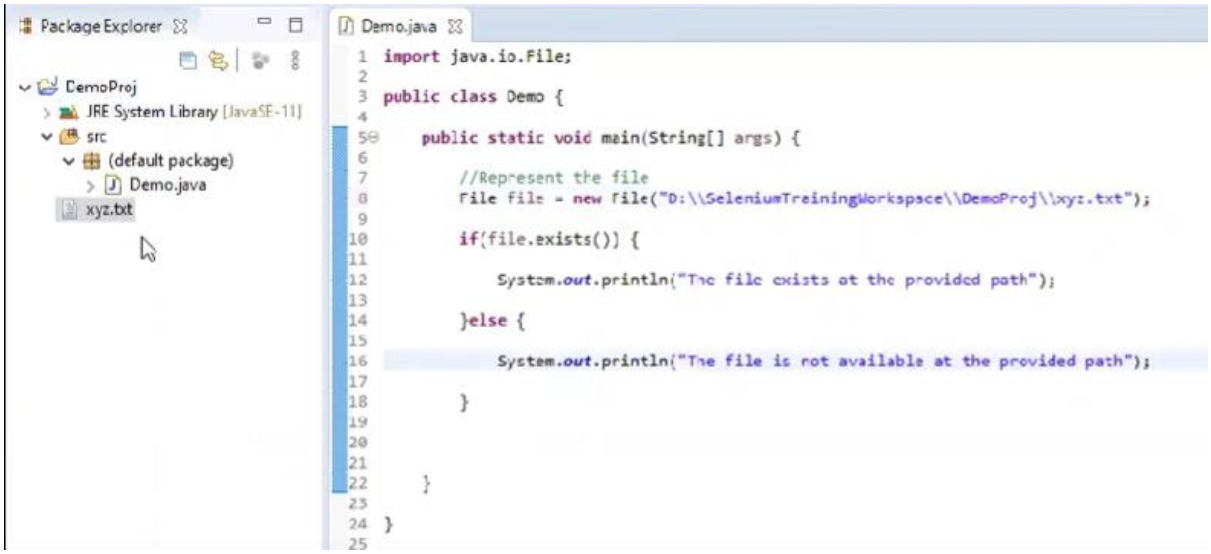
Java (Part 9) - Handling Files and Collections Framework

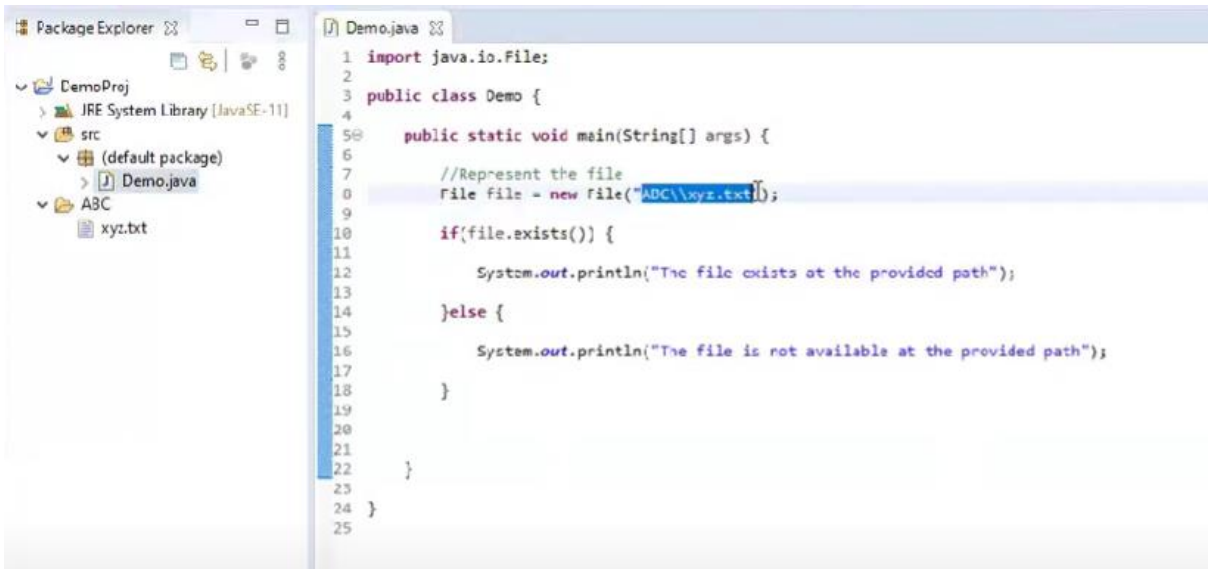
Handling Files

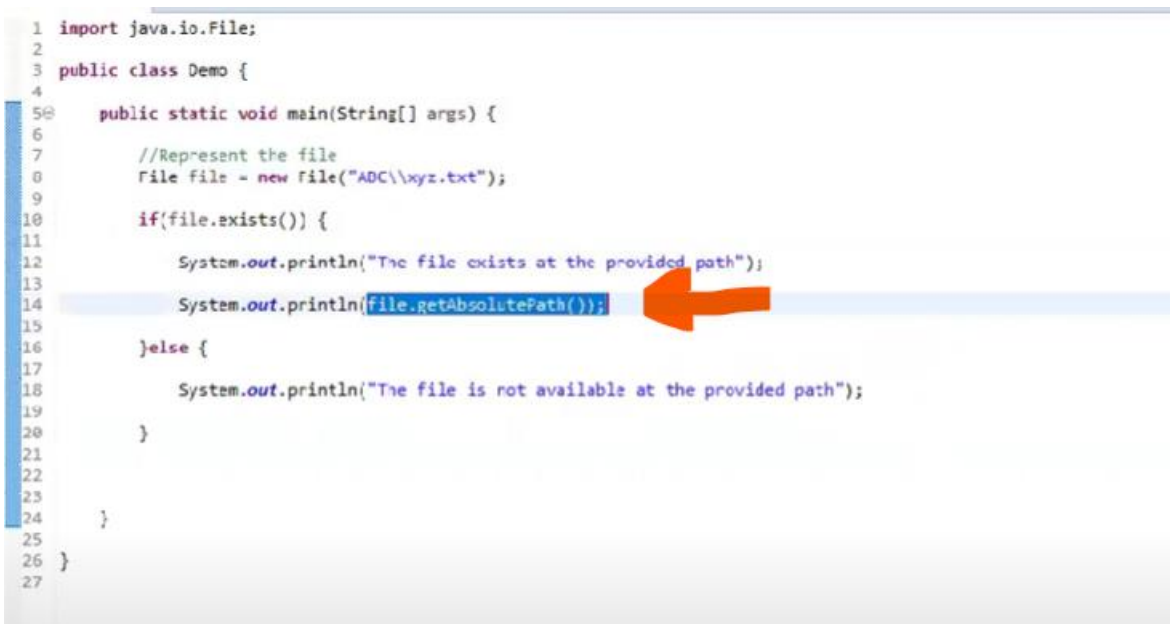
The purpose of handling files in Selenium is to read the text from the Files. (Demonstrate here)

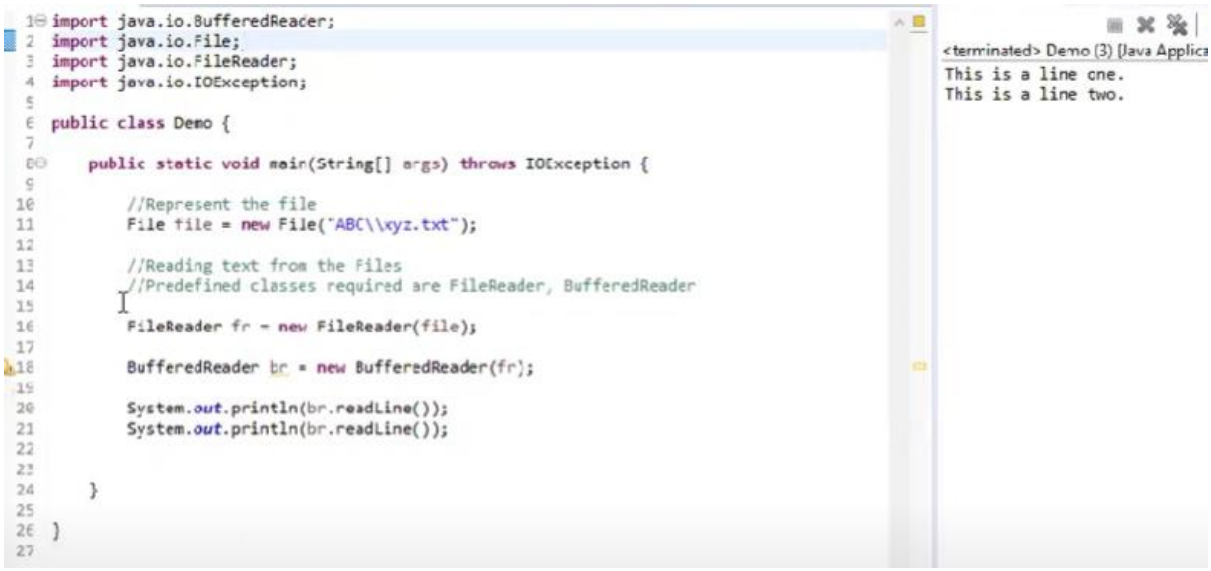
- File is a predefined Class in Java
- Using File Class represent a file in Java, which is available outside the Project workspace.
- Using File Class represent a file in Java, which is available inside the Project workspace.
  - Absolute Path
  - Shortcut Path
  - Finding the absolute path
- Read a File in Java and print every line in the file to the output console
  - FileReader
  - BufferedReader
    - readLine()
- Optimize the reading and printing from a File using while loop

```java
import java.io.File;

public class Demo {

    public static void main(String[] args) {

        //Represent the file
        File file = new File("D:\\ADC\\xyz.txt");

        if(file.exists()) {

            System.out.println("The file exists at the provided path");

        }else {

            System.out.println("The file is not available at the provided path");

        }


    }

}
```


Package Explorer
DemoProj
JRE System Library [JavaSE-11]
src
(default package)
Demo.java
xyz.txt
Demo.java
1 import java.io.File;
2
3 public class Demo {
4
5 public static void main(String[] args) {
6
7 //Represent the file
8 File file = new File("D:\\SeleniumTreiningWorkspace\\DemoProj\\xyz.txt");
9
10 if(file.exists()) {
11
12 System.out.println("The file exists at the provided path");
13
14 }else {
15
16 System.out.println("The file is not available at the provided path");
17
18 }
19
20
21
22 }
23
24 }
25

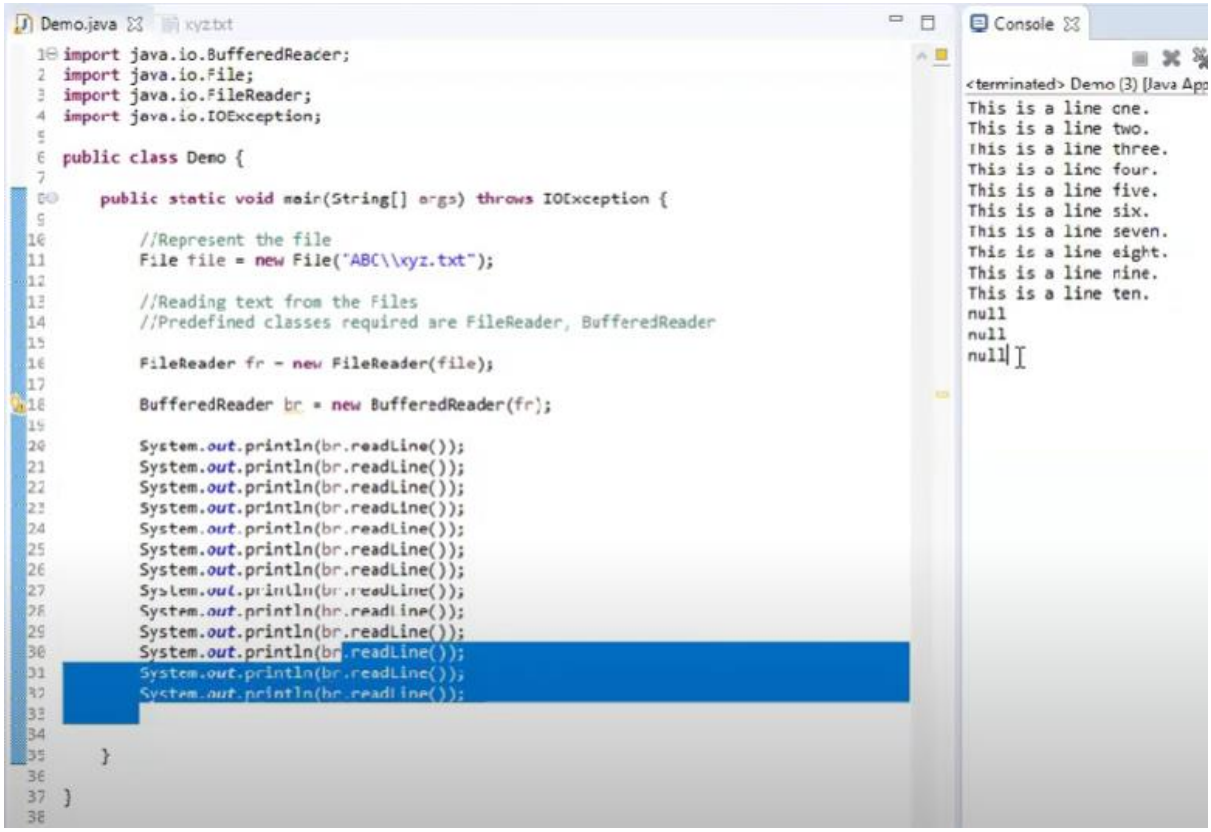

Package Explorer
DemoProj
JRE System Library [JavaSE-11]
src
(default package)
Demo.java
ABC
xyz.txt
Demo.java
1 import java.io.File;
2
3 public class Demo {
4
5 public static void main(String[] args) {
6
7 //Represent the file
8 File file = new File("ADC\\xyz.txt");
9
10 if(file.exists()) {
11
12 System.out.println("The file exists at the provided path");
13
14 }else {
15
16 System.out.println("The file is not available at the provided path");
17
18 }
19
20
21
22 }
23
24 }
25


1 import java.io.File;
2
3 public class Demo {
4
5 public static void main(String[] args) {
6
7 //Represent the file
8 File file = new File("ADC\\xyz.txt");
9
10 if(file.exists()) {
11
12 System.out.println("The file exists at the provided path");
13
14 System.out.println(file.getAbsolutePath());
15
16 }else {
17
18 System.out.println("The file is not available at the provided path");
19
20 }
21
22
23
24 }
25
26 }
27



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

public class Demo {

    public static void main(String[] args) throws IOException {

        //Represent the file
        File file = new File("ABC\\xyz.txt");

        //Reading text from the files
        //Predefined classes required are FileReader, BufferedReader

        FileReader fr = new FileReader(file);

        BufferedReader br = new BufferedReader(fr);

        System.out.println(br.readLine());
        System.out.println(br.readLine());


    }

}
```

```
<terminated> Demo (3) [Java Applica
This is a line one.
This is a line two.
```



Demo.java xyz.txt

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

public class Demo {

    public static void main(String[] args) throws IOException {

        //Represent the file
        File file = new File("ABC\\xyz.txt");

        //Reading text from the files
        //Predefined classes required are FileReader, BufferedReader

        FileReader fr = new FileReader(file);

        BufferedReader br = new BufferedReader(fr);

        System.out.println(br.readLine());
        System.out.println(br.readLine());
        System.out.println(br.readLine());
        System.out.println(br.readLine());
        System.out.println(br.readLine());
        System.out.println(br.readLine());
        System.out.println(br.readLine());
        System.out.println(br.readLine());
        System.out.println(br.readLine());
        System.out.println(br.readLine());
        System.out.println(br.readLine());
        System.out.println(br.readLine());


    }

}
```

Console

```
<terminated> Demo (3) [Java App
This is a line one.
This is a line two.
This is a line three.
This is a line four.
This is a line five.
This is a line six.
This is a line seven.
This is a line eight.
This is a line nine.
This is a line ten.
null
null
null
```

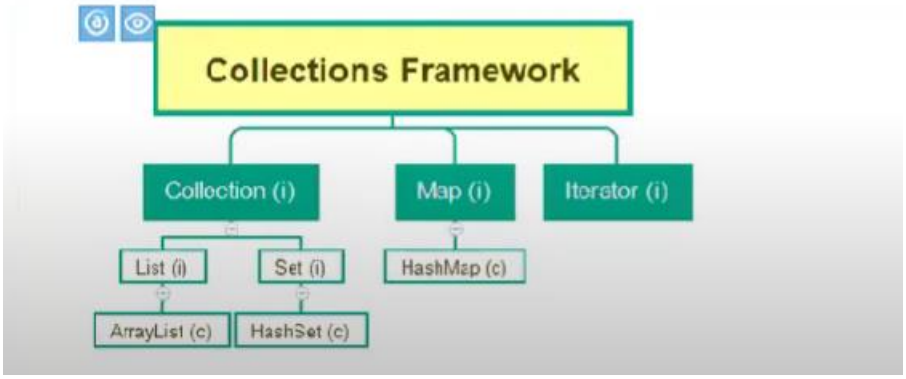
```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class Demo {

    public static void main(String[] args) throws IOException {

        //Represent the file
        File file = new File("ABC\\xyz.txt");

        //Reading text from the files
        //Predefined classes required are FileReader, BufferedReader

        FileReader fr = new FileReader(file);

        BufferedReader br = new BufferedReader(fr);

        String str;

        while((str=br.readLine())!=null) {

            System.out.println(str);

        }

        br.close();


    }

}
```

```
<terminated> Demo (3) [Java Application] C
This is a line one.
This is a line two.
This is a line three.
This is a line four.
This is a line five.
This is a line six.
This is a line seven.
This is a line eight.
This is a line nine.
This is a line ten.
```

Collections framework

Collection is a group of individual **Objects**.

- Array's are fixed in sized, where as Collections are grow-able in size
- Though Collections Framework is a vast subject, we have to only learn the below for Selenium:



Array	Collection
Group of values	Group of Objects
Fixed in size	Not Fixed in size

- **ArrayList**
  - ArrayList is nothing but a re-sizable array and is not of fixed size
  - Demonstrate an ArrayList which stores integer values and uses for loop to print those values - Demonstrate here
  - Demonstrate an ArrayList which stores different types of values and uses for each loop to print those values - Demonstrate here
  - Assigning the object of ArrayList class to Collection / List Interface - Demonstrate
- **HashSet**
  - Unlike ArrayList, HashSet wont have index values and hence we cannot use for loop with HashSet
  - Unlike ArrayList, HashSet stores the values in a random orcer
  - Demonstrate HashSet which stores integer type of values and uses for each loop to print those values - Demonstrate here
  - Assigning the object of HashSet class to Collection / Set Interface - Demonstrate

- **Iterator interface and iterator() method**
  - iterator() is a predefined method of Collection interface, who's return type is Iterator interface
  - hasNext() and next() are the predefined methods of the Iterator interface
  - Demonstrate using Iterator and iterator() with ArrayList - Demonstrate here
  - Demonstrate using Iterator and iterator() with HashSet - Demonstrate here
- **HashMap**
  - Instead of storing the objects as a group of Objects, HashMap stores the objects in the form of key value pairs.
  - Demonstrate a HashMap which stores different key value pairs and uses get() method to retrieve a value based on the provided key - Demonstrate here
  - Demonstrate a HashMap which stores different key value pairs and uses for each loop to print those values - Demonstrate here

length

Arrays

length()

String text

size()

ArrayList

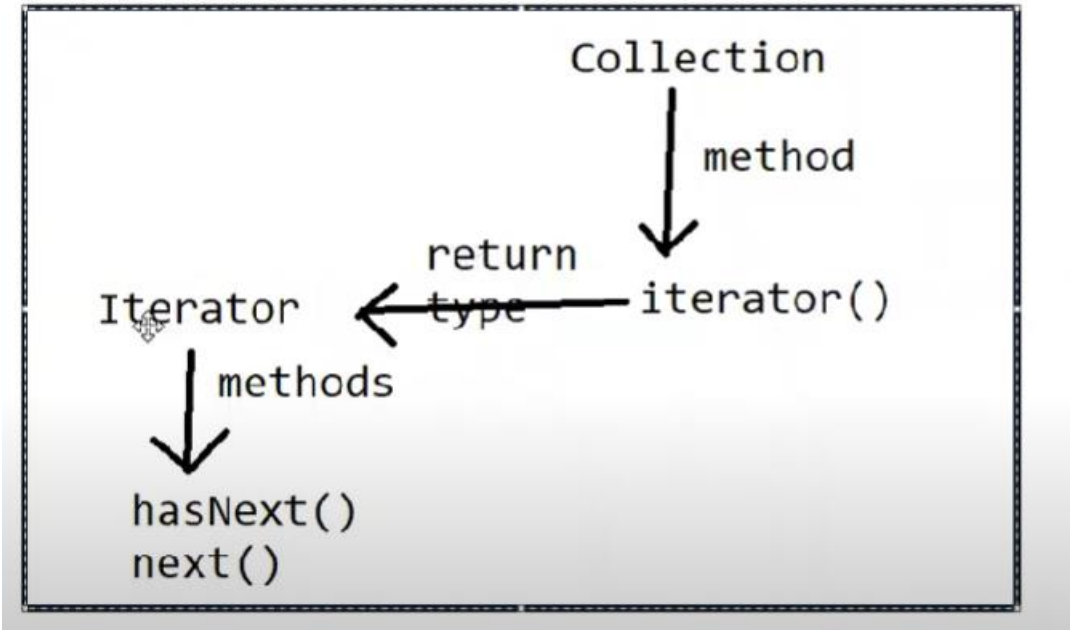
```java
import java.util.ArrayList;

public class Demo {

    public static void main(String[] args)  {

        ArrayList<Integer> alist = new ArrayList<Integer>();

        alist.add(5);   // index 0
        alist.add(9);   // index 1
        alist.add(7);   // index 2
        alist.add(3);   // index 3

        System.out.println("The size of the ArrayList is "+alist.size());

        for(int i=0;i<alist.size();i++) {   //4   //4

            System.out.println(alist.get(i));  //5  9  7  3

        }

    }

}
```

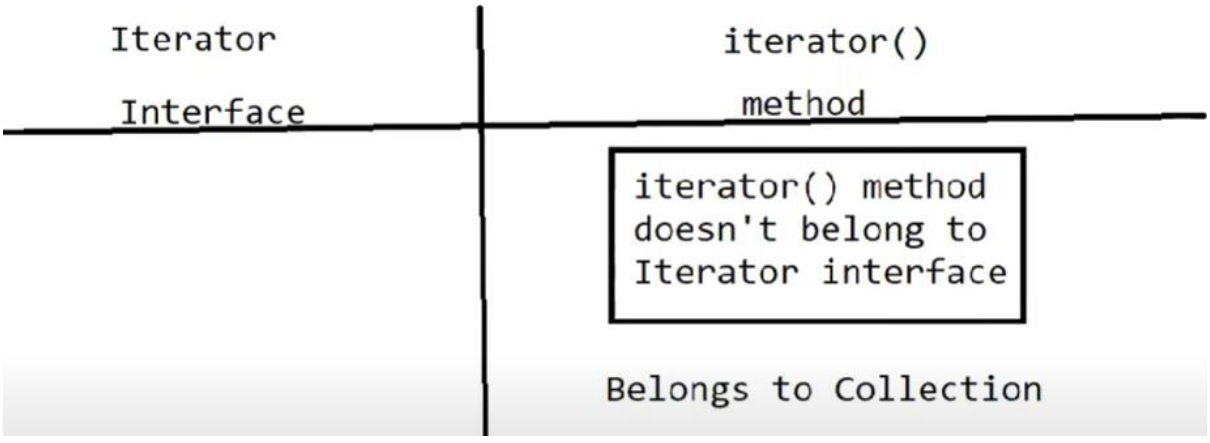
```java
import java.util.ArrayList;
import java.util.List;

public class Demo {

    public static void main(String[] args)  {

        List<Integer> list = new ArrayList<Integer>();

        list.add(5);   // index 0
        list.add(9);   // index 1
        list.add(7);   // index 2
        list.add(3);   // index 3

        for(Integer i :  list) {

            System.out.println(i);

        }

    }

}
```

```java
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

public class Demo {

    public static void main(String[] args)  {

        Collection<Integer> collection = new ArrayList<Integer>();

        collection.add(5);   // index 0
        collection.add(9);   // index 1
        collection.add(7);   // index 2
        collection.add(3);   // index 3

        for(Integer i :  collection) {

            System.out.println(i);

        }

    }

}
```

ArrayList	HashSet
Printed in Order Stored in a order index for loop for-each loop	Objects will be stored in random order No index Should not use for loop for-each loop

```java
import java.util.HashSet;
import java.util.Set;

public class Demo {

    public static void main(String[] args)  {

        Set<String> set = new HashSet<String>();

        set.add("My");
        set.add("Name");
        set.add("Is");
        set.add("Arun");


        for(String i :  set) {

            System.out.println(i);

        }

    }

}
```





```java
import java.util.ArrayList;
import java.util.Iterator;

public class Demo {

    public static void main(String[] args)  {

        ArrayList<Integer> alist = new ArrayList<Integer>();

        alist.add(5);
        alist.add(9);
        alist.add(8);
        alist.add(6);

        //for loop
        //for-each loop
        //Iterator and iterator()

        Iterator<Integer> itr = alist.iterator();

        while(itr.hasNext()) {

            System.out.println(itr.next());

        }


    }

}
```

ArrayList	HashSet	HashMap
add()	add()	put()
get(index)	✗	get(Key)

```java
import java.util.HashMap;

public class Demo {

    public static void main(String[] args)  {

        HashMap<Integer,String> hmap = new HashMap<Integer,String>();

        hmap.put(101,"Arun");
        hmap.put(102,"Tharun");
        hmap.put(103,"Varun");
        hmap.put(104,"Kiran");

        for(Integer i : hmap.keySet()) {

            System.out.println(hmap.get(i));  // Arun Tharun Varun Kiran


        }

    }

}
```