Day-1

Java Programming

Java is Object Oriented programming Language.

1) Class

2) Object

3) Inheritance

4) Polymorphism

5) Abstraction

6) Encapsulation

Types of programming languages

1) Structured: C, Python

2) Object based: Visual Basic, VB Script, Python

3) Object oriented Java, C++, C#, Python

Java -1994 Sun Microsystems

JDK- Java Development Kit

JRE - Java Run Time Environment

JVM - Java Virtual Machine (because of JVM java is system independent. Basically, JVM convert java file into class file which contains Byte code so any system can read it.)

**program(.java) ----> .class (byte code)**

**windows**

**Linux**

**Mac**

**\*\* Java is platform independent language.**

**\*\* Java is case sensitive language.**

**Naming Conventions when you create a class**

**1) Class name should start with upper case letter**

**2) Should not start with numbers**

**3) should not use special characters ~! %&\* etc. (is accepted in the middle of class name)**

**4) Do not give spaces**

**Day-2**

**Java Variables and Data Types**

**Java Operators**

**Variable? Variable is a container which can hold data.**

**int x=10; (Statically typed language)**

**String name="John"**

**float price=150.50**

**double price=150.50**

**char grade='A'**

**boolean status=true**

**Data types**

**1) Primitive Data Types**

**2) Non-Primitive /Derived Data types**

**Primitive Data Types**

1. **short/int/long -----> Integer type data**
2. **float/double --------> Decimal type of data**
3. **char ------------> single character (single quotes)**
4. **boolean -----------> boolean value (true/false)**

**Wrapper classes**

**Integer , Double, Character, Boolean**

// PREMITIVE DATA TYPES

// boolean m; declaring variable

// m = false; initialization

// Warper class Integer, Double, Character, Boolean

**boolean** pass = **true**; // 1 byte Sores true or false value

**byte** num1 = 127; // 1 byte Stores whole numbers from -128 to 127

**short** num2 = 32767; // 2 bytes Stores whole numbers from -32,768 to 32,767

**int** num3 = 22000; // 4 bytes Stores whole numbers from -2,147,483,648 to 2,147,483,647

**long** num4 = 910000; // 8 bytes Stores whole numbers from -+9,223,372,036,854,775,808

**char** grade = 'A'; // 2 bytes Stores a single character/letter or ASCII values\

**char** grade1 = 65; // same as grade = 'A'

**float** GPA = 3.12f; // 4 bytes Stores fractional numbers. for storing 6 to 7 decimal digits

**double** gpa = 3.12; // 8 bytes, Stores fractional numbers. Sufficient for storing 15 decimal digits

i = i++ is a postfix increment operator - it increments i, then returns it to its original value (because the i++ essentially "returns" the value of i before it was incremented.)

i = ++i would work since it's a prefix increment operator, and would return the value of I after the increment. However, you probably just want to do i++ there without any extra assignment as you do in the first run - it's (essentially) shorthand as it is for i = i+1.

// swapping in different way without using third variable

**int** j = 10;

**int** i = 20;

System.***out***.println("Approach 1");

System.***out***.println("Before Swapping j and i are " + j +" " + i);

j = j + i;

i = j - i;

j = j - i;

System.***out***.println("After Swapping j and i are " + j +" " + i);

j = 10; i = 20;

System.***out***.println("Approach 2");

System.***out***.println("Before Swapping j and i are " + j +" " + i);

i = j + i - (j=i); // right to left

System.***out***.println("After Swapping j and i are " + j +" " + i);

j = 10; i = 20;

System.***out***.println("Approach 3");

System.***out***.println("Before Swapping j and i are " + j +" " + i);

i= i^j;

j= i^j;

i= i^j; // using bitwise xor operation

System.***out***.println("After Swapping j and i are " + j +" " + i);

switch(variable)

{

case value: statements; break;

case value: statements; break;

case value: statements; break;

case value: statements; break;

case value: statements; break;

case value: statements; break;

default: statements

}

Number’s manipulation

**int** num = 3215870;

**int** rev = 0;

**int** count = 0 , sum = 0;

**int** oddDigits = 0 , evenDigits = 0 ;

System.***out***.println(num);

**while**(num != 0)

{

**if**((num % 10) % 2 == 0)

evenDigits++;

**else**

oddDigits++;

sum = sum + (num % 10);

rev = rev \* 10 + num%10;

num = num / 10;

count++;

}

System.***out***.println(rev);

System.***out***.println(count);

System.***out***.println("This is sum of digits" +sum);

System.***out***.println("even digits in num” + evenDigits);

System.***out***.println("odd digits in num "+oddDigits);

**Arrays**

**Array is collection elements of same data type (Homogenies data).**

**We can store multiple values into a single variable.**

**int** arr[] = **new** **int**[5]; This is fixed size Array

**int** arr2[] = {1,2,3,4,5};

**for**(**int** value:arr2) { //Enhanced for loop is for array or collection

System.***out***.println(value);}

2D Arrays: Two-dimensional array - rows & columns

**int** arr[][] = **new** **int** [3][2];

|  |  |
| --- | --- |
| 100 | 200 |
| 300 | 400 |
| 500 | 600 |

arr[0][0] = 100;

arr[0][1] = 200;

arr[1][0] = 300;

arr[1][1] = 400;

arr[2][0] = 500;

arr[2][1] = 600;

Find size of an 2D array

arr.length; // return number of rows

arr[0].length;// return number of columns

Traversing 2D Array Approach 1

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

**for**(**int** column=0; column<arr[0].length;column++) {

System.***out***.print(arr[row][column]);

}

System.***out***.println();

}

Traversing 2D Array Approach 2

**for**(**int** oneDemension[]: arr) {

**for**(**int** value:oneDemension) {

System.***out***.print(value);

}

System.***out***.println();}

Arrays cons:

Fixed Size, Same type values only (heterogeneous data)

Array of objects

System.***out***.println();

Object b[] = {10,5.6,'A',"Mohammad", **false**};

**for**(Object value: b) {

System.***out***.println(value);

}

System.***out***.println(b.length);

Sort Elements in Array

Ref Links:

https://www.youtube.com/watch?v=cJ2eMUiCFy4

<https://www.youtube.com/watch?v=3PLtvTUOCpM>

**int** a[] = {50,8,7,55,14,-8,20,40,10,1000};

**int** b[] = {50,8,7,55,14,-8,20,40,10,1000};

Integer c[] = {50,8,7,55,14,-8,20,40,10,1000};

// Using bubble sort

**int** sortedElement = 0;

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

**for**(**int** j = a.length -1 ; j>=sortedElement; j--){

**if**(a[i] > a[j])

a[j] = a[i] + a[j] - (a[i] = a[j]);

}

sortedElement++;

}

**for**(**int** value:a) {

System.***out***.print(value +" ");

}

System.***out***.println("\nB before sort: "+ Arrays.*toString*(b));

Arrays.*parallelSort*(b);

System.***out***.println("B after sort: "+Arrays.*toString*(b));

System.***out***.println("c before sort: "+Arrays.*toString*(c));

Arrays.*sort*(c);

System.***out***.println("c after sort: "+Arrays.*toString*(c));

Arrays.*sort*(c,Collections.*reverseOrder*());//c Must be object to use collection

System.***out***.println("c after sort: "+Arrays.*toString*(c));

**Strings**

**String is collection of characters.**

**String is a predefined class which has many built-in methods.**

String name="Mohammad”; // This is normal variable

String name1 = **new** String("Mohammad"); // This will create object

name.length();

name1.concat(name);

name1.trim(); //remove all leading and trilling spaces

name.equals("Mohammad");

name.equalsIgnoreCase("MoHaMmAd");

name.endsWith("ad");

"heiwh".contains("ei"); // true

"Wellcome to Java".replace("Java", "Selenium");

name.toUpperCase();

name.toLowerCase();

String s = "abc xyz:sdfdfs:sdafasf";

String[] r = s.split(" ");

Arrays.*binarySearch*(arr, "xyd")

DO NOT CONFIUSE length with length();

length: is keyword length of array(how many elements exists in the array)

length(): is a method returns number of characters in a string.

System.***out***.println( 1 + 1 + name1 + 1 + 2); output: 2 Mohammad 12

String s1 = "Mohammad";

String s2 = "Mohammad"; // it is value

System.***out***.println(s1 == s2); //true

System.***out***.println(s1.equals(s2));//true

String s3 = **new** String("Mohammad");

String s4 = **new** String("Mohammad"); // Objects

System.***out***.println(s3 == s4); // false here compares the objects/instances System.***out***.println(s3.equals(s4)); //true compares the value of the objects

Search string in a Array

String s = "abc:xyz:sdfdfs:sdafasf";

//search in Array

String []arr = s.split(":");

**boolean** found = **false**;

**for**(String value:arr) {

**if** (value.equals("abc")) {

found = **true**;

**break**;

}

**else**

System.***out***.println("Not found");

}

**if**(found){

System.***out***.println("found");

}

System.***out***.println(Arrays.*binarySearch*(arr, "xyd"));

# Revers String

String str = "Mohammad Ameri";

**int** sizeStr = str.length();

String revStr = "";

**for**(**int** i= sizeStr-1; i >= 0; i--) {

System.***out***.print(str.charAt(i));

revStr = revStr+str.charAt(i);

}

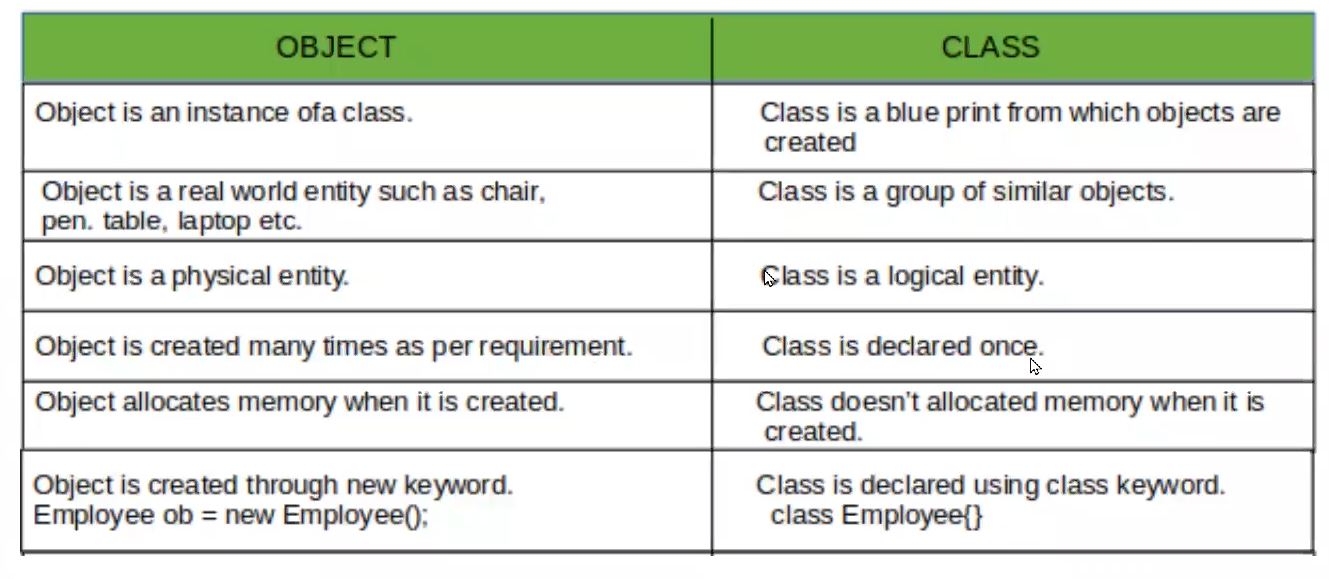
System.***out***.println("\n"+revStr);

Note: length() returns the length of the string, but last index is length– 1;

OOP Concepts

Class: is consists of two things Attributes and Methods.

Class is logical Entity (blue print). and it doesn’t have a physical existence, so it will not occupy any space in a MEMORY.

Object: is an instance of a class. It has a physical existence, so it needs space in memory. Objects are independent and we can Instantiate as many objects as we need.

Inside a method we cannot create another method.

Inside a class we cannot create another class.

Constructor method: same as the class name, no return, no void keyword, may take Arguments. Constructor invokes at the time of object creation.

Passing reference via method we passing an object;

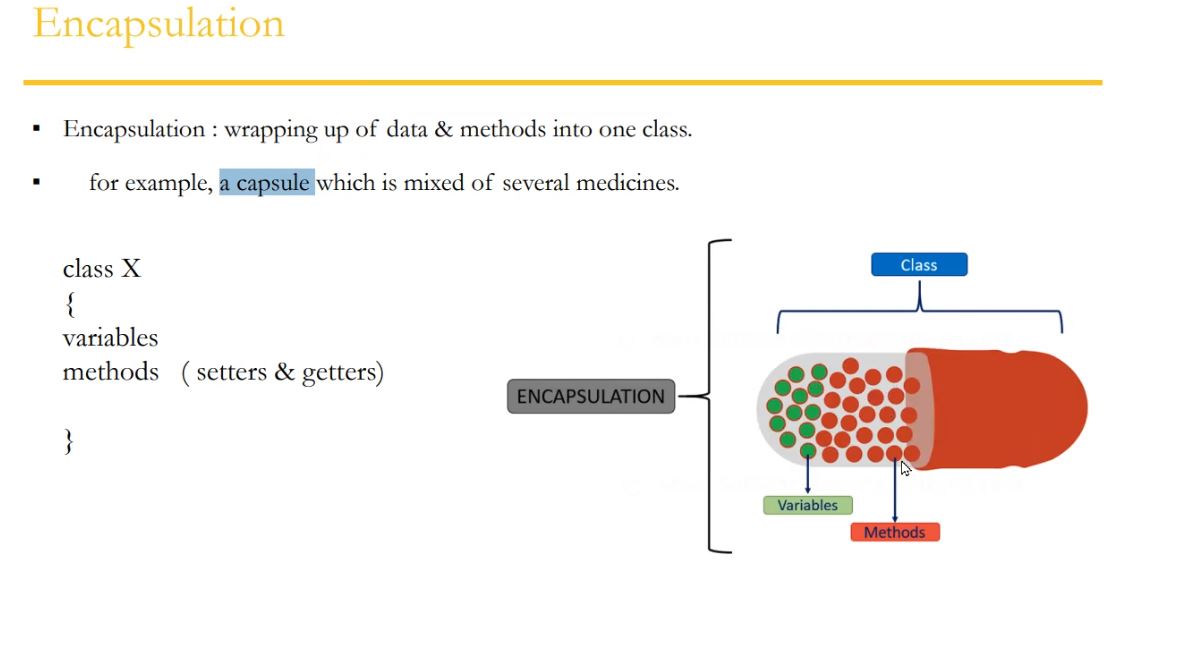
cal v1 = **new** cal();

v1.x = 17;

v1.add(v1);

System.***out***.println(v1.x);

**NOTE: Object name is a reference (pointer) to the location of the object in memory**

Encapsulation: Wrapping up of variables and methods into a single unite. Is required for security purposes. All attributes must be private, and only attributes are accessible via seters and getters.

**public** **class** Account {

**private** **int** acc\_no;

**private** String name;

**private** **double** amount;

**public** **int** getAcc\_no() {

**return** acc\_no;

}

**public** **void** setAcc\_no(**int** acc\_no) {

**this**.acc\_no = acc\_no;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **double** getAmount() {

**return** amount;

}

**public** **void** setAmount(**double** amount) {

**this**.amount = amount;

}

@Override

**public** String toString() {

**return**("Name is: " + **this**.getName() +" Account number is: " + **this**.getAcc\_no() + "Account Amount:"+ **this**.getAmount());

}

}

Polymorphism: One thing having many forms. Ploy(many).

Overloading is a technical term for Polymorphism.

Overloading is writhing multiple methods with the same name, but Parameters should be different.

Ex. Shape: circle, square, rectangle.

**public** **void** add(**int** c) {

x = c + 5;

}

**public** **void** add(**float** b, **int** c) {

x = c + 5;

}

**public** **void** add(**int** c , **float** b) {

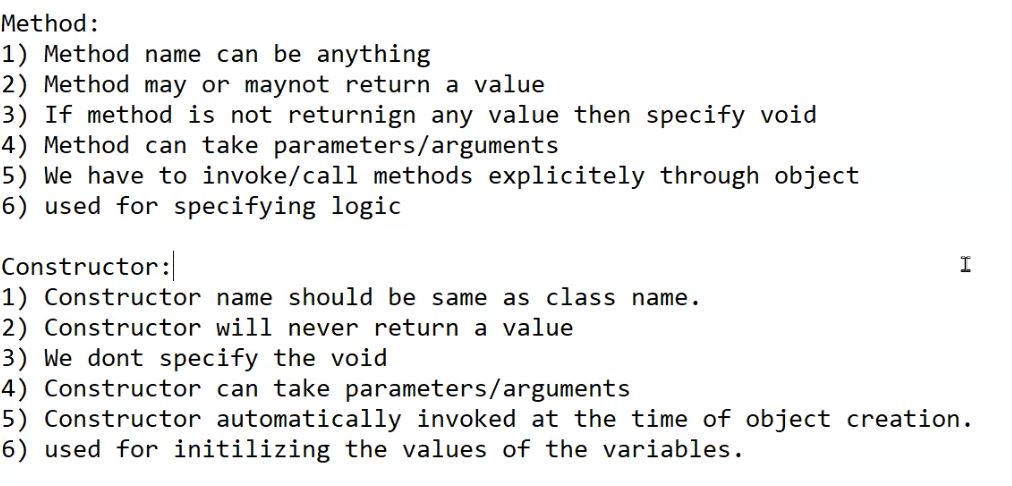
x = c + 5;

}

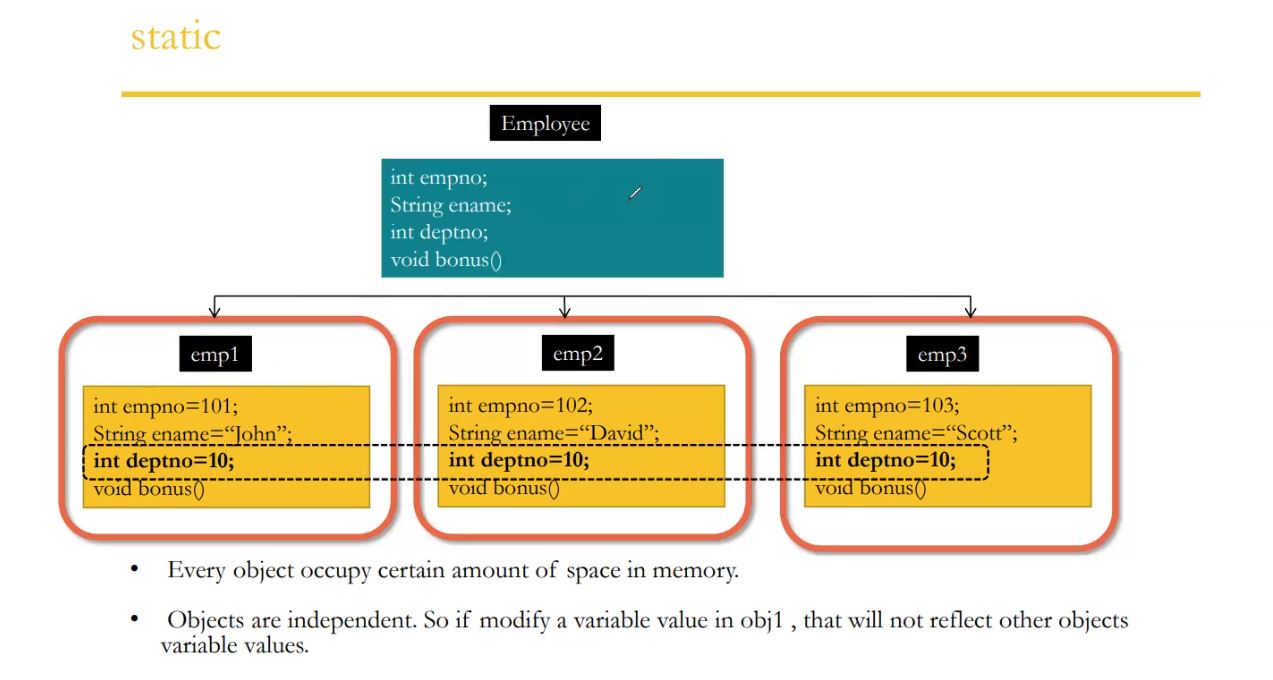
**public** **void** add(**int** c , **int** b) {

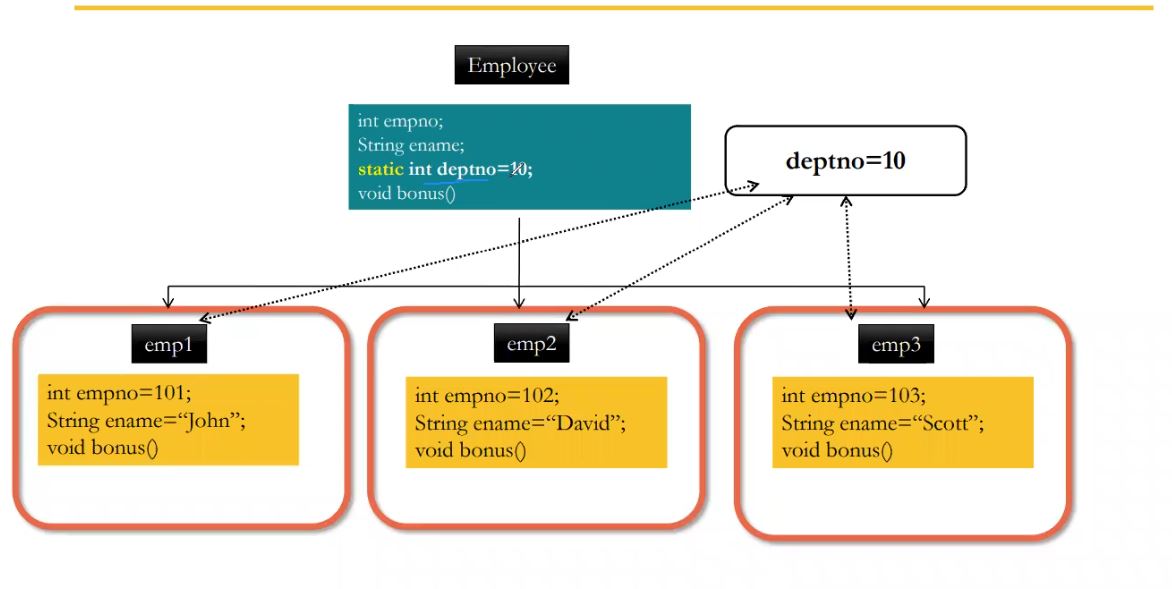
x = c + 5;

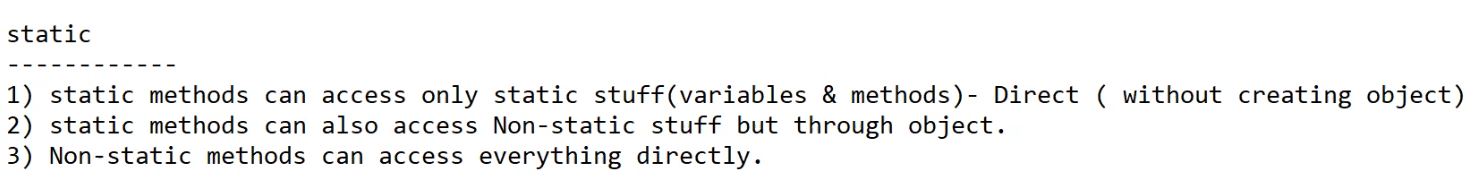
}

Arguments types and orders are considered, return type not.

Static Keyword







**Note: When we create static variable or method, we can access that directly by Classname.method() or Classname.variable (If the main function is in the different file use classname.method ) without creating any objects.**

There advantages of static are avoiding DUBLICATIOAN and REUSABILITY, also direct access.

**int** m;

**static** **int** *n*;

**public** **static** **void** n1() {

System.***out***.println("this is static method");

*n* = 200;

m = 400; // Cannot access Directly a non-static method

m1(); // Cannot access Directly a non-static variable

}

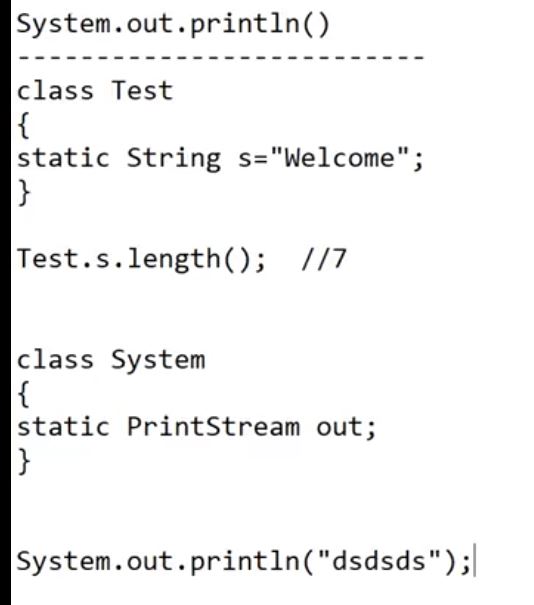
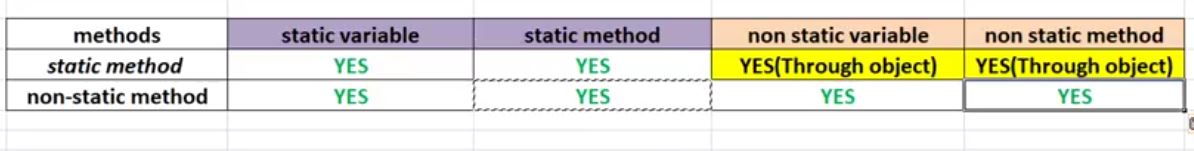
**public** **void** m1() {

System.***out***.println("this is non-static method");

*n* = 100;

m = 200;

*n1*();

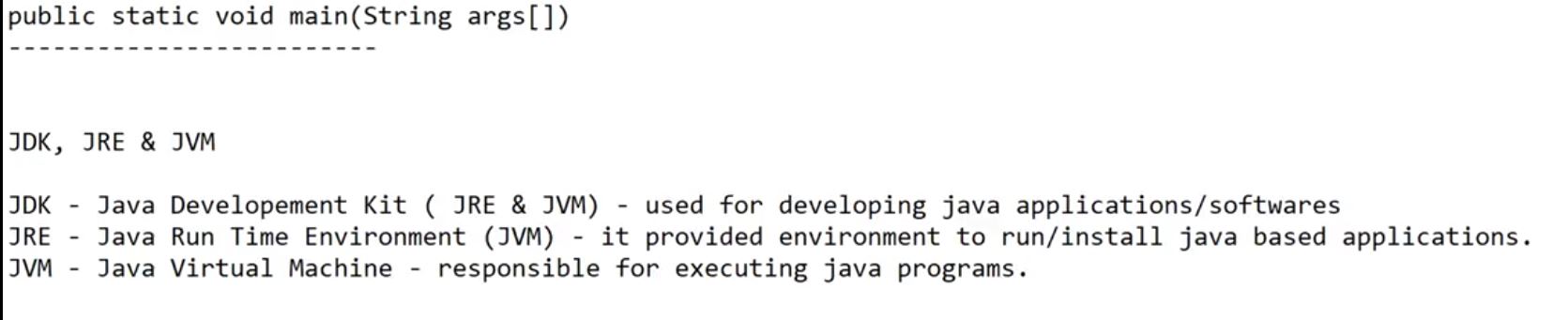


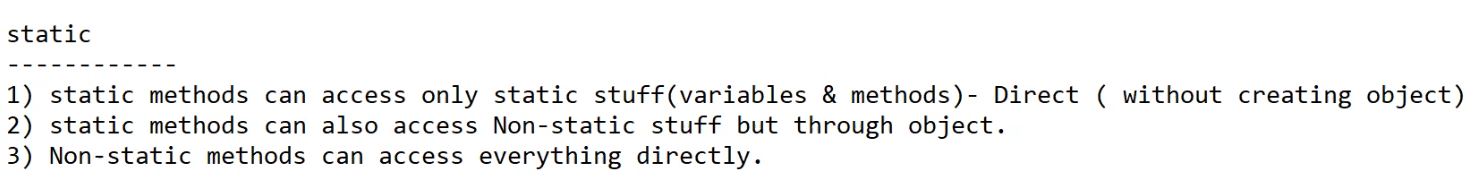
I the left picture we see how important static keyword is.

System is a class that has a variable out type of PrintStream which contains a method println();

It is exactly same as Test class which has a variable s type of String that contains length() method. We can use this method directly (without creating any object)

Basic fundamental concept: JVM runs the main function and it look for the specific pattern for the main function which is: public static void main(String[] args).





**Note: When we create static variable or method, we can access that directly by Classname.method() or Classname.variable (If the main function is in the different file use classname.method ) without creating any objects.**

There advantages of static are avoiding DUBLICATIOAN and REUSABILITY, also direct access.

**int** m;

**static** **int** *n*;

**public** **static** **void** n1() {

System.***out***.println("this is static method");

*n* = 200;

m = 400; // Cannot access Directly a non-static method

m1(); // Cannot access Directly a non-static variable

}

**public** **void** m1() {

System.***out***.println("this is non-static method");

*n* = 100;

m = 200;

*n1*();

}

**Inheritance**

Parent child relationship. Child class will inherit all method and variables from the parent class.

**Single Inheritance: child has a one parent.**

A

class B extends A{} B is the child and A is the parent.

B

**Multi-level Inheritance**

A

B

Class B extends A{}

Class C extends B{}

C

Class C will contain A and B methods And Variable

**Multiple Inheritance**

This type of inheritance is

Not allowed in java.

But we may use Interface

To achieve this pattern.

**Hierarchy Inheritance**

Method Overriding is used here.

**Hybrid Inheritance: consist of Hierarchy and multiple Inheritance.**

This pattern is only Achievable by Interface.

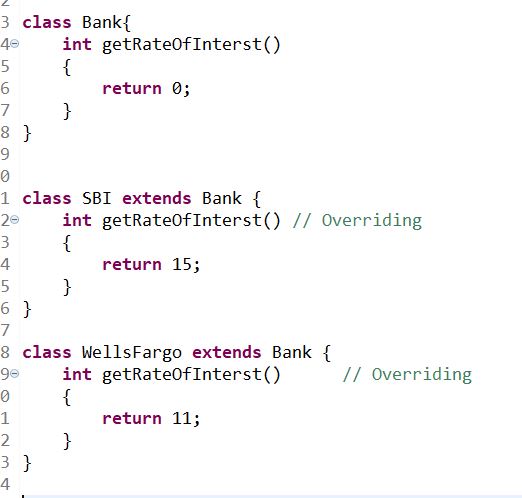
Method Overriding: we use method overriding to change a parent functio

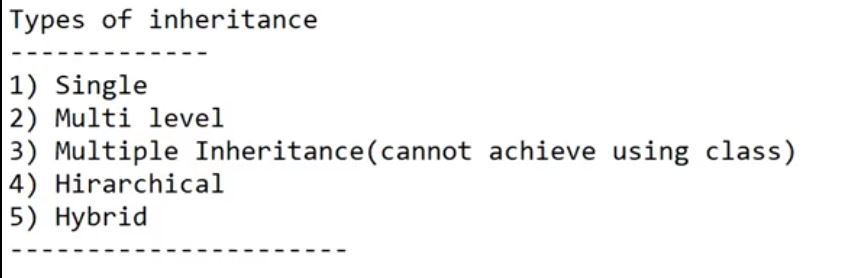
In method Overriding signature of the method should not change, only we can change the body.

Method Overloading Method overriding

No Inheritance required Inheritance required.

Method Signature must change Method Signature must be same





Final keyword

Interface

Packages

1- Final keyword can be used for variable and method & class.

Final keyword for variable makes it constant.

Final keyword for method: method can not be override in the child class.

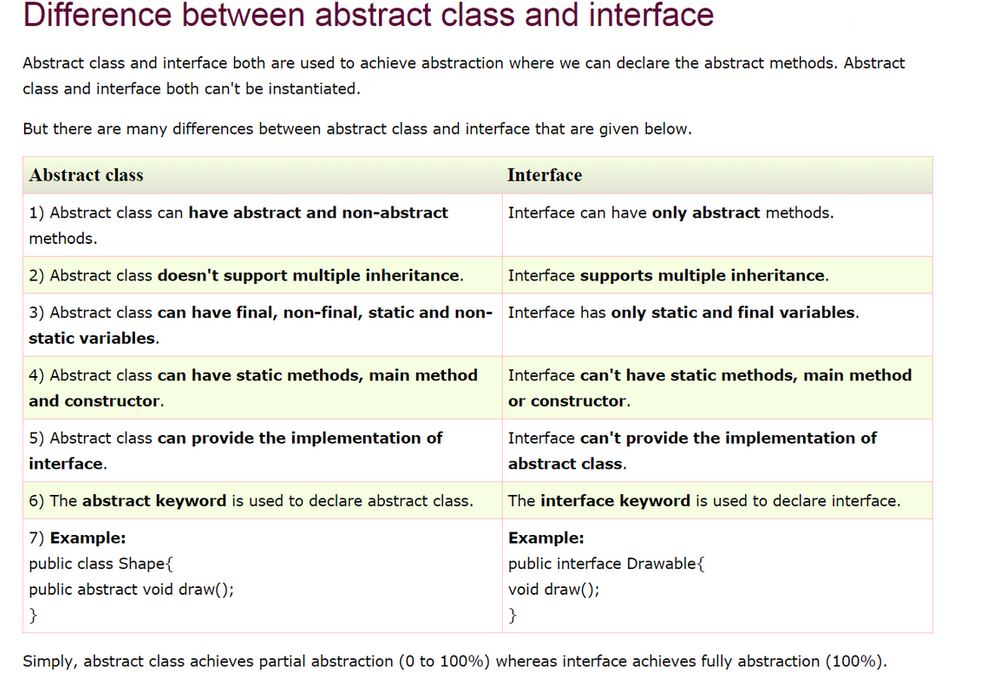
Final keyword for class: class cannot be extended.

2- Interface

* + It makes it possible to achieve some Inheritance types that it not possible alone with classes.
  + Structure of Interface is exactly as class.
  + Interface contains only static method and static variables. (By default, they are static). Also, Abstract methods.
  + In interface by default methods are public.
  + In interface methods are by default abstract
  + We cannot instantiate interface (make a object of Interface)

Abstract: A method have only the definition(signature) but not implementation (Body).

We use implements keyword to extends an interface

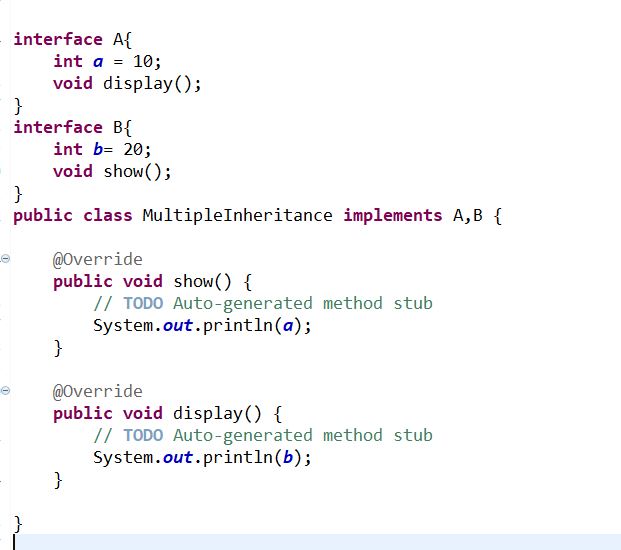


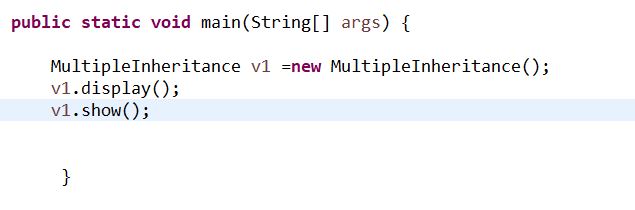
Multiple Inheritance using Interface

Class to class we use extends

Interface to class we use implements

Interface to Interface we use extends

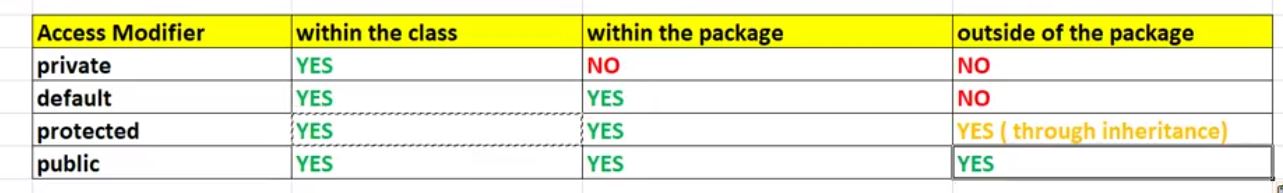
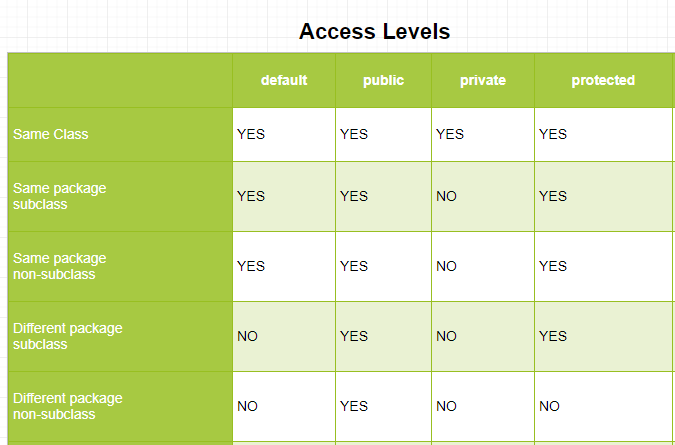




Note: A.display() and B.show() by default are public methods so when implementing in the child class we must change Access modifiers to public.

Access modifiers:

* Public: everywhere
* Default: no access modifier is set. Accessible only within the package. (Classes in pkg)
* Protected: accessible within the package and outside of the package only through inheritance
* Private: Only within the class



Exception handling

Exception is an event which will terminate program unexpectedly.

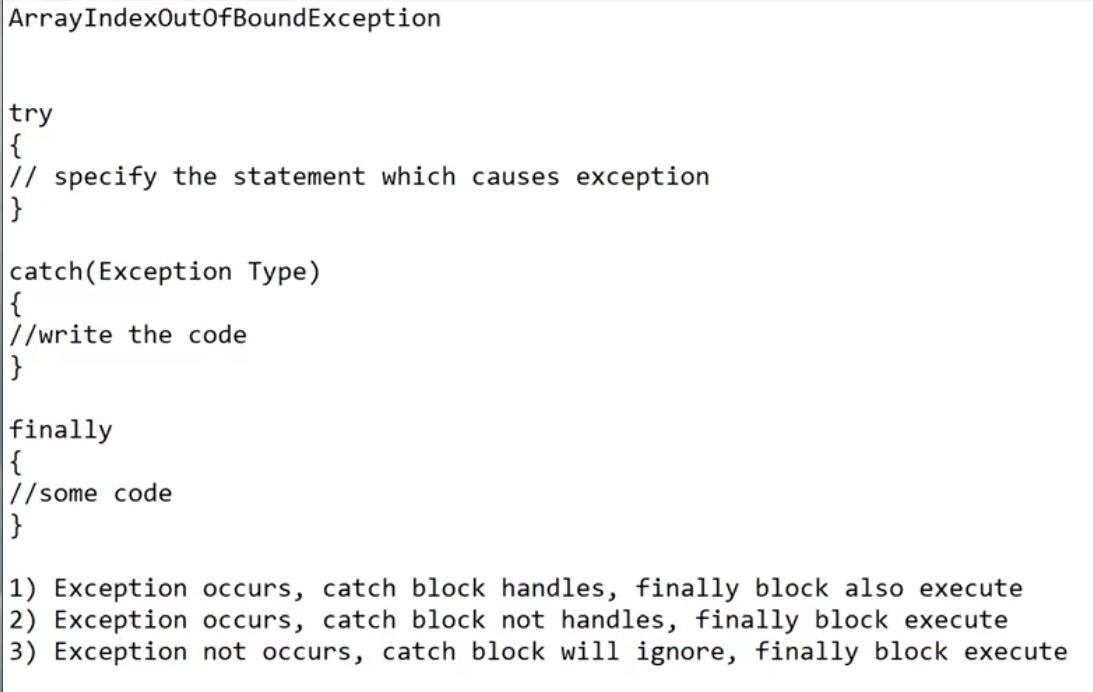
There are two type exceptions:

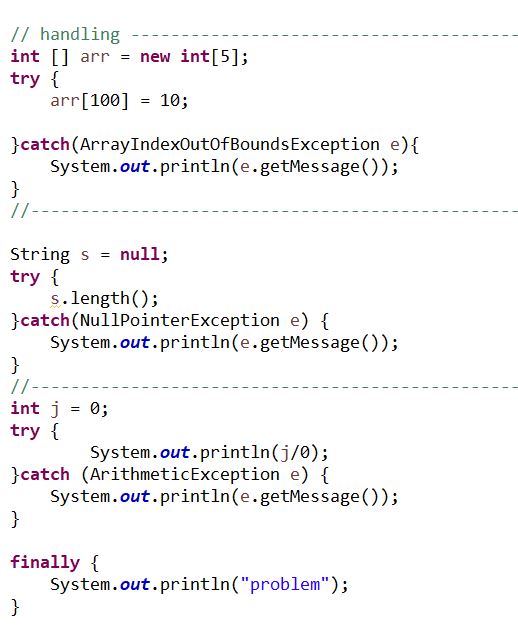
1. Checked Exception: Identified by Java Compiler before execution.

Ex IOException, FileNotFound Exception ……

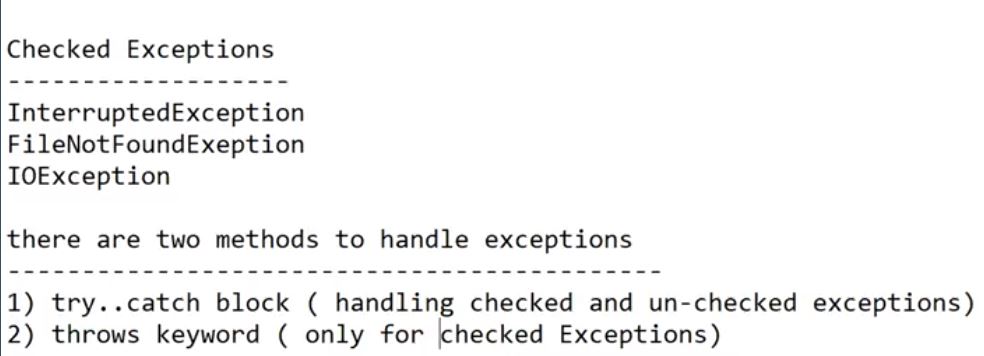
1. Unchecked Exception: Exceptions which are not identified with java compiler.

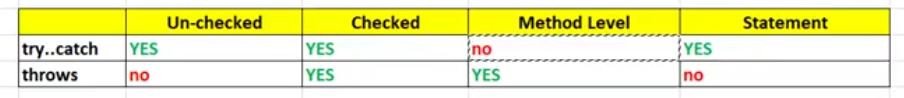
ArithmeticException, nullPointer Exception, ArrayOutofBound Exception, …...





For checked Exception we have two methods to handle an Exception





ArrayList

Hash map

Array limitations

1- Fixed in size => to overcome this limitation we use ArrayList

2- can hold same type of elements => to overcome this we use array of Object

ArrayList is a data structure un java.

ArrayList has no limit.

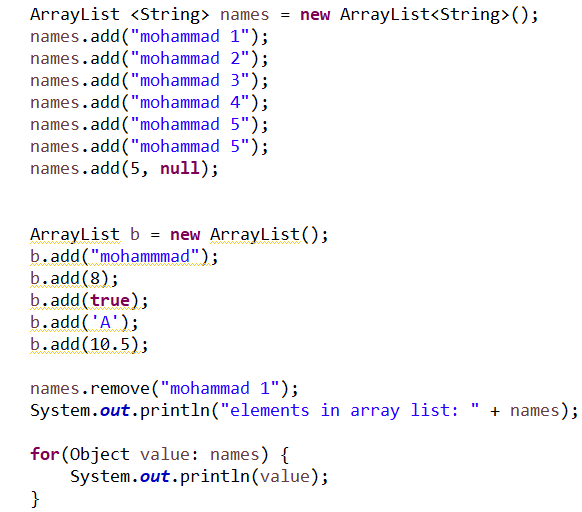
We also can store any kind of data type elements

Int a[] = new int[5] // normal array declaration

ArrayList b = new ArrayList(); allows all data type elements

ArrayList<String> c = new ArrayList<String>(); // allows only for string type elements

ArrayList<Integer> d = new ArrayList<integer>(); // allow only of integer type elements



Hash Map

HashMap contains key and value.

Key and their value called keypair.

Key ----------------🡪 value

EMPNAME Mohammad

EMP\_no 123

DEP\_No 15