

**CLASS :-** Class generates objects when it retrieves request for new keyword.

**new :-** this sends the request to the class to generate the objects.

**object :-** once object is created new keyword will get address and stores in reference variable.

**Code :-**

```
public class A{
    public static void main(String[] args){
        A a1 = new A();
        System.out.println(a1);

        A a2 = new A();
        System.out.println(a2);

        A a3 = new A();
        System.out.println(a3);
    }
}
```

Output :-  
A@7d417077  
A@3cb5cdba

## Non static:-

- Non static variables are created outside the methods and inside the class without static keyword.
- Without object creation we cannot access non static variables hence below program gives error.

**Code:-**

```
public class A{
    int x = 10;
    public static void main(String[] args){
        System.out.println(x); //Error
    }
}
```

Output:-

Error: Unresolved compilation problem: Cannot make a static reference to the non-static field x

- Every time we create an object, Copy of non Static member is loaded in to object as shown in below example.
- Non static variables belongs to objects.

**Code:-**

```
public class A{
    int x = 10;
    public static void main(String[] args){
        A a1 = new A();
        A a2 = new A();
        a2.x = 20;
        System.out.println(a1.x); //10
        System.out.println(a2.x); //20
    }
}
```

## **static:-**

- These variables are created outside methods but inside class using static keyword.
- These variables are accessed using class name as shown in below example.
- These variables have only one copy.
- If we change the value of static variable then the value change everywhere. Syntax to change value of static variable [A.y = 30].

**Code:-**

```
public class A {
    static int y = 20; //static variable
    public static void main(String[] args) {

        System.out.println(A.y); //20
    }
}
```

Output:

10  
20

# Memory Management in java

**Garbage collector** :- it removes unused objects from the memory. The objects which are not used in the code.

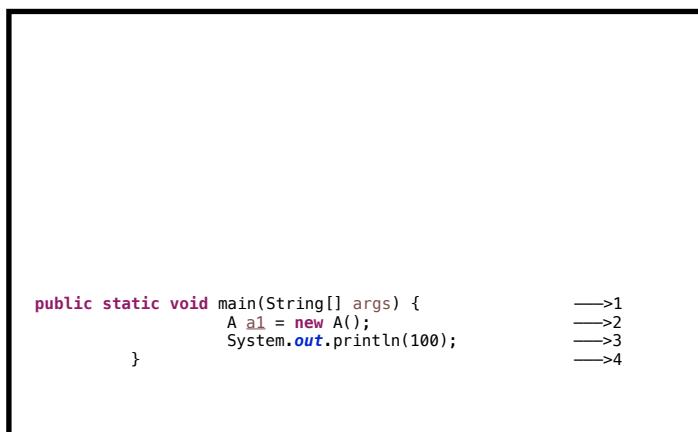
- In java the program execution starts with the opening brackets of the main method, And ends with the closing bracket of the main method.

The main method of the program will stay in the stack memory till the closing bracket of the main method executes, once the closing bracket of the main method will be executed then the whole program will be removed from stack memory.

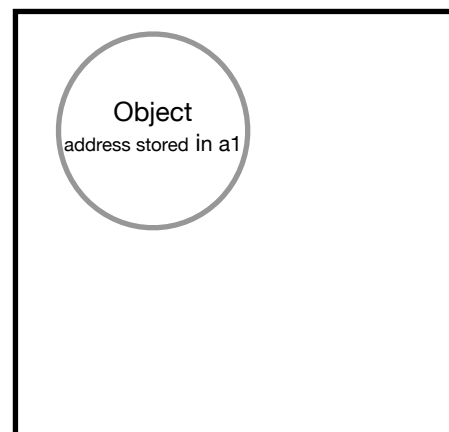
```
public class A{  
    public static void main(String[] args) { —>1  
        A a1 = new A(); —>2  
        System.out.println(100); —>3  
    } —>4  
}
```

Output:-  
100

## Stack



## Heap



## Stack :-[FILO or LIFO]

- Stack contains the execution Flow of the program.
- When the second line of the code executes then the object is created in the Heap memory.
- Once the 4th step of program executed then the entire execution flow of the program removes from the Stack, then the Address of the object created in the Heap memory will also be removed.

- Of the address of the object is not present in the Stack memory then that object is unused and it is eligible for garbage collection.

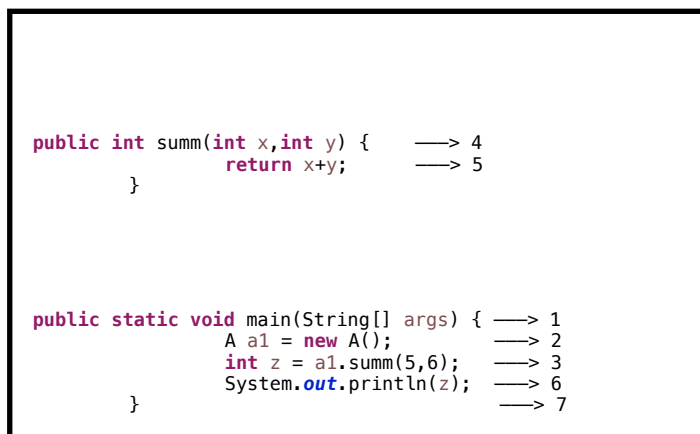
**Code:-**

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        int z = a1.summ(5,6);
        System.out.println(z);
    }
    public int summ(int x,int y) {
        return x+y;
    }
}
```

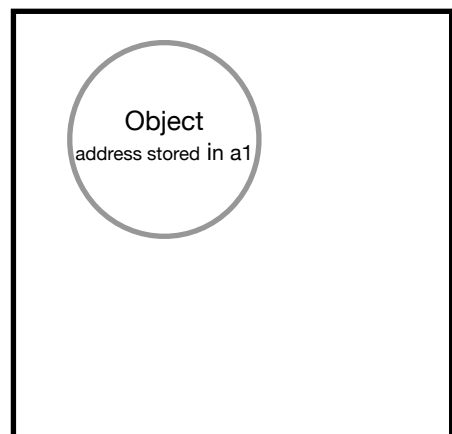
**Output:-**

11

## Stack



## Heap



- Here, first the execution flow of main method will be stored in the Stack memory
- Once the 2nd line of the code executes an object will be created in the Heap memory whose address is stored in Stack
- When the main method calls the other method then the execution of that method will be stored in Stack, After the execution of closing bracket of that method then that method will be removed from the stack memory
- At the end closing bracket of the main method executed then main method from Stack also removed and address of the object will be no longer in Stack
- Then the object with no address will be removed from the Heap memory

# Types of variables in JAVA

- Local Variable
- Static Variable
- Non Static variable
- Reference Variable

## Local Variable:-

These variables are created inside the methods and should be used only within the created method. Outside the method if used then it will throw an error as shown in below program.

- without initialising local variable we can't use it.

```
public class A{
    public static void main(String[] args) {
        int x = 10; // Local variable created
        System.out.println(x);
    }
    public void test() {
        System.out.println(x); //Error
    }
}
```

Output:-

10 —> //for print statement in main method

Returns error for the print statement in **test** method hence x is a local variable

## Static Variable:-

These variables have global access and can be used anywhere in the program.

- it is not mandatory to initialise Static variables. If not initialised then it takes default value based on the data type.
- Static variables belong to Class

**We can access Static variables in 3 ways:-**

1. ClassName.VariableName
2. VariableName
3. ObjectReference.VariableName (Wrong but will work)

```
public class A{
    static int x = 10; //if not initialised then x value will be 0
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(x); // Using variableName
        System.out.println(A.x); // Using ClassName
        System.out.println(a1.x); // Using ObjectReference
        a1.test();
    }
    public void test() {
        A a2 = new A();
        System.out.println(a2.x); // Using ObjectReference
    }
}
```

} Output:- 10/n 10/n 10/n 10/n

## non static variables:-

These variables has global access and can used any where in the program.

- it is not mandatory to initialise non static variables. If not initialised then it'll take default value based in the Data type.
- We need to use object's address to access these variables.

```
public class A{
    int x = 10;
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1.x);
        a1.test();
    }
    public void test() {
        A a2 = new A();
        System.out.println(a2.x);
    }
}
```

Output:

10  
10

## Reference Variable:-

The reference variable can store objects addressor null value.

- A Reference variable can be Static or local.
- Data type of Reference variable is ClassName.

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();//Reference variable storing object add.
        A a2 = null;// Reference variable containing null value
        System.out.println(a1);
        System.out.println(a2);
    }
}
```

Output:-

A@7d417077  
null

```
public class A{
    static A a1 = new A();// Static Reference variable
    public static void main(String[] args) {
        System.out.println(A.a1);
        A.a1.test();
    }
    public void test() {
        System.out.println(A.a1);
    }
}
```

Output:-

A@7d417077  
A@7d417077

# Data Types in JAVA

In Java, data types are categorized into two main types: **primitive data types** and **reference data types**.

## 1. Primitive Data Types

Java has eight primitive data types, each serving a specific purpose and optimized for performance:

1. **byte:**
  - Size: 8-bit
  - Range: -128 to 127
  - Example: `byte b = 100;`
2. **short:**
  - Size: 16-bit
  - Range: -32,768 to 32,767
  - Example: `short s = 10000;`
3. **int:**
  - Size: 32-bit
  - Range:  $-2^{31}$  to  $2^{31}-1$
  - Example: `int i = 100000;`
4. **long:**
  - Size: 64-bit
  - Range:  $-2^{63}$  to  $2^{63}-1$
  - Example: `long l = 100000L;`
5. **float:**
  - Size: 32-bit
  - Precision: Approximately 7 decimal digits
  - Example: `float f = 234.5f;`
6. **double:**
  - Size: 64-bit
  - Precision: Approximately 15 decimal digits
  - Example: `double d = 123.4;`
7. **boolean:**
  - Size: 1-bit
  - Values: `true` or `false`
  - Example: `boolean b = true;`
8. **char:**
  - Size: 16-bit (Unicode)
  - Range: `'\u0000'` to `'\uffff'`
  - Example: `char c = 'A';`

## 2. Reference Data Types

Reference data types are used to refer to objects. They are not predefined like primitive types and are created using constructors of the classes.

### 1. Class:

Example: `String str = "Hello";`

### 2. Interface:

Example: `Runnable r = new Runnable() { public void run()  
{ /*...*/ } };`

### 3. Array:

Example: `int[] arr = {1, 2, 3, 4};`

### 4. Enum:

Example:

```
enum Color { RED, GREEN, BLUE }  
Color c = Color.RED;
```

## Differences Between Primitive and Reference Data Types

### 1. Memory Allocation:

- **Primitive types** are stored in the stack memory.
- **Reference types** are stored in the heap memory.

### 2. Default Values:

- **Primitive types** have default values (e.g., 0 for numeric types, `false` for boolean).
- **Reference types** have `null` as their default value.

### 3. Operations:

- **Primitive types** can be used with arithmetic and logical operations.
- **Reference types** are objects and can call methods and use other object-oriented features.

## Var type:-

- This feature is introduced in java version 10
- A var type can store any kind of value in it
- Var is not a keyword in java
- Var type can not be static or non static, it can only be local variable



```

public class A{
    public static void main(String[] args) {
        var x1 = 'a';
        var x2 = "hello";
        var x3 = 10;
        var x4 = 5.5;
        var x5 = false;
        var x6 = new A();
        System.out.println(x1);
        System.out.println(x2);
        System.out.println(x3);
        System.out.println(x4);
        System.out.println(x5);
        System.out.println(x6);
    }
}

```

Output:-

```

a
hello
10
5.5
false
A@2d209079

```

#### Limitation of var:-

- var can not be static or non static

```

public class A{
    static var x1 = 'a'; // ERROR
    var x2 = "hello"; //ERROR
    public static void main(String[] args) {
        var x3 = 100 ; //Used
    }
}

```

# Methods

We need to call method, Then only method will run

**Code:-**

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        a1.test1(); // call using object address
        A.test2(); // call using class name
    }
    public void test1() { // non static method
        System.out.println("Test1 Output");
    }
    public static void test2() { // static method
        System.out.println("Test2 Output");
    }
}
```

**Output:-**

Test1 Output  
Test2 Output

Here,

- if we want to call a non static method then we have to create an object and then with the object address we have to call the method.
- If we want to call a static method then we can call the method using class name.
- We have different types of methods like **void**, **int**, **string**, **float**, **etc.**

**NOTE:-** A method return type can not be var

**Void method:-**

- A void method can never return a value.

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        a1.test();
    }
    public void test() {
        System.out.println("Hello");
    }
}
```

Output:-  
Hello

## Int Method:-

- int methods return the value of integer type

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        int x = a1.test();
        System.out.println(x);
    }
    public int test() {
        return 100;
    }
}
```

Output:-  
100

## String method:-

- String method return the value of String type

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        String x = a1.test();
        System.out.println(x);
    }
    public String test() {
        return "Hi";
    }
}
```

Output:-  
Hi

## Float Method:-

- float method returns the value of float type

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        float x = a1.test();
        System.out.println(x);
    }
    public float test() {
        return 5.5f;
    }
}
```

Output:-  
5.5

## Void Method with arguments:-

- these type of methods take argument when the used to call

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        a1.test(5, 6);
    }
    public void test(int x, int y) {
        int z = x+y;
        System.out.println(z);
    }
}
```


Output:-  
11

- o Like this we can take n number of arguments for all the methods but the value that should return by the method should be as same as the method return type

## return:-

- We can use return keyword inside void method only, It is optional.
- It will return control to method calling statement.

**NOTE :-** If we write anything after return Keyword then code will never run and we will get unreachable code error.

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        a1.test();
        System.out.println("World")
    }
    public void test() {
        System.out.println("Hello");
        return;  returns control to main method
    }
}
```

Output:-  
Hello  
World

## return value :-

- We can use return value Keyword inside not a void method only, and it is mandatory to use.
- It will return both control and value to method calling statement.

```
public class A{  
    public static void main(String[] args) {  
        A a1 = new A();  
        int x = a1.test();  
        System.out.println(x);  
    }  
    public int test() {  
        return 100;—> returns control and value to main method  
    }  
}
```

Output:-  
100

# Constructors in Java

- Constructor should have same name as that of class.
- Whenever we create object constructor will be called as shown in below example.
- Constructors are void and never return any value.

## Example 1

```
public class A{  
    A(){  
        System.out.println('A');  
    }  
    public static void main(String[] args) {  
        A a1 = new A();  
    }  
}
```

Output:-

A

## Example 2

```
public class A{  
    A(){  
        System.out.println('A');  
    }  
    public static void main(String[] args) {  
        A a1 = new A();  
        A a2 = new A();  
        A a3 = new A();  
    }  
}
```

Output:-

A

A

A

## Example 3

```
public class A{  
    A(int x){  
        System.out.println(x);  
    }  
    public static void main(String[] args) {  
        A a1 = new A(100);  
    }  
}
```

Output:-

100

#### Example 4

```
public class A{
    A(int x){
        return 100 ; //Error
    }
    public static void main(String[] args) {
        A a1 = new A(100);
    }
}
```

#### Example 5

```
public class A{
    A(int x){
        System.out.println(x);
        return;
    }
    public static void main(String[] args) {
        A a1 = new A(100);
    }
}
```

Output:-

### Constructor Overloading:-

- Here we create more than one constructor in a same class provided they have different number of arguments or different type of arguments.

#### Example 1: Using Different number of arguments

```
public class A{
    A(){
        System.out.println("Hello");
    }
    A(int x){
        System.out.println(x);
    }
    A(int x, int y){
        System.out.println(x);
        System.out.println(y);
    }
    public static void main(String[] args) {
        A a1 = new A();
        A a2 = new A(100);
        A a3 = new A(100,200);
    }
}
```

Output:-

```
Hello
100
100
200
```

## Example 2: Using different type of arguments but same number

```
public class A{
    A(int x){
        System.out.println(x);
    }
    A(String x){
        System.out.println(x);
    }
    A(float x){
        System.out.println(x);
    }
    public static void main(String[] args) {
        A a1 = new A(10);
        A a2 = new A("Hello");
        A a3 = new A(5.5f);
    }
}
```

Output:-

10

Hello

5.5



# this keyword

- **this** keyword holds object address or in other words it acts like a reference variable which is automatically created

## Example 1

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1);
        a1.test();
    }
    private void test() {
        System.out.println(this);
    }
}
```

Output:-

A@7d417077

A@7d417077

- **this** keyword holds the address of current object running in the program.

## Example 2

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1);
        a1.test();
        A a2 = new A();
        System.out.println(a2);
        a2.test();
    }
    private void test() {
        System.out.println(this);
    }
}
```

Output:

app\_java\_1.A@7d417077 // address a1

app\_java\_1.A@7d417077 // address in this keyword

app\_java\_1.A@3cb5cdba // address a2

app\_java\_1.A@3cb5cdba // address in this keyword

- We can call non static method without creating an object with the help of **this** keyword.[Example 3]
- If we call any non static method inside another non static method without creating an object or **this** keyword then by default this keyword will be used . [Example 4]

### Example 3

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        a1.test1();
    }
    public void test1() {
        this.test2(); // Using this keyword
    }
    private void test2() {
        System.out.println(100);
    }
}
```

Output:-  
100

### Example 4

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        a1.test1();
    }
    public void test1() {
        test2(); // Using this keyword by default
    }
    private void test2() {
        System.out.println(100);
    }
}
```

Output:-  
100

### Limitations of this keyword:-

- We cannot use this keyword in static methods

### Example 1

```
public class A{
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(this); //ERROR
    }
    private static void test() {
        System.out.println(this); //ERROR
    }
}
```

# this()

## Constructor chaining :-

- It is used to call constructor form another constructor

### Example 1

```
public class A{
    A(){
        System.out.println("Hello");
    }
    A(int x){
        this();
    }
    public static void main(String[] args){
        A a1 = new A(100);
    }
}
```

Output:-

Hello

### Example 2

```
public class A{
    A(){
        this(100);
    }
    A(int x){
        System.out.println(x);
    }
    public static void main(String[] args) {
        A a1 = new A();
    }
}
```

Output:-

100

### Example 3

```
public class A{
    A(){
        System.out.println("Hello");
    }
    A(int x){
        this();
    }
    A(int x,int y){
        this(x);
    }
    public static void main(String[] args){
        A a1 = new A(1,2);
    }
}
```

Output:-

Hello

## Limitations of this() Keyword:-

- While calling a constructor this() keyword should be the first statement.

### Example 1

```
public class A{
    A(){
        System.out.println("Hello");
    }
    A (int x){
        System.out.println(100);
        this();//ERROR
    }
}
```

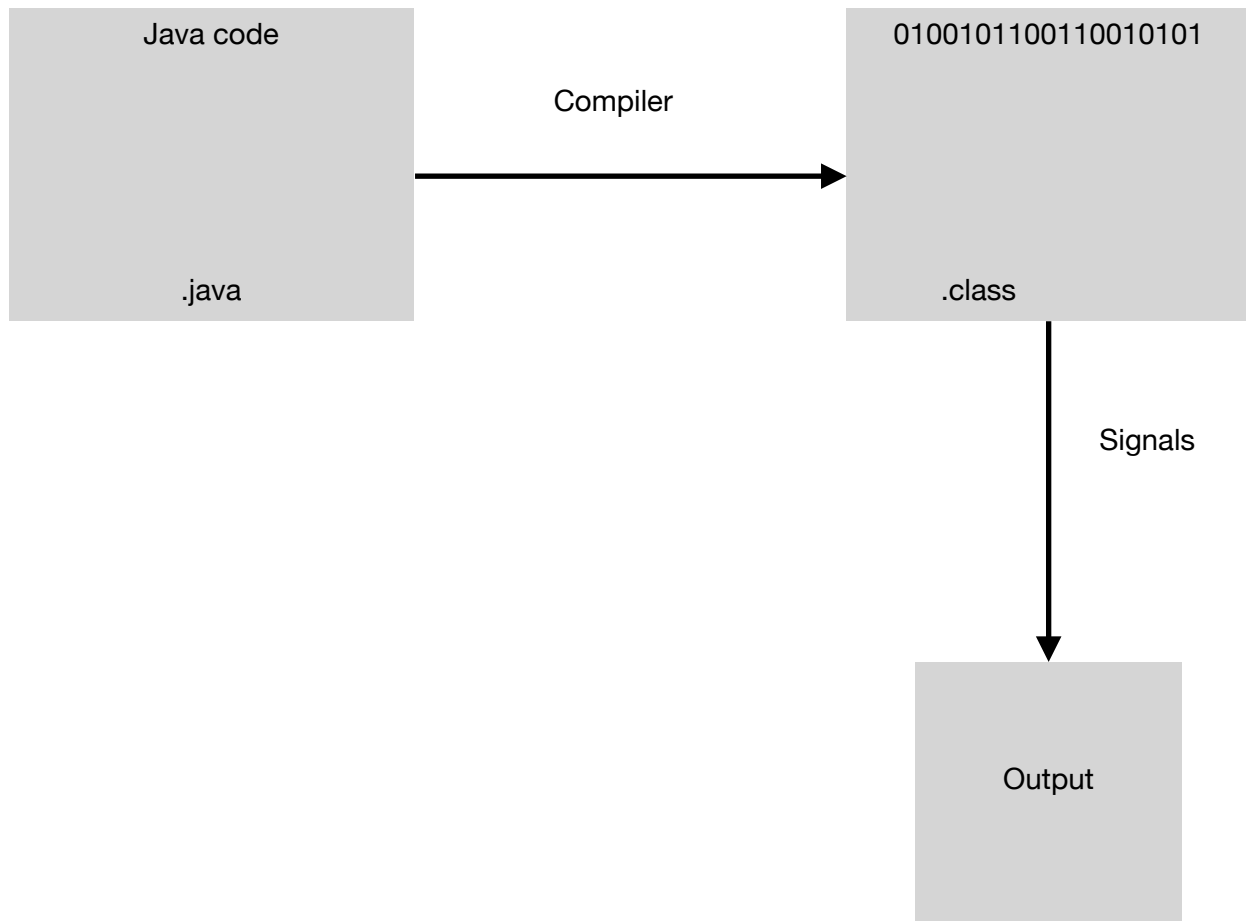
### Example 2

```
public class A{
    A(){
        System.out.println("Hello");
    }
    A (int x){
        this();
        System.out.println(100);
    }
    public static void main(String[] args) {
        A a1 = new A(100);
    }
}
```

Output:-

```
Hello
100
```

- \* During compilation we convert .java file to .class file with the help of compiler present inside the JDK folder.
- \* During the run time we run .class file with the help of JER(Java Run time Environment present inside JER folder).



## Default Constructor:-

- If we do not create constructor in .java file then during the runtime automatically empty body constructor with noArgs will get added in .class file. This is called as **Default constructor**.

**Note:-** During object creation it is mandatory to call constructor or else else object will not get created.

```

Class A {

    psvm(){
        A a1 = new A();
    }

}
  
```

A.java

```

Class A {
    A(){ // noArgs Constructor
    }
    psvm(){
        A a1 = new A();
    }
}
  
```

A.class

# Oops in JAVA

## Inheritance

- Here we inherit the members of parent class to child class so that we can re use parent class functionalities into child class.

### Example 1

#### Animal Class

```
public class Animal{  
    public void eat() {  
        System.out.println("Eating....");  
    }  
    public void sleep() {  
        System.out.println("Sleeping....");  
    }  
}
```

#### Dog Class

```
public class Dog extends Animal {  
    public static void main(String[] args) {  
        Dog d = new Dog();  
        d.eat();  
        d.sleep();  
        d.noice();  
    }  
}
```

Output:-

Eating...  
Sleeping...

#### Cat Class

```
public class Cat extends Animal {  
    public static void main(String[] args) {  
        Cat c = new Cat();  
        c.eat();  
        c.sleep();  
        c.noice();  
    }  
}
```

Output:-

Eating...  
Sleeping...

- Inheritance is tightly coupled code.

## Example 2

### Animal Class

```
public class Animal{
    public void eat() {
        System.out.println("Eating...");
    }
    public void sleep() {
        System.out.println("Sleeping...");
    }
}
```

### Dog Class

```
public class Dog extends Animal {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();
        d.sleep();
        d.noice();
    }
    public void noice() {
        System.out.println("Bow...Bow...");
    }
}
```

Output:-

Eating...

Sleeping...

Bow...Bow...

### Cat Class

```
public class Cat extends Animal {
    public static void main(String[] args) {
        Cat c = new Cat();
        c.eat();
        c.sleep();
        c.noice();
    }
    public void noice() {
        System.out.println("Meow...Meow...");
    }
}
```

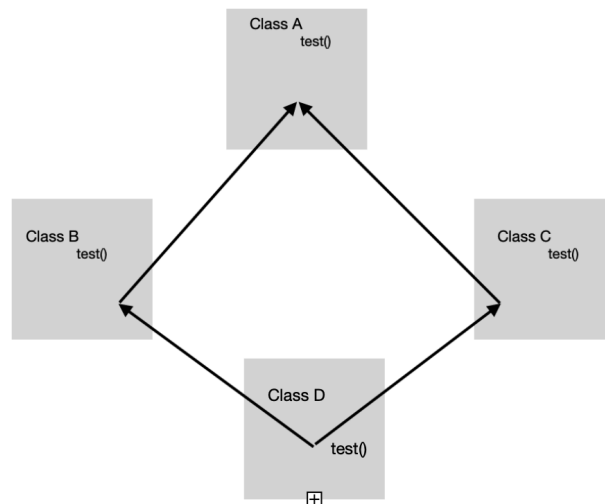
Output:-

Eating...

Sleeping...

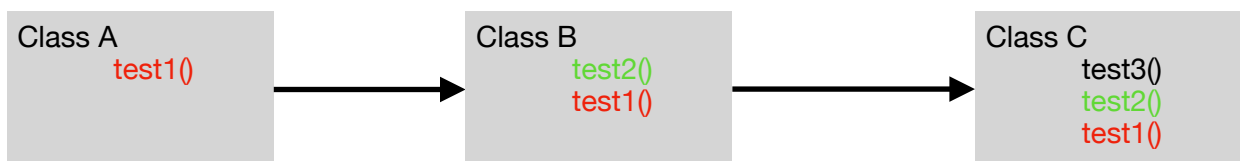
Meow...Meow...

- Java does not support multi inheritance of class because this results in diamond problem, but **interface** in java supports multiple inheritance.



- IN the above diagram test() method form class A is inherited to class B and then class D. From class A test() method is inherited to class C and then to class D. Which leads to confusion form where test() method is inherited.

## Multilevel Inheritance



```

public class A {
    public void test1() {
        System.out.println(100);
    }
}

public class B extends A {
    public void test2() {
        System.out.println(200);
    }
}

public class C extends B {
    public static void main(String[] args) {
        C c1 = new C();
        c1.test1();
        c1.test2();
        c1.test3();
    }
    public void test3() {
        System.out.println(300);
    }
}
  
```

Output:-

```

100
200
300
  
```



# Packages in Java

- Folders that we create to store programs in organised manner.
- To use the class present in different package we will have to perform import as shown in below example.

```
package p1;  
public class A {  
  
}
```

```
package p1;  
public class B {  
  
}
```

```
package p2;  
import p1.A;  
public class c extends A {  
  
}
```

- Now in this scenario we have two packages p1 and p2, To create object of a class present in package p1 in package p2 by importing package p1 in package p2.

```
package p1;  
public class A {  
  
}
```

```
package p1;  
public class B {  
  
}
```

```
package p2;  
import p1.A;  
public class c extends A {  
    public static void main(String[] args) {  
        A a1 = new A();  
    }  
}
```

- By performing import “ packageName.\* ”, All the classes present in that package will be imported.

```
package p1;  
public class A {  
  
}
```

```
package p1;  
public class B {  
  
}
```

```
package p2;  
import p1.*;  
public class c extends A {  
    public static void main(String[] args) {  
        A a1 = new A();  
        B b1 = new B();  
    }  
}
```

- We can also use without importing by using their packageName

```
package p1;  
public class A {  
  
}
```

```
package p2;  
public class B {  
  
}
```

```
package p2;  
public class c extends A {  
    public static void main(String[] args) {  
        p1.A a1 = new p1.A();  
        p2.B b1 = new p2.B();  
    }  
}
```

# Access Specifiers

- \* Private
- \* Default
- \* Protected
- \* Public

	Private	Default	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non subclass	No	No	No	Yes

## Private:-

### Same class

```
package p1;
public class A{
    private int x = 10;
    private void test() {
        System.out.println(100);
    }
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1.x);
        a1.test();
    }
}
```

Output:-

```
10
100
```

### Class A in p1 is common for all examples

```
package p1;
public class A{
    private int x = 10;
    private void test() {
        System.out.println(100);
    }
}
```

## Same package subclass

```
package p1;
public class B extends A {
    public static void main(String[] args) {
        B b1 = new B();
        b1.test(); // ERROR
        System.out.println(b1.x); // ERROR
    }
}
```

## Same package non sub class

```
package p1;
public class B {
    public static void main(String[] args) {
        A a1 = new A();
        a1.test(); // ERROR
        System.out.println(a1.x); // ERROR
    }
}
```

## Different package subclass

```
package p2;
import p1.A;
public class C extends A{
    public static void main(String[] args) {
        C c1 = new C();
        c1.test(); //ERROR
        System.out.println(c1.x1); //ERROR
    }
}
```

## Different package non subclass

```
package p2;
import p1.A;
public class C{
    public static void main(String[] args) {
        A a1 = new A();
        a1.test(); //ERROR
        System.out.println(a1.x1); //ERROR
    }
}
```

## Default:-

### Same Class

```
package p1;
public class A{
    int x = 10;
    void test() {
        System.out.println(100);
    }
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1.x);
        a1.test();
    }
}
```

Output:-

```
10
100
```

### Same package subclass

From here class A in p1 is same for all examples

```
package p1;
public class A{
    int x = 10;
    void test() {
        System.out.println(100);
    }
}
package p1;
public class B extends A {
    public static void main(String[] args) {
        B b1 = new B();
        b1.test();
        System.out.println(b1.x);
    }
}
```

Output:-

```
100
10
```

### Same package non subclass

```
package p1;
public class B {
    public static void main(String[] args) {
        A a1 = new A();
        a1.test();
        System.out.println(a1.x);
    }
}
```

Output:- 100 10

## Different package subclass

```
package p2;
import p1.A;
public class C extends A{
    public static void main(String[] args) {
        C c1 = new C();
        c1.test(); //ERROR
        System.out.println(c1.x1); //ERROR
    }
}
```

## Different package non subclass

```
package p2;
import p1.A;
public class C{
    public static void main(String[] args) {
        A a1 = new A();
        a1.test(); //ERROR
        System.out.println(a1.x1); //ERROR
    }
}
```

## Protected:-

### Same Class

```
package p1;
public class A {
    protected int x = 10;
    protected void test() {
        System.out.println(100);
    }
    public static void main(String[] args) {
        A a1 = new A();
        a1.test();
        System.out.println(a1.x);
    }
}
```

Output:-

100

10

## Class A for all examples

```
package p1;
public class A {
    protected int x = 10;
    protected void test() {
        System.out.println(100);
    }
}
```

## Same package subclass

```
package p1;
public class B extends A {
    public static void main(String[] args) {
        B b1 = new B();
        b1.test();
        System.out.println(b1.x);
    }
}
Output:-
100
10
```

## Same package non subclass

```
package p1;
public class B {
    public static void main(String[] args) {
        A a1 = new A();
        a1.test();
        System.out.println(a1.x);
    }
}
Output:-
100
10
```

## Different package subclass

```
package p2;
import p1.A;
public class C extends A {
    public static void main(String[] args) {
        C c1 = new C();
        c1.test();
        System.out.println(c1.x);
    }
}
Output:-
100
10
```

## Different package non subclass

```
package p2;
import p1.A;
public class C {
    public static void main(String[] args) {
        A a1 = new A();
        a1.test(); //Error
        System.out.println(a1.x); //Error
    }
}
```

## Public:-

- For public variables and methods we can access from any where by using their class objects or their subclass objects.

## Access Specifiers for class

- A class can be **public** or **default** only

```
// Public class
package p1;
public class A{
    // Created successfully
}
```

```
// Default class
package p1;
class A{
    // Created successfully
}
```

- A class cannot be **private** or **protected**

```
// private class
package p1;
private class A{ //Error
}
```

```
// protected class
package p1;
protected class A{ //Error
}
```



# Access specifiers on constructors

## Private constructor:-

- If you make constructor private then it's object cannot be created outside the class.

```
package p1;
public class A {
    private A() {

    }
    public static void main(String[] args) {
        A a1 = new A();
    }
}
package p1;
public class B {
    public static void main(String[] args) {
        A a1 = new A(); //Error
    }
}
package p2;
import p1.A;
public class C {
    public static void main(String[] args) {
        A a1 = new A(); //Error
    }
}
```

- If you make constructor private then that class is not eligible for inheritance.

```
package p1;
public class A {
    private A() {

    }
}
package p1;
public class B extends A{ //Error
}
package p2;
import p1.A;
public class C extends A{ //Error
}
```

## Default constructor:-

- If you make your constructor Default then its object cannot be created outside the package.

```
package p1;
public class A {
    A() {
    }
    public static void main(String[] args) {
        A a1 = new A();
    }
}

package p1;
public class B {
    public static void main(String[] args) {
        A a1 = new A();
    }
}

package p2;
import p1.A;
public class C{
    public static void main(String[] args) {
        A a1 = new A();//Error
    }
}
```

- If you make a Constructor Default then that class is not eligible for inheritance outside the package.

```
package p1;
public class A {
    A() {
    }
}

package p1;
public class B extends A{
}

package p2;
import p1.A;
public class C extends A{ // Error
}
}
```

## Protected Constructor:-

- If you make a constructor protected then its object cannot be created outside the package.

```
package p1;
public class A {
    protected A() {
    }
    public static void main(String[] args) {
        A a1 = new A();
    }
}

package p1;
public class B{
    public static void main(String[] args) {
        A a1 = new A();
    }
}

package p2;
import p1.A;
public class C {
    public static void main(String[] args) {
        A a1 = new A();// Error
    }
}
```

- If you make a constructor protected then that class is eligible for inheritance outside the package also.

```
package p2;
import p1.A;
package p1;
public class A {
    protected A() {
    }
}

package p1;
public class B extends A{
}

package p2;
import p1.A;
public class C extends A{
}
```

## Public constructor:-

- If you make a constructor public then its object can be created in the same and different package

```
package p1;
public class A {
    public A() {
    }
    public static void main(String[] args) {
        A a1 = new A();
    }
}

package p1;
public class B{
    public static void main(String[] args) {
        A a1 = new A();
    }
}

package p2;
import p1.A;
public class C {
    public static void main(String[] args) {
        A a1 = new A();
    }
}
```

- If you make a constructor public then that class is eligible for inheritance in same and different package.

```
package p1;
public class A {
    public A() {
    }
}

package p1;
public class B extends A{
}

package p2;
import p1.A;
public class C extends A{
}
```

# Unary operators

We have two types of unary operators

- Incrementation
- Decrementation

## Incrementation

We have two types of incrementation

- Post increment
- Pre increment

### Post increment:-

In this process the value of the variable will be incremented when the value appears next time in the program.

```
package p1;
public class A {
    public static void main(String[] args) {
        int i = 10;
        int j = i++ + i++; // 10++ + 11++
        System.out.println(i); // i = 12
        System.out.println(j);
    }
}
```

Output:

12  
21

### Pre Increment:-

In this process the value of the variable will be incremented first and then moves to next step.

```
package p1;
public class A {
    public static void main(String[] args) {
        int i = 10;
        int j = ++i + ++i; // ++10(11) + ++11(12)
        System.out.println(i); // i = 12
        System.out.println(j);
    }
}
```

Output:

12  
23

## Decrementation

We have two types of Decrementation

- Post Decrement
- Pre Decrement

### Post Decrement:-

In this process the value of the variable will be decremented when the value appears next time in the program.

```
package p1;
public class A {
    public static void main(String[] args) {
        int i = 10;
        int j = i-- + i--; // 10-- + 9--
        System.out.println(i); // i = 8
        System.out.println(j);
    }
}
```

Output:-

8  
19

### Pre Decrement:-

In this process the value of the variable will be decremented first and then moves to next step.

```
package p1;
public class A {
    public static void main(String[] args) {
        int i = 10;
        int j = --i + --i; // --10(9) + --9(8)
        System.out.println(i); // i = 8
        System.out.println(j);
    }
}
```

Output:-

8  
17

# Data Types and Type casting

	Memory size	Default value
byte	1	0
short	2	0
int	4	0
long	8	0
float	4	0,0
double	8	0,0
boolean	N/A	flase
char	2	Blank
String	N/A	null

## Type Casting:-

converting a particular data type into require data type is called as type casting.

### 1. Auto Upcasting

- Converting the smaller memory data type to bigger memory data type without any loss of data is called as Auto Upcasting.

```
public class A {  
    public static void main(String[] args) {  
        int i = 10;  
        long j = i;  
        System.out.println(j);  
    }  
}
```

Output:

10

- When ever there is loss of data regardless of memory size Upcasting cannot happen.

```
public class A {  
    public static void main(String[] args) {  
        float i = 10.3f;  
        long j = i; //Error  
        System.out.println(j);  
    }  
}
```

## 2. Explicit Down Casting

- Converting bigger memory data type to smaller memory data type is called as Explicit Down Casting.

```
public class A {  
    public static void main(String[] args) {  
        int i = 10;  
        short j = (short)i;  
        System.out.println(j);  
    }  
}
```

Output:-  
10

- During conversion if there is nay loss of data then regardless of memory size we need to perform Explicit Down Casting

```
public class A {  
    public static void main(String[] args) {  
        float i = 10.3f; // we will loss .3 in this conversion  
        int j = (int)i;  
        System.out.println(j);  
    }  
}
```

Output:-  
10

- Here while the process of Auto Upcasting it will automatically cast the value for smaller memory data type to bigger, if we do it manually it will also work as shown in below example.

```
public class A {  
    public static void main(String[] args) {  
        int i = 10;  
        long j = (long)i;  
        System.out.println(j);  
    }  
}
```

Output:  
10



## Explicit down casting String(class) to other datatypes using wrapper classes

### String to int:-

```
public class A{
    public static void main(String[] args) {
        String x = "100";
        int y = Integer.parseInt(x); //wrapper class
        System.out.println(y);
    }
}
```

### String to Float:-

```
public class A{
    public static void main(String[] args) {
        String x = "10.1f";
        float y = Float.parseFloat(x); //wrapper class
        System.out.println(y);
    }
}
```

### String to double:-

```
public class A{
    public static void main(String[] args) {
        String x = "10.1";
        Double y = Double.parseDouble(x); //wrapper class
        System.out.println(y);
    }
}
```

### String to boolean:-

```
public class A{
    public static void main(String[] args) {
        String x = "True"; // other than true every thing will
        //return false
        boolean y = Boolean.parseBoolean(x); //wrapper class
        System.out.println(y);
    }
}
```

Output:-  
true

## String to long:-

```
public class A{
    public static void main(String[] args) {
        String x = "9515055232";
        long y = Long.parseLong(x);
        System.out.println(y);
    }
}
```

For all these type of conversions the string value is similar to their original datatype value. If the values like Alphanumeric or alphabets then we try to convert them in to integers or decimals then it will throw

**NumberFormatException.**

```
public class A{
    public static void main(String[] args) {
        String x = "Syed1234";
        long y = Long.parseLong(x); // NumberFormatException
        System.out.println(y);
    }
}
```

# Exception

when bad user input is given program will start suddenly. This is called an Exception.

```
public class A{
    public static void main(String[] args) {
        int x = 10;
        int y = 0;
        int z = x/y;//Exception
        System.out.println("Welcome");// This will not run
    }
}
```

## Exception Handling:

we use try and catch block to handle exception in java. If any line of code in try block throw an exception then try block will create exception object and that object address will give it to catch block.

- catch block will suppress that exception and hence further code will continue to run.

```
public class A{
    public static void main(String[] args) {
        try {
            int x = 10;
            int y = 0;
            int z = x/y;
            System.out.println(z);// This will not run
        }catch(Exception e){
            e.printStackTrace();
        }
        System.out.println("Welcome");// This will run
    }
}
```

Output:-

`java.lang.ArithmeticException: / by zero`

Welcome

`at syed/p1.A.main(A.java:7)java.lang.ArithmeticException: / by zero`

# Types of Exceptions in JAVA

There are two types of exceptions in java

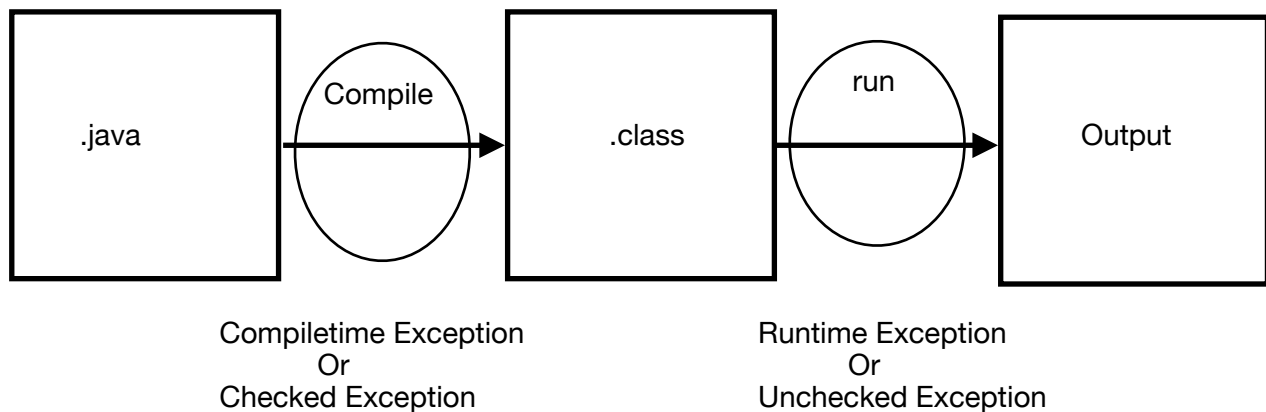
1. Compiletime Exception **or** checked Exception
2. Runtime Exception **or** unchecked Exception

## Compiletime Exception:-

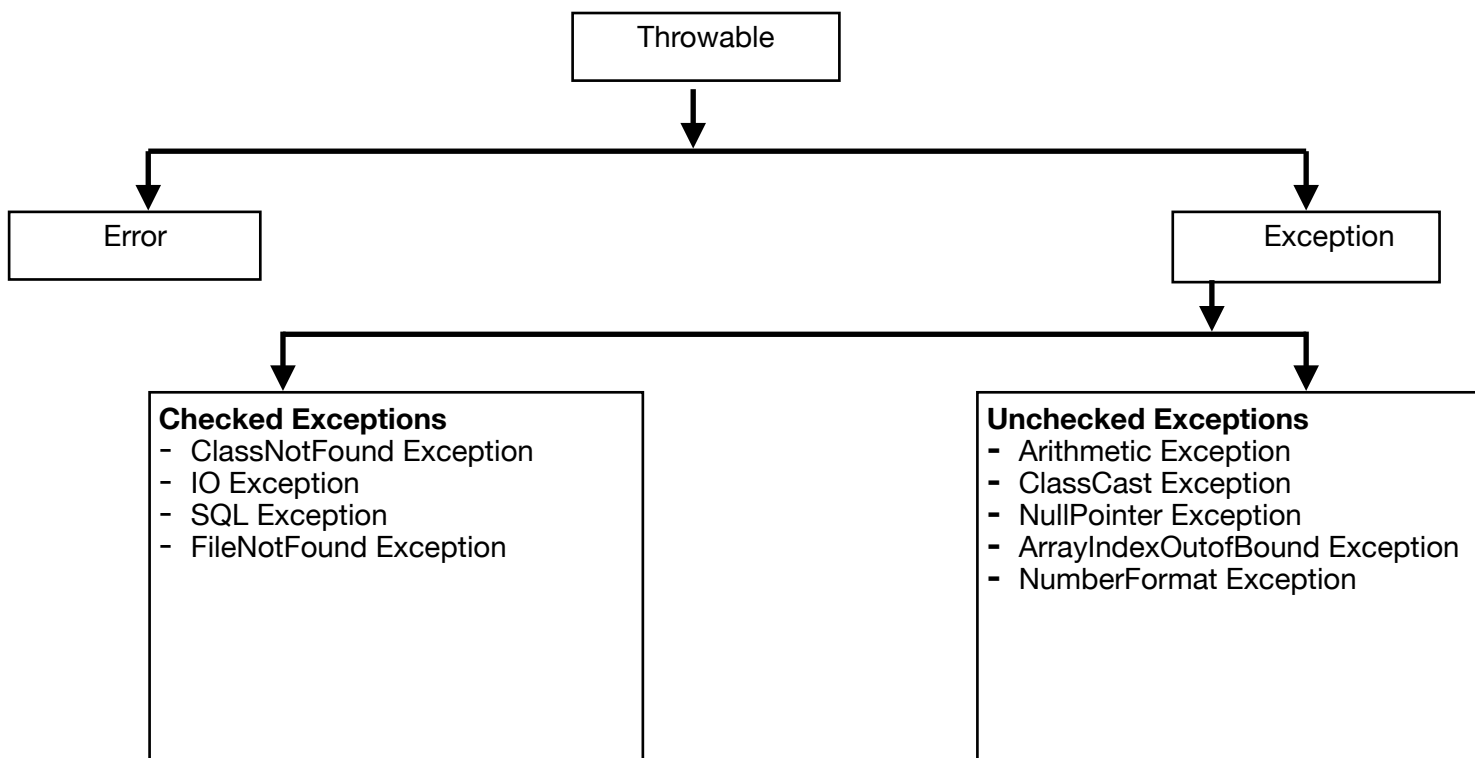
The exception will occur when you convert .java file to .class file or in other words during the compilation of the program.

## Runtime Exception:-

These Exceptions will occur when you run .java file or in other words during the runtime of program.



## Exception class Hierarchy:-



## ArithmeticException:-

This Exception occurs when invalid mathematical operation is performed like dividing a number by zero as shown in below example.

To handle this type of Exception we use ArithmeticException class.

```
public class A{
    public static void main(String[] args) {
        try {
            int x = 10;
            int y = 0;
            int z = x/y;
            System.out.println(z);
        }catch(ArithmeticException e){
            e.printStackTrace();
        }
        System.out.println("Welcome");// This will run
    }
}
```

Output:-

```
java.lang.ArithmeticException: / by zero
Welcome
at syed/p1.A.main(A.java:7)
```

## NullPointerException:-

This Exception occurs when with null reference variable we are access non static member as shown in below example.To handle this Exception we use NullPointerException class.

```
public class A{
    int x = 10;
    public static void main(String[] args) {
        try{
            A a1 = null;
            System.out.println(a1.x);
        }catch(NullPointerException e){
            e.printStackTrace();
        }
    }
}
```

Output:-

```
java.lang.NullPointerException: Cannot read field "x" because
"a1" is null
at syed/p1.A.main(A.java:7)
```

## Multi catch block:-

If a program throws multiple types of exceptions then we can use multiple catch block in a same program to handle such type of Exceptions.

```
public class A{
    int m = 13;
    public static void main(String[] args) {
        try{
            int x = 10/0;
            A a1 = null;
            System.out.println(a1.m);
            System.out.println(x);
        }catch(ArithmeticException e) {
            e.printStackTrace();
        }catch(NullPointerException e) {
            e.printStackTrace();
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

## NumberFormatException:-

When we try to convert string value to integer or float in these cases we may get NumberFormatException as shown below.

```
public class A{
    public static void main(String[] args) {
        try{
            String s = "syed";
            long l = Long.parseLong(s);
            System.out.println(l);
        }catch(NumberFormatException e) {
            e.printStackTrace();
        }
    }
}
```

# Arrays in JAVA

Ordered collection of elements is called an Array.

Arrays are fundamental datastructures that allows you to store and access multiple values under a single variable name.

```
public class A{
    public static void main(String[] args) {
        String[] fruits = new String[3];
        fruits[0] = "apple";
        fruits[1] = "orange";
        fruits[2] = "mango";
        System.out.println(fruits[0]);
        System.out.println(fruits[1]);
        System.out.println(fruits[2]);
    }
}
```

Output:-  
apple  
orange

## For loop

A “for” loop in Java is a control flow statement that allows code to be executed repeatedly based on a condition. It's commonly used for iterating over a range of values or through elements in an array or collection

```
public class A{
    public static void main(String[] args) {
        for(int i =0;i<5;i++) { // 0 to 4
            System.out.println(i);
        }
    }
}
```

Output:-  
0  
1  
2  
3  
4

Now we will print the elements of array using for loop

```
public class A{
    public static void main(String[] args) {
        String[] fruits = new String[3];
        fruits[0] = "apple";
        fruits[1] = "orange";
        fruits[2] = "mango";
        for (int i = 0; i < 3; i++) {
            System.out.println(fruits[i]);
        }
    }
}
```

Output:-

```
apple
orange
mango
```

Dynamic **for** loop:-

Here we make the loop dynamic because if the array size will change then the loop also should work based on that.

```
public class A{
    public static void main(String[] args) {
        String[] fruits = new String[3];
        fruits[0] = "apple";
        fruits[1] = "orange";
        fruits[2] = "mango";
        for (int i = 0; i < fruits.length; i++) {
            System.out.println(fruits[i]);
        }
    }
}
```

Output:-

```
apple
orange
mango
```



# Scanner Class

Scanner class is used to take user input from the keyboard.

## String Input

```
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter your Name : ");
        String name = s.next();
        System.out.println(name);
    }
}
```

Output:-

Enter your Name :

syed

syed

## Integer Input

```
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter your age : ");
        int age = s.nextInt();
        System.out.println(age);
    }
}
```

Output:-

Enter your age :

24

24

## Decimal Input

```
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter your weight : ");
        float weight = s.nextFloat();
        System.out.println(weight);
    }
}
```

Output:-

Enter your weight :

69.5

69.5

**next()** → can only read string of single word.

**nextLine()** → can read string of multiple words.

### Multiple Word input

```
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter your description : ");
        String describe = s.nextLine();
        System.out.println(describe);
    }
}
```

Output:-

Enter your description :

Hello World

Hello World

### Boolean Input

```
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter your openion : ");
        boolean openion = s.nextBoolean();
        System.out.println(openion);
    }
}
```

Output:-

Enter your openion :

true

true

## ATM pin-check (for loop)

```
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        for(int i=3;i>0;i--) {
            System.out.println("Enter your pinNumber : ");
            int pinNumber = s.nextInt();
            if(pinNumber==1234) {
                System.out.println("welcome");
                break;
            }
            else {
                System.out.println("Incorrect pinNumber");
                if(i>1) {
                    System.out.println("Attempts Remaining : "+(i-1));
                }
                else {
                    System.out.println("Card is Blocked!");
                }
            }
        }
    }
}
```

Output1:-

```
Enter your pinNumber :
1234
welcome
```

Output2:-

```
Enter your pinNumber :
1235
Incorrect pinNumber
Attempts Remaining : 2
Enter your pinNumber :
1324
Incorrect pinNumber
Attempts Remaining : 1
Enter your pinNumber :
1432
Incorrect pinNumber
Card is Blocked!
```

# While loop

```
public class A{
    public static void main(String[] args) {
        int x = 0;
        while (x<3) {
            System.out.println(x);
            x++;
        }
    }
}
```

Output:-

0  
1  
2

# do while loop

Example 1

```
public class A{
    public static void main(String[] args) {
        int x = 0;
        do {
            System.out.println(x);
            x++;
        }while (x<3);
    }
}
```

Output:-

0  
1  
2

Example 2

```
public class A{
    public static void main(String[] args) {
        int x = 100;
        do {
            System.out.println(x);
            x++;
        }while (x<3);
    }
}
```

Output:-

100

## While loop example:-

```
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        String cnd = "yes";
        while(cnd.equals("yes")) {
            System.out.print("Enter the amount : ");
            int amount = scan.nextInt();
            System.out.println("please collect the cash :"+amount);
            System.out.print("Do you want to continue[yes/no] : ");
            cnd = scan.next();
        }
        System.out.println("Thank you.visit again😊");
    }
}
```

Output:-

```
Enter the amount : 1000
please collect the cash :1000
Do you want to continue[yes/no] : yes
Enter the amount : 2000
please collect the cash :2000
Do you want to continue[yes/no] : no
Thank you.visit again😊
```

## Conditional operators:-

```
public class A{
    public static void main(String[] args) {
        System.out.println(2==3);
        System.out.println(2!=3);
        System.out.println(3>2);
        System.out.println(3<2);
        System.out.println(3>3);
        System.out.println(3>=3);
        System.out.println(2<=1);
    }
}
```

Output:-

```
false
true
true
false
false
true
false
```

# Laddered if block

```
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int x = s.nextInt();
        if(x==0) {
            System.out.println(0*2);
        }else if(x==1) {
            System.out.println(1*2);
        }else if(x==2) {
            System.out.println(2*2);
        }else {
            System.out.println("No match Found");
        }
    }
}
Output:-
2
4
```

# Switch case

```
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int x = s.nextInt();
        switch(x) {
            case 1:
                System.out.println(1);
                break;
            case 2:
                System.out.println(2);
                break;
            case 3:
                System.out.println(3);
                break;
            default:
                System.out.println("no match found");
                break;
        }
    }
}
Output:-
1
1
```

# ArrayIndexOutOfBoundsException

- This exception occurs when you try to exceed the size of an array.

```
public class A{
    public static void main(String[] args) {
        try {
            int x[] = new int[3];
            x[0]=1;
            x[1]=2;
            x[2]=3;
            x[3]=4;
        } catch (ArrayIndexOutOfBoundsException e) {
            e.printStackTrace();
        }
    }
}
```

[java.lang.ArrayIndexOutOfBoundsException](#): Index 3 out of bounds  
for length 3  
at syed/p1.A.main([A.java:10](#))

## Remove Duplicate values for an array

**Step1:-** Create **temp** array with same size that of array **x**

**Step2:-** Compare **i** with **i+1**, when not equal copy the value to temp and increment **j** value by 1 in **temp** array

**Step3:-** When you reach last index of array **x** copy that value as it is to **temp** array.

```
public class A{
    public static void main(String[] args) {
        int x[] = {1,1,2,2,3,4,4,5};
        int temp[] = new int[x.length];
        int j = 0;
        for(int i=0;i<x.length-1;i++) {
            if(x[i]!=x[i+1]) {
                temp[j]=x[i];
                j++;
            }
        }
        temp[j] = x[x.length-1];
        for(int z=0;z<=j;z++) {
            System.out.println(temp[z]);
        }
    }
}
```

Output:-

1 2 3 4 5

# Sorting of array

```
public class A{
    public static void main(String[] args) {
        int x[] = {32,24,14,2,1};
        int z = 0;
        for(int j=0;j<x.length-1;j++) {
            for(int i=0;i<x.length-1-j;i++) {
                if(x[i]>x[i+1]) {
                    z = x[i+1];
                    x[i+1] = x[i];
                    x[i] = z;
                }
            }
        }
        for(int k=0;k<x.length;k++) {
            System.out.println(x[k]);
        }
    }
}
```

Output:-

```
1
2
14
24
32
```



## Sorting and removing duplicate elements for an array

```
public class A{
    public static void main(String[] args) {
        int[] x = {30,2,30,2,12,14,8,24,14,8};
        int[] temp = new int[x.length];
        int j = 0;
        int z = 0;
        //sorting of an array
        for(int n=0;n<x.length-1;n++) {
            for(int m=0;m<x.length-1;m++) {
                if(x[m]>x[m+1]) {
                    z = x[m];
                    x[m]=x[m+1];
                    x[m+1]=z;
                }
            }
        }
        //Removing Duplicate elements of array
        for(int i=0;i<x.length-1;i++) {
            if(x[i]!=x[i+1]) {
                temp[j] = x[i];
                j++;
            }
        }
        temp[j]=x[x.length-1];
        for(int k=0;k<=j;k++) {
            System.out.println(temp[k]);
        }
    }
}
```

Output:-

2  
8  
12  
14  
24  
30

# File Handling (File class)

```
import java.io.File;
public class A{
    public static void main(String[] args) {
        File f = new File("/Users/syedmujeeb/Desktop/JAVA/FileHandling/A.txt");
        System.out.println(f);
    }
}
```

Output:-

/Users/syedmujeeb/Desktop/JAVA/FileHandling/A.txt

## exists():-

- It is a Non-static method present inside File Class
- It's return type is boolean
- It checks whether the file is present in the given path. If present it will return true or else false.

```
import java.io.File;
public class A {
    public static void main(String[] args) {
        File f = new File("/Users/syedmujeeb/Desktop/JAVA/FileHandling/A.txt");
        boolean val = f.exists();
        System.out.println(val);
    }
}
```

Output:- false

## delete():-

- It is a non-static method present inside File class
- It's return type is boolean
- It will delete the file int given path. If file is deleted it will return true or else false.

```
import java.io.File;
public class A{
    public static void main(String[] args) {
        File f = new File("/Users/syedmujeeb/Desktop/JAVA/FileHandling/A.txt");
        boolean val = f.delete();
        System.out.println(val);
    }
}
```

Output:- false

### length():-

- It is a non-static method present inside File class
- It's return type is long
- It will count the number of characters inside the file including space characters.

```
import java.io.File;
public class A{
    public static void main(String[] args) {
        File f = new File("/Users/syedmujeeb/Desktop/JAVA/FileHandling/A.txt");
        long val = f.length();
        System.out.println(val);
    }
}
```

Output:- 0 //because there no file with name A.txt

### createNewFile():-

- It is a non-static method present inside File class
- It's return type is boolean
- It will create new file if the file does not exists in the given path and returns true. If the file Already exists then it will not replace that file and returns false.

**Note:-** checked/compiletime Exceptions are mandatory to be handled.

```
import java.io.File;
import java.io.IOException;
public class A{
    public static void main(String[] args) {
        try{
            File f = new File("/Users/syedmujeeb/Desktop/JAVA/FileHandling/A.txt");
            boolean val = f.createNewFile();
            System.out.println(val);
        }catch(IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:-  
true

### **mkdir():-**

- It is a non-static method present inside File class
- It's return type is boolean
- It will create new folder if the folder does not exists in the given path and return true. If the folder already exists the it will not replace that folder and retrun false

```
import java.io.File;
public class A{
    public static void main(String[] args) {
        File f = new File("/Users/syedmujeeb/Desktop/JAVA/FileHandling2");
        boolean val = f.mkdir();
        System.out.println(val);
    }
}
```

Output:-  
true

### **list():-**

- It is a non-static method present inside File class
- It's return typeString Array
- It will get all the files and folder names in the given path

```
import java.io.File;
public class A{
    public static void main(String[] args) {
        File f = new File("/Users/syedmujeeb/Desktop/JAVA");
        String[] val = f.list();
        for(int i=0;i<val.length;i++) {
            System.out.println(val[i]);
        }
    }
}
```

Output:-  
Screenshot 2024-06-25 at 1.00.23 PM.png  
.DS\_Store  
pankaj JAVA notes.pages  
FileHandling2  
FileHandling

# FileReader Class

Method 1:-

```
import java.io.File;
import java.io.FileReader;
public class A{
    public static void main(String[] args) {
        try {
            File f = new File("/Users/syedmujeeb/Desktop/JAVA/FileHandling/A.txt");
            FileReader fr = new FileReader(f);
            for(int i=0;i<f.length();i++) {
                System.out.print((char)fr.read());
                //\n is removed to not to print in next line every time
            }
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:-  
abcd

Method 2:-

```
import java.io.File;
import java.io.FileReader;
public class A{
    public static void main(String[] args) {
        try {
            File f = new File("/Users/syedmujeeb/Desktop/JAVA/FileHandling/A.txt");
            FileReader fr = new FileReader(f);
            char ch[] = new char[(int)f.length()];
            fr.read(ch);
            for(char i:ch) {
                System.out.print(i);
            }
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:-  
abcd

# FileWriter Class

```
import java.io.FileWriter;
public class A{
    public static void main(String[] args) {
        try {
            FileWriter fw = new FileWriter("/Users/syedmujeeb/Desktop/JAVA/
FileHandling/B.txt", true); // Here writing true mean append to existing data
            fw.write(97); // this will append letter "a" to the file
            fw.write("Sony"); // this will append String "Sony" to
the file
            fw.write("100"); // this will append 100 as text to the
file
            fw.close(); // This will save the changes in file
[Mandatory to write]
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.io.FileWriter;
public class A{
    public static void main(String[] args) {
        try {
            FileWriter fw = new FileWriter("/Users/syedmujeeb/Desktop/JAVA/
FileHandling/B.txt", true);
            fw.write(100);
            fw.write("\n");
            fw.write("mike");
            fw.write("\n");
            char[] ch= {'a', 'b', 'c'};
            fw.write(ch);
            fw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

# BufferedWriter Class

- Improves File Writing performance
- Has `newLine()` method for writing context in new line.[ Where as `fw.write("\n")` is acts like bug some times]

```
import java.io.FileWriter;
import java.io.BufferedWriter;
public class A{
    public static void main(String[] args) {
        try {
            FileWriter fw = new FileWriter("/Users/syedmujeeb/Desktop/
JAVA/FileHandling/B.txt",true);
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write(100);
            bw.newLine();
            bw.write("mike");
            bw.newLine();
            char[] ch= {'a','b','c'};
            bw.write(ch);
            bw.close(); //BufferedWriter Should close first
            fw.close(); //After closing the file we can't write or read anything
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

# BufferedReader Class

- Improves File Reading performance.
- Has readLine() method in BufferedReader class can read complete line in a file.[Which reads entire line from the file]

```
import java.io.FileReader;
import java.io.BufferedReader;
public class A{
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("/Users/syedmujeeb/Desktop/
JAVA/FileHandling/B.txt");
            BufferedReader br = new BufferedReader(fr);
            System.out.println(br.readLine()); // reads 1st line of the file
            System.out.println(br.readLine()); // reads 2nd line of the file
            System.out.println(br.readLine()); // reads 3rd line of the file
            br.close();
            fr.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

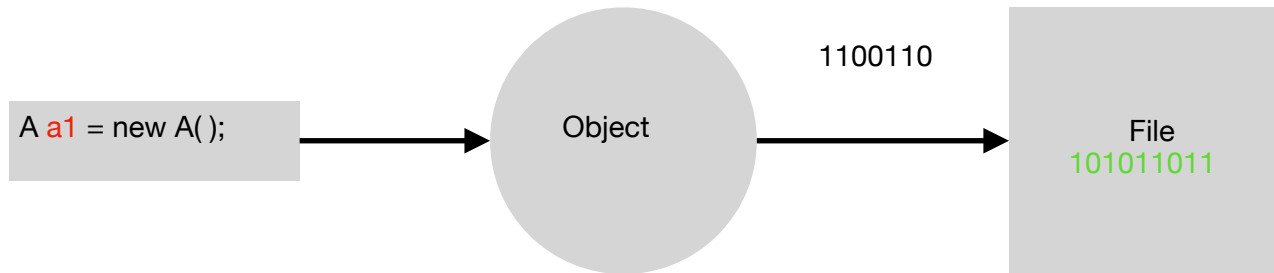
Output:-

```
b
mike
abc
```

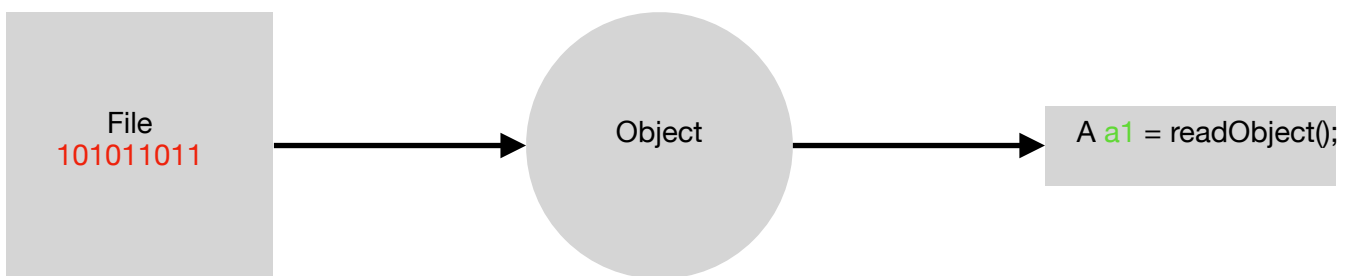


# Serialization & De-Serialization

## Serialization



## De-Serialization



```
package p1;
import java.io.Serializable;
public class A implements Serializable{
    public String name = "Syed";
    public String password = "Mujeeb@123";
}
```

## Serialization

```
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
public class B{
    public static void main(String[] args) {
        try {
            FileOutputStream fos = new FileOutputStream("/Users/syedmujeeb/Desktop/JAVA/FileHandling/D.ser");
            A a1 = new A();
            ObjectOutputStream oos = new
ObjectOutputStream(fos);
            oos.writeObject(a1);
            oos.close();
            fos.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

## De-Serialization

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;
public class C {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream("/Users/
syedmujeeb/Desktop/JAVA/FileHandling/D.ser");
            ObjectInputStream ois = new
ObjectInputStream(fis);
            A a1 = (A)ois.readObject();//return type is Object
            System.out.println(a1.name);
            System.out.println(a1.password);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
Output:-
Syed
Mujeeb@123
```

## Object class:-

- Object Class is a parent class of all classes in java.

## Transient:-

- Transient keyword skip's writing the content of variable to the object during Serialization.

# JDBC(java to database connectivity)

Steps to connect java(eclipse) with mysql database:-

Step1:- We need to download a mysql connector jar.

Step2:- We have to create a folder in java-project in eclipse.

Step3:- Copy jar file and paste in the folder created in java-project.

Step4:- Right click on project and go to properties.

Step5:- then, Click on java Build path.

Step6:- go to libraries tab and Click on Add jar.

Step7:- Select the jar file pasted in the folder in project.

Step8:- Apply and Close.

After these steps are done our **java-project** is ready to connect with **database**.

There are **CRUD** operations in SQL, to perform **Create**, **Update** and **Delete** we can use **executeUpdate( )** method to run the query.

To perform **Retrieval** operations we have to use **executeQuery( )** method.

To **Insert** record into table

```
package p1;
import java.sql.*;
public class A{
    public static void main(String[] args) {
        try {
            //connect to database
            Connection con = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/psadb1","root","9515055232");

            //Write & execute SQL Query
            Statement stmt = con.createStatement();
            stmt.executeUpdate
                ("insert into registration values('syed','syed@gmail.com','9515055232')");

            //Close connection
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

To **Delete** record from table

```
package p1;
import java.sql.*;
public class A{
    public static void main(String[] args) {
        try {
            //connect to database
            Connection con = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/psadb1","root","9515055232");
            //write & execute SQL Query
            Statement stmt = con.createStatement();
            stmt.executeUpdate
                ("Delete from registration where name = 'mike'");
            //Close connection
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

To **Update** record of table

```
package p1;
import java.sql.*;
public class A{
    public static void main(String[] args) {
        try {
            //connect to database
            Connection con = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/psadb1","root","9515055232");
            //write & execute SQL Query
            Statement stmt = con.createStatement();
            stmt.executeUpdate
                ("update registration set phone = '8179302796' where name = 'syed'");
            //Close connection
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

To **Retrieve** data of table

```
import java.sql.*;
public class B {
    public static void main(String[] args) {
        try {
            //connect to database
            Connection con = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/psadb1","root","9515055232");

            //write & execute sql Query
            Statement stmt = con.createStatement();
            ResultSet result =
                stmt.executeQuery("select * from psadb1.registration");

            while(result.next()) {
                System.out.println(result.getString(1));
                System.out.println(result.getString(2));
                System.out.println(result.getString(3));
            }

            //close connection
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

## Take values form user and Insert into database

Taking input form user their name, email, mobile number and adding their information to the database using jdbc.

```
import java.sql.*;
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        try {
            Scanner scan = new Scanner(System.in);
            System.out.println("Enter your name : ");
            String name = scan.next();
            System.out.println("Enter your email : ");
            String email = scan.next();
            System.out.println("Enter your mobile number : ");
            String phone = scan.next();
            //connect to database
            Connection con = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/psadb1","root","9515055232");
            //write & execute SQL Query
            Statement stmt = con.createStatement();
            stmt.executeUpdate
                ("insert into registration values('"+name+"','"+email+"','"+phone+"')");
            //Close connection
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:-

```
Enter your name : shaik
Enter your email : shaik@gmail.com
Enter your mobile number : 8341716022
```

**Taking name as input and deleting their data form Database related to that person**

```
import java.sql.*;
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        try {
            Scanner scan = new Scanner(System.in);
            System.out.print("Enter name : ");
            String name = scan.next();

            //connect to database
            Connection con = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/psadb1","root","9515055232");
            //write & execute SQL Query
            Statement stmt = con.createStatement();
            stmt.executeUpdate
                ("delete from registration where name = '"+name+"'");
            //Close connection
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

## Taking name and new mobile number of user and updating in database

```
import java.sql.*;
import java.util.Scanner;
public class A{
    public static void main(String[] args) {
        try {
            Scanner scan = new Scanner(System.in);
            System.out.print("Enter name : ");
            String name = scan.next();
            System.out.print("Enter new mobile number : ");
            String new_phone = scan.next();

            //connect to database
            Connection con = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/psadb1","root","9515055232");
            //write & execute SQL Query
            Statement stmt = con.createStatement();
            stmt.executeUpdate
                ("update registration set phone = '"+new_phone+"' where name = '"+name+"'");
            //Close connection
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



# Finally block

- Regardless of exception finally block will run 100%.

```
public class A {  
    public static void main(String[] args) {  
        try {  
            int x = 10/0; //Exception  
        } catch (Exception e) {  
            e.printStackTrace();  
        } finally {  
            System.out.println("Finally...");  
        }  
    }  
}
```

Output:-

```
java.lang.ArithmeticException: / by zero  
Finally...  
at p1.A.main(A.java:5)
```

```
public class A {  
    public static void main(String[] args) {  
        try {  
            int x = 10/2; //No exception  
        } catch (Exception e) {  
            e.printStackTrace();  
        } finally {  
            System.out.println("Finally...");  
        }  
    }  
}
```

}

Output:-

Finally...

```
public class A {  
    public static void main(String[] args) {  
        try {  
            int x = 10/0; //Exception  
        } finally { // Without catch block  
            System.out.println("Finally...");  
        }  
    }  
}
```

}

Output:-

Finally...

```
Exception in thread "main" java.lang.ArithmeticException: / by  
zero  
at p1.A.main(A.java:5)
```

# Realtime Example of Finally block

- Closing the Database connection
- Closing File, etc.

```
import java.sql.*;
public class A {
    public static void main(String[] args) {
        //we are creating Connection con reference variable outside the try block,
        //So that it can also be used in finally block.
        //If we create con variable inside try block then it will be local variable
        //of try block
        Connection con = null;
        try {
            con = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/psadb1","root","9515055232");

            Statement stmt = con.createStatement();
            stmt.executeUpdate
                ("insert into registration values('syed','syed@gmail.com','9515055232')");

        }catch(Exception e) {
            e.printStackTrace();
        }finally {
            //we are closing connection in finally block because at
            //the end the connection should be closed at any cost.
            try {
                con.close();
                // Here, con.close is database code so we are
                // writing inside try catch block
            }catch(Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

# Polymorphism

- Here we develop a future such that it can take more than one form depending on the situation.

**Note:-** Polymorphism can be applicable only on methods.

Polymorphism is of two types,

## Overriding[Runtime Polymorphism]

Overriding is done by following these 3 rules:-

1. Classes Should be inherited and in child class we can make Overriding.
2. During Overriding if we increase the scope of visibility then Overriding takes place.
3. During Overriding method return type should match.

**Note:-** We cannot Override Private methods because they will not be inherited

**Rule 1:-**

```
package p1;
public class Dog{
    public void eat() {
        System.out.println("Eating...");
    }
    public void noice() {
        System.out.println("bow bow...");
    }
}
package p1;

public class Cat extends Dog {

    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();
        d.noice();
        Cat c = new Cat();
        c.eat();
        c.noice();
    }

    @Override
    public void noice() {
        System.out.println("meow meow...");
    }
}
```

Output:-

```
Eating...
bow bow...
Eating...
meow meow...
```

## Rule 2:-Method overriding with different access specifiers

```
public class A{
    void test() {
        System.out.println(100);
    }
}
public class B extends A{
    @Override
    public void test() { //Increase Scope of visibility from Default to public
        System.out.println(200);
    }
}
//Implemented Overriding Successfully
```

```
public class A{
    public void test() {
        System.out.println(100);
    }
}
public class B extends A{
    @Override
    void test() { //Error
        System.out.println(200);
    }
}
//Overriding dose not happened because here Scope of visibility
is decreasing form public to Default
```

```
public class A{
    protected void test() {
        System.out.println(100);
    }
}
public class B extends A{
    @Override
    public void test() { //Increasing scope of visibility from Protected to public
        System.out.println(200);
    }
}
//Implements Overriding Successfully
```

# Overloading[compiletime Polymorphism]

- Here we create more than one method with the same name in same class provided they have different number of arguments and different type of arguments.

```
public class Calculator{
    public int sum(int a,int b) {
        return a+b;
    }
    public float sum(float a,float b) {
        return a+b;
    }
    public static void main(String[] args) {
        Calculator c = new Calculator();
        int result1 = c.sum(10, 20);
        float result2 = c.sum(10.5f , 15.7f);
        System.out.println(result1);
        System.out.println(result2);
    }
}
```

Output:-

30  
26.2

# Interface

- An interface can consist of only incomplete methods init.
- Interface puts contract on class so that the class mandatorily has to implement those inherited methods of interface or else you will get an error in the class

```
//interface A
public interface A {
    public void test();
}
//class B inheriting A
public class B implements A {
    @Override
    public void test() {
        System.out.println(100);
    }
    public static void main(String[] args) {
        B b1 = new B();
        b1.test();
    }
}
```

Output:-  
100

- We cannot create private and protected incomplete methods in an interface.
- We can only create public and default incomplete methods in an interface.

```
public interface A {
    public void test1(); //No Error it will work
    void test2(); //No Error it will work
    protected void test3(); //Error
    private void test4(); //Error
}
```

**Note:-** Class and interface can only be **public** or **Default**, but not **protected** and **private**.

## Possibilities of inheritance and how:

Class	to	Class	—> extends
Interface	to	interface	—> extends
Interface	to	Class	—> implements

- We have to implement all the methods present in interface otherwise we will get an error in class

```
//Interface A
public interface A {
    public void test1();
    public void test2();
}
//Class B inheriting A
public class B implements A {
    @Override
    public void test1() {
        System.out.println(100);
    }
    @Override
    public void test2() {
        System.out.println(200);
    }
    public static void main(String[] args) {
        B b1 = new B();
        b1.test1();
        b1.test2();
    }
}
```

Output:-

```
100
200
```

```
//Interface A
public interface A {
    public void test1();
}
//interface B inheriting A
public interface B extends A {
    public void test2();
}
// Class C inheriting B
public class C implements B{
    @Override
    public void test1() {
        System.out.println(100);
    }
    @Override
    public void test2() {
        System.out.println(200);
    }
    public static void main(String[] args) {
        C b1 = new C();
        b1.test1();
        b1.test2();
    }
}
```

Output:-

```
100
200
```

# Multiple inheritance of interface

Class does not support multiple inheritance when we are inheriting classes. But, we can do multiple inheritance of interface to the class like shown in below example.

```
//Interface A
public interface A {
    public void test1();
}
//Interface B
public interface B {
    public void test2();
}
// Class C inheriting B
public interface C extends A,B{
}
public class D implements C{
    @Override
    public void test2() {
        System.out.println(200);
    }
    @Override
    public void test1() {
        System.out.println(100);
    }
    public static void main(String[] args) {
        D d1 = new D();
        d1.test1();
        d1.test2();
    }
}
```

Output:-

100  
200



```

public interface A {
    public void test1();
}
public interface B {
    public void test1();
}
public class C implements A,B{
    @Override
    public void test1() {
        System.out.println(100);
    }
    public static void main(String[] args) {
        C d1 = new C();
        d1.test1();
    }
}

```

Sequence is important while we are inheriting both class and inheritance.

**Order:-** extends → implements

Only one class can be inherited as class does not support multiple inheritance.

Multiple number of interface can be inherited as interface allows multiple inheritance.

```

public class A {
}
public interface B {
}
public interface C {
}
public class D extends A implements B,C{
}

```

## Marker Interface

- An empty interface is called as marker interface.

# Final keyword in java

- If you make a variable final it's value cannot be changed.

```
public class A {  
    public static void main(String[] args) {  
        final int x = 10;  
        x = 100; //Error  
        x = 10; //Error  
    }  
}
```

- If you make static/non-static variable final then it is mandatory to initialize or else you will get an error (The blank field x may not have been initialize).

```
public class A {  
    final int x; //Error  
    final static int x; //Error  
}
```

- If you make method final then Overriding is not allowed.

```
public class A {  
    final public void test() {  
    }  
}  
public class B extends A {  
    @Override  
    public void test() { //Error  
    }  
}
```

- If you make class final then inheritance is not allowed.

```
final public class A {  
}  
public class B extends A { //Error  
}
```

# Variables in interface

- All variables by default in an interface is final and static.
- We cannot create object of interface.
- An object of interface cannot be created but reference variable can be created.

```
public interface A {  
    final static int CURRENT_SCORE = 1;  
    String USER_NAME = "mike";  
}  
public class B {  
    public static void main(String[] args) {  
        System.out.println(A.CURRENT_SCORE);  
        System.out.println(A.USER_NAME);  
        A a1 = new A(); //ERROR  
        A a2; //ERROR  
    }  
}
```

# Features of java version 8

1. Optional class
2. Default keyword
3. Lambda Expression
4. Functional interface
5. Stream API

## Optional class

- This was introduced in java 8.
- It is an alternative way of handling NullPointerException.

```
import java.util.Optional;
public class A {
    int x = 10;
    public static void main(String[] args) {
        A a1 = new A();
        Optional<A> val = Optional.ofNullable(a1);
        if(val.isPresent()) {
            System.out.println(a1.x);
        }else {
            System.out.println("Null Pointer");
        }
    }
}
```

Output:-

Null Pointer

## Default keyword

- This was introduced in java 8.
- Using default keyword we can create complete methods in an interface.

```
public interface A{
    default public void test1() {
        System.out.println(100);
    }
    default public void test2() {
        System.out.println(200);
    }
}
public class B implements A {
    public static void main(String[] args) {
        B b1 = new B();
        b1.test1();
        b1.test2();
    }
}
```

# Functional Interface

- This was introduced in java 8.
- This interface can consist of exactly one incomplete method init.
- But we can built number of complete methods in Functional interface.

```
@FunctionalInterface
public interface A{
    public void test1();
    default public void test2() {
        System.out.println(200);
    }
    default public void test3() {
        System.out.println(300);
    }
}

public class B implements A {
    @Override
    public void test1() {
        System.out.println(100);
    }
    public static void main(String[] args) {
        B b1 = new B();
        b1.test1();
        b1.test2();
        b1.test3();
    }
}
```

Output:-

```
100
200
300
```

# Lambda Expression

- This was introduced in java 8.
- Reduces number of lines of code.

```
@FunctionalInterface
public interface A {
    public void test1(int x);
    default public void test2() {
        System.out.println(200);
    }
}

public class B implements A {
    public static void main(String[] args) {
        A a1 = (int x)->{
            System.out.println(x);
        };
        a1.test1(100);
        a1.test2();
    }
}
```

**Note1:-** we can create main method in interface, This was introduced in java 8.

**Note2:-** we can create static complete method in interface without using default keyword.

**Note3:-** we can not create incomplete static method.

# Abstract class

Abstraction means hiding of implementation details. To do this in java we use interface and abstract classes.

- It can consist of both complete and incomplete methods in it.
- Objects of abstract class cannot be created. But we can create reference variable.
- To create incomplete methods it is mandatory to use **abstract keyword**.

```
abstract public class A {
    int x = 10;
    static int y = 20;
    public void test1() {
        System.out.println(100);
    }
    abstract public void test2();
}
public class B extends A{
    public static void main(String[] args) {
        B b1 = new B();
        b1.test1();
        b1.test2();
        System.out.println(b1.x);
        System.out.println(A.y);
    }
    @Override
    public void test2() {
        System.out.println(200);
    }
}
```

Output:-

```
100
200
10
20
```

- We can create static and non static members both in abstract class.
- Abstract class does not support multiple inheritance.

```
abstract public class A{
}
abstract public class B{
}
public class C extends A,B{ //Error
} //Multiple inheritance of abstract class is not allowed
```

```
public interface A{
    public void test1();
}

abstract public class B{
    abstract public void test2();
}

public class C extends B implements A {

    @Override
    public void test1() {
        System.out.println(100);
    }

    @Override
    public void test2() {
        System.out.println(200);
    }

}
```



# Encapsulation

Encapsulation refers to the bundling of data with the methods that operate on the data.

To achieve Encapsulation:-

1. Make variables private to avoid direct access
2. Define setters and getters with public access

```
public class A{
    private String password;

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}

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

public class B extends A{
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("/Users/syedmujeeb/
Desktop/JAVA/FileHandling/A.txt");
            BufferedReader br = new BufferedReader(fr);
            A a1 = new A();
            a1.setPassword(br.readLine());
            System.out.println(a1.getPassword());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

## How to create tomcat server in eclipse

1. Press **ctrl+n**
2. Select **server** click on **next**
3. Select **tomcat version** from **Apache folder** and click on **next**
4. Then click on **browse** and select the **path** of the downloaded tomcat file
5. Click on **finish**

## How to delete tomcat server in eclipse

1. Right click on **server folder** in **Package Explorer**
2. Click on **delete** and **check** the box to delete from disk
3. Press **ctrl+n**, select **server** click on next
4. Select **tomcat version** from **Apache folder** and click on **{Configure runtime environments...}**
5. Select **Apache tomcat runtime environment** and click on **remove**
6. Click on **apply and close**, click on **cancel**

### **Java Demo Session 1 Recordings:**

TOPICS COVERED : - [ **class, new, garbage collector intro**]

Link: [https://us02web.zoom.us/rec/share/JuhLP\\_O2ZF5ejjzxcPp\\_3hFgYwglCu-UhVhg7fMPzLAI-ltbyy\\_1jL6fx0GljeCg.hpHB34GJyOXy9uAX](https://us02web.zoom.us/rec/share/JuhLP_O2ZF5ejjzxcPp_3hFgYwglCu-UhVhg7fMPzLAI-ltbyy_1jL6fx0GljeCg.hpHB34GJyOXy9uAX)

Passcode: Pankaj123\$

### **Java Demo Session 2 Recordings:**

TOPICS COVERED : - [ **Static, non Static, Methods Introduction**]

Link: [https://us02web.zoom.us/rec/share/UhpXfAQiwcZCECrMC9o-owv2xlqu8FucbZOBPyH9M8sjGtccTW5nEVMynJm5cBI.Ax\\_9utAp8RDXfAFk](https://us02web.zoom.us/rec/share/UhpXfAQiwcZCECrMC9o-owv2xlqu8FucbZOBPyH9M8sjGtccTW5nEVMynJm5cBI.Ax_9utAp8RDXfAFk)

Passcode: Pankaj123\$

### **Java Demo Session 3 Recordings:**

TOPICS COVERED : - [ **Memory Management** ]

Link: [https://us02web.zoom.us/rec/share/xxTrZjl-1nH\\_FXkxyR4t6-rOX4jsMNoZ6BfL7oZX5LQoHrOguNYkcZK81-z27sjq.cizlJrn701egmKht](https://us02web.zoom.us/rec/share/xxTrZjl-1nH_FXkxyR4t6-rOX4jsMNoZ6BfL7oZX5LQoHrOguNYkcZK81-z27sjq.cizlJrn701egmKht)

Passcode: Pankaj123\$