

JAVA

- To execute we need to set the right path of the programs, changing the directory:

```
C:\Users\ganieswar>cd desktop
```

```
C:\Users\ganieswar\Desktop>cd java_programs
```

- In java, System and String need to have uppercase 'S'.

```
/*class GlobalVariable{
int a =10;
public static void main(String []args){
int b=20;
System.out.println(b);
System.out.println(a);
}
}
error: non-static variable a cannot be referenced from a static context
System.out.println(a);
                ^
1 error
*/
class GlobalVariable{
static int a =10;
public static void main(String []args){
int b=20;
System.out.println(b);
System.out.println(a);
}
}

//output: C:\Users\ganieswar\Desktop\java_programs>java GlobalVariable.java
//20
//10
```

- We got error in the first case, because a is not a static content which stores in a heap area and the main method is static content which is in stack area, so we can't access it.

```

/*class SameVariable{
static int a =10;
public static void main(String []args){
int a=20;
System.out.println(a);
}
}
output: C:\Users\ganieswar\Desktop\java_programs>java SameVariable.java
20
*/
class SameVariable{
static int a =10;
public static void main(String []args){
int a=20;
System.out.println(a);
System.out.println(SameVariable.a);
}
}
//C:\Users\ganieswar\Desktop\java_programs>java SameVariable.java
//20
//10

```

- Local Variables have high priority than Global Variable.
- We can't declare a local variable as static if so, we will get a compile time error of illegal start of expression.
- We can access the static variable by using class name.
- We can't access the static variable with this keyword and also, we can't use this keyword from the static method.

```

class LocalVariable{
static int a =10;
static int b =20;
public static void main(String []args){
System.out.println(a);
a=20;
System.out.println(b);
int a=30;
System.out.println(LocalVariable.a);
LocalVariable.a =40;
System.out.println(a);
b=50;
System.out.println(b);
}
}
//\Users\ganieswar\Desktop\java_programs>java LocalVariable.java
//10
//20
//20
//30
//50

```

- Here we can see that before declaring a local variable with same name as global variable we access the global variable directly without the class name. Once it is declared we will access the local variable if we don't mention the class name.
- Local Variable can't be accessed outside the block.

```
class Concatination{
public static void main(String []args){
System.out.println(10+20+'a'+"hello");
System.out.println("hello"+"bye");
System.out.println('a'+"hello"+10+20);
System.out.println("hello"+10+'a'+"bye");
}
}
//output:C:\Users\ganieswar\Desktop\java_programs>java Concatination.java
//127hello
//hellobye
//ahello1020
//hello10abye
```

- String + Any Datatype = String
- In first case integer + char value gives 127.

```
class LastDigit{
public static void main(String []args){
int a = 12345;
System.out.println(a%10);
}
}
//output:5|
```

- % gives the remainder and / gives the quotient.

```
import java.util.Scanner;
class ScannerClass{
public static void main(String []args){
Scanner sc = new Scanner(System.in);
System.out.println("enter your name");
String name = sc.next();
System.out.println("enter your age:");
int age= sc.nextInt();
System.out.println("Your name is "+name+" and age is "+age);
}
}
//C:\Users\ganieswar\Desktop\java_programs>java Scanner.java
//enter your name
//ganesh
//enter your age:
//21
//Your name is ganesh and age is 21
```

- next () is used to read a single word and nextLine () is used to read a sentence.
- System.out.println() gives an output of an empty line, we can also use System.out.print() to get the output but it doesn't give a new line after the result.

```
import java.util.Scanner;
class SwitchCase{
public static void main(String []args){
Scanner sc = new Scanner(System.in);
System.out.println("enter a character");
char ch = sc.next().charAt(0);
switch(ch){
case 'a':
case 'e':
case 'i':
case 'o':
case 'u':System.out.println("vowel");
break;
default:System.out.println("consonent");
}
}
}
//output:C:\Users\ganieswar\Desktop\java_programs>java SwitchCase.java
//enter a character
//i
//vowel
|
//C:\Users\ganieswar\Desktop\java_programs>java SwitchCase.java
//enter a character
//z
//consonent
```

■

```
class ForLoop{
public static void main(String []args){
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
System.out.print("* ");
}
System.out.println();
}
}
}
//C:\Users\ganieswar\Desktop\java_programs>java ForLoop.java
//* * *
//* * *
//* * *
```

```
class ForLoop{
public static void main(String []args){
int a=1;
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
System.out.print(a+" ");
a++;
}
System.out.println();
}
}
}|

//C:\Users\ganieswar\Desktop\java_programs>java ForLoop.java
//1 2 3
//4 5 6
//7 8 9
```

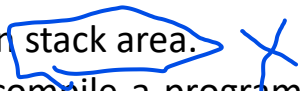
■ LEAP YEAR PROGRAM

■ METHODS

```
/*class Methods1{
public void display(){
System.out.println("this is display method");
}
public static void main(String []args){
System.out.println("this is main method");
display();
}
}
Methods1.java:7: error: non-static method display() cannot be referenced from a static context
display();
^
1 error*/

class Methods1{
public static void display(){
System.out.println("this is display method");
}
public static void main(String []args){
System.out.println("this is main method");
display();
}
}

//C:\Users\ganieswar\Desktop\java_programs>java Methods1.java
//this is main method
//this is display method
```

- We can access the method directly if it is in the same area like in the above case we can access because both are static so they are in stack area.  method area ✓
- Execution starts with the main method. We can compile a program without a main method but we can't run a program without a main method.

```

import java.util.Scanner;
class Length{
public static int length(int a){
int count=0;
while(a!=0){
a=a/10;
count++;
}
return count;
}
public static int SumOfDigits(int a){
int rem=0;
while(a!=0){
rem+=a%10;
a=a/10;
}
return rem;
}

public static void main(String []args){
Scanner sc = new Scanner(System.in);
System.out.print("Enter a number:");
int number = sc.nextInt();
System.out.println("the length of given number is "+length(number));
System.out.println("the sum of digits in given number is "+SumOfDigits(number));
}
}
//C:\Users\ganieswar\Desktop\java_programs>java Length.java
//Enter a number:12345
//the length of given number is 5
//the sum of digits in given number is 15

```

■ Some predefined methods

```

class Predefined{
public static void main(String []args){
System.out.println(Math.PI);
System.out.println(Math.abs(-23));
System.out.println(Math.round(99.67));
System.out.println(Math.min(2,3));
System.out.println(Math.pow(2,3));
}
}
//C:\Users\ganieswar\Desktop\java_programs>java Predefined.java
//3.141592653589793
//23
//100
//2
//8.0

```

■ Static_INITIALIZER

```
class Initializer{
static{
System.out.println("static initializer1 ");
}
public static void main(String []args){
System.out.println("main method");
}
static{
System.out.println("static initializer2 ");
}}
//C:\Users\ganieswar\Desktop\java_programs>java Initializer.java
//static initializer1
//static initializer2
//main method
```

- Execution always starts from main method but Static_INITIALIZER executes first.
- Initializers will start from top to bottom.
- Non-Static Members (non-static variables, non-static methods, non-static initializers)
- We can access a non-static member only by creating an object to the class. With the help of that object-reference variable we can access them.

```
class NonStatic{
int a=10;
public void test(){
System.out.println("test method");
}
public static void main(String []args){
NonStatic ns = new NonStatic();
System.out.println(ns.a);
ns.test();
}
}
//C:\Users\ganieswar\Desktop\java_programs>java NonStatic.java
//10
//test method
```


- Here a and test () are in heap area and whereas main method is in stack area. So, we use object to connect them.

```
class Sample{
    static int a=10;
    int b=20;
    public static void test(){
        System.out.println("static test method");
    }
    public void display(){
        System.out.println("non-static display method");
    }

    static{
        System.out.println("static initializer1 ");
    }
    public static void main(String []args){
        System.out.println("main method");
        test();
        Sample s= new Sample();
        s.display();
        System.out.println(a);
        System.out.println(s.b);
    }
    static{
        System.out.println("static initializer2 ");
    }
    {
        System.out.println("non-static Initializer");
    }
}
//C:\Users\ganieswar\Desktop\java_programs>java Sample.java
//static initializer1
//static initializer2
//main method
//static test method
//non-static Initializer
//non-static display method
//10
//20
```

- Non-Static Initializer will execute only after the object gets created.
- Static members will load when class gets loaded, whereas non-static members will load once the object gets created.
- Static members will be loaded when class is loaded, for static members only one memory get created and only once.

- Non-static members will load when object gets created, for every new keyword new object will be created and those objects load all the non-static members.
- Static members can also be accessed by class name as well as object reference variable but non- static members can be accessed only by using object reference variable.

```
class Demo1{
static int a=10;
int b=20;
public static void main(String []args){
int c=30;//local variable
Demo1 s= new Demo1();
System.out.println(a);
System.out.println(s.a);
System.out.println(Demo1.a);
System.out.println(s.b);
//System.out.println(s.c);
System.out.println(c);
}}
//C:\Users\ganieswar\Desktop\java_programs>java Demo1.java
//10
//10
//10
//20
//30
```

- Here we can see that we can't able to access the local variable using reference object instead we can directly access it because it is in the same block and also it is in static main method so it is also considered to be static.
- In non-static method we can access all the static and non-static members directly. Because to access that non-static method also we need to create an object so indirectly we are using object to access the members in non-static class.

```
class Demo3{
static int a=10;
int b=20;
public void display(){
System.out.println(a);
System.out.println(b);
}
public static void main(String []args){
Demo3 d=new Demo3();
d.display();
}}
//C:\Users\ganieswar\Desktop\java_programs>java Demo3.java
//10
//20
```

■ CONSTRUCTORS (To-Do Initialization)

- It should have same name as class name.
- There should be no return type.
- **It is a special type of method, which is mainly used for initializing the non-static variable and it will execute only when object get created.**
- When an object gets created, non-static initializer will execute before the constructor got executed.

```
class Demo2{
{
System.out.println("non-static initializer");
}
Demo2(){
System.out.println("constructor");

}
public static void main(String []args){
System.out.println("main method");
Demo2 d=new Demo2();
}}
//C:\Users\ganieswar\Desktop\java_programs>java Demo2.java
//main method
//non-static initializer
//constructor
```

- We use constructors to do initialization like:

```
class Constructor{
private int a;
String b;
Constructor(int a, String b){
this.a=a;
this.b=b;
}
public void display(){
System.out.println(a+" "+b);
}
public static void main(String []args){
Constructor c =new Constructor(5, "ganesh");
c.display();
}
}
```

- Here we can also use scanner class and variables to take input from the user.

```

import java.util.Scanner;
class PersonDetails{
String name;
int age;
long phone_number;
long aadhar_number;
PersonDetails(String name, int age,long phone_number,long aadhar_number){
this.name = name;
if(age>=18){
this.age=age;
}else{
System.out.println("not allowed");
}
if(length(phone_number)==10){
this.phone_number = phone_number;
}
else{
System.out.println("not valid");
}
if(length(aadhar_number)==12){
this.aadhar_number = aadhar_number;
}
else{
System.out.println("not valid");
}
}
public int length(long num){
int count=0;
while(num!=0){
num = num/10;
count++;
}
return count;
}
public void display(){
System.out.println("*** (Person Details*** ");
System.out.println("name: "+name);
System.out.println("age: "+age);
System.out.println("phone_number: "+phone_number);
System.out.println("aadhar_number: "+aadhar_number);
}
public static void main(String []args){
String name;
int age;
long an,pn;
Scanner sc =new Scanner(System.in);
System.out.print("enter the name:");
name = sc.next();
System.out.print("enter the age:");
age =sc.nextInt();
System.out.print("enter phone number:");
pn = sc.nextLong();
System.out.print("enter the aadhar:");
an = sc.nextLong();
PersonDetails pd=new PersonDetails(name,age,pn,an);
pd.display();
}
}

```

```
//C:\Users\ganieswar\Desktop\java_programs>java PersonDetails.java
//enter the name:ganesh
//enter the age:19
//enter phone number:9949201457
//enter the aadhar:823530044980
//*** (Person Details) ***
//name: ganesh
//age: 19
//phone_number: 9949201457
//aadhar_number: 823530044980
```

```
C:\Users\ganieswar\Desktop\java_programs>java PersonDetails.java
enter the name:ganesh
enter the age:16
enter phone number:9949201457
enter the aadhar:123456789012
not allowed
*** (Person Details) ***
name: ganesh
age: 0
phone_number: 9949201457
aadhar_number: 123456789012
```

■ Constructor Overloading

- We can create more constructors with different number of parameters in a program.
- We can also call the constructor from another class; we can see both programs below using the same class and calling from different class.
- While calling the different constructors, we create an object for that but here we need to create an object for each type of constructor and the names of the object should be different.
- Calling from the same class:

Output for the below program:

```
C:\Users\ganieswar\Desktop\java_programs>java ConstructorOverloading.java
enter the name:ganesh
enter the age:20
enter phone number:9949201457
no argument constructor
*** (Person Details) ***
name: null
age: 0
phone_number: 0
string argument constructor
*** (Person Details) ***
name: ganesh
age: 0
phone_number: 0
string and age argument constructor
*** (Person Details) ***
name: ganesh
age: 20
phone_number: 0
String, age and phone number argument constructor
*** (Person Details) ***
name: ganesh
age: 20
phone_number: 9949201457
```

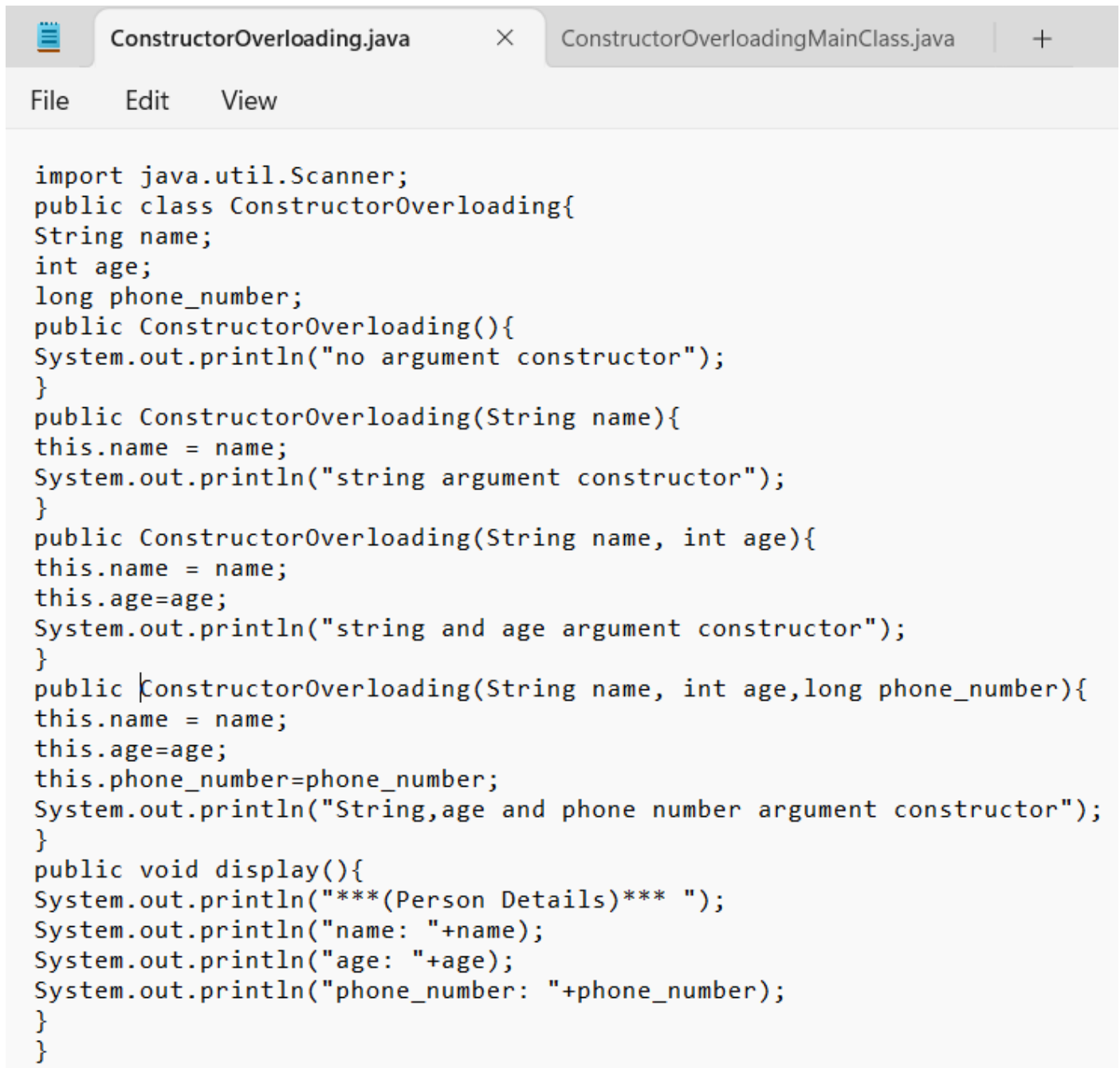
```

import java.util.Scanner;
class ConstructorOverloading{
String name;
int age;
long phone_number;
ConstructorOverloading(){
System.out.println("no argument constructor");
}
ConstructorOverloading(String name){
this.name = name;
System.out.println("string argument constructor");
}
ConstructorOverloading(String name, int age){
this.name = name;
this.age=age;
System.out.println("string and age argument constructor");
}
ConstructorOverloading(String name, int age,long phone_number){
this.name = name;
this.age=age;
this.phone_number=phone_number;
System.out.println("String,age and phone number argument constructor");
}
public void display(){
System.out.println("****(Person Details)*** ");
System.out.println("name: "+name);
System.out.println("age: "+age);
System.out.println("phone_number: "+phone_number);
}
public static void main(String []args){
String name;
int age;
long pn;
Scanner sc =new Scanner(System.in);
System.out.print("enter the name:");
name = sc.next();
System.out.print("enter the age:");
age =sc.nextInt();
System.out.print("enter phone number:");
pn = sc.nextLong();
ConstructorOverloading pd=new ConstructorOverloading();
pd.display();
ConstructorOverloading pd1=new ConstructorOverloading(name);
pd1.display();
ConstructorOverloading pd2=new ConstructorOverloading(name,age);
pd2.display();
ConstructorOverloading pd3=new ConstructorOverloading(name,age,pn);
pd3.display();

}
}

```

■ Calling from different class:



```
import java.util.Scanner;
public class ConstructorOverloading{
String name;
int age;
long phone_number;
public ConstructorOverloading(){
System.out.println("no argument constructor");
}
public ConstructorOverloading(String name){
this.name = name;
System.out.println("string argument constructor");
}
public ConstructorOverloading(String name, int age){
this.name = name;
this.age=age;
System.out.println("string and age argument constructor");
}
public ConstructorOverloading(String name, int age, long phone_number){
this.name = name;
this.age=age;
this.phone_number=phone_number;
System.out.println("String,age and phone number argument constructor");
}
public void display(){
System.out.println("*** (Person Details) *** ");
System.out.println("name: "+name);
System.out.println("age: "+age);
System.out.println("phone_number: "+phone_number);
}
}
```

- If you observe clearly, here we can find that we make the class and all the methods and constructors public, if not so we cannot access them from outside.
- If we don't make them public, We will get an error as:

Exception in thread "main" java.lang.IllegalAccessError: failed to access class ConstructorOverloading from class ConstructorOverloadingMainClass (ConstructorOverloading is in unnamed module of loader 'app'; ConstructorOverloadingMainClass is in unnamed module of loader com.sun.tools.javac.launcher.Main\$MemoryClassLoader @5e25a92e) at ConstructorOverloadingMainClass.main(ConstructorOverloadingMainClass.java:14).



File Edit View

```
import java.util.Scanner;
public class ConstructorOverloadingMainClass{
public static void main(String []args){
String name;
int age;
long pn;
Scanner sc =new Scanner(System.in);
System.out.print("enter the name:");
name = sc.next();
System.out.print("enter the age:");
age =sc.nextInt();
System.out.print("enter phone number:");
pn = sc.nextLong();
ConstructorOverloading pd=new ConstructorOverloading();
pd.display();
ConstructorOverloading pd1=new ConstructorOverloading(name);
pd1.display();
ConstructorOverloading pd2=new ConstructorOverloading(name,age);
pd2.display();
ConstructorOverloading pd3=new ConstructorOverloading(name,age,pn);
pd3.display();

}
}
```

```
C:\Users\ganieswar\Desktop\java_programs>javac ConstructorOverloading.java
```

```
C:\Users\ganieswar\Desktop\java_programs>javac ConstructorOverloadingMainClass.java
```

```
C:\Users\ganieswar\Desktop\java_programs>java ConstructorOverloadingMainClass.java
```

```
enter the name:ganesh
```

```
enter the age:20
```

```
enter phone number:9949201457
```

```
no argument constructor
```

```
*** (Person Details) ***
```

```
name: null
```

```
age: 0
```

```
phone_number: 0
```

```
string argument constructor
```

```
*** (Person Details) ***
```

```
name: ganesh
```

```
age: 0
```

```
phone_number: 0
```

```
string and age argument constructor
```

```
*** (Person Details) ***
```

```
name: ganesh
```

```
age: 20
```

```
phone_number: 0
```

```
String, age and phone number argument constructor
```

```
*** (Person Details) ***
```

```
name: ganesh
```

```
age: 20
```

```
phone_number: 9949201457
```


■ Final Keyword:

```
class Final{
final int a=10;
public static void main(String []args){
Final f = new Final();
System.out.println(f.a);
f.a=20;
System.out.println(f.a);
}
}
```

■ We cannot re-initialize the final member. So we will get a error like:

```
C:\Users\ganieswar\Desktop\java_programs>javac Final.java
Final.java:6: error: cannot assign a value to final variable a
f.a=20;
  ^
1 error
```

```
class Final{
final int a=10;
public static void main(String []args){
Final f = new Final();
System.out.println(f.a);
}
}

//output:10|
```

■ We can also initialize a blank final variable; we can access that variable using constructor only:

```
class BlankFinal{
final int a;
BlankFinal(int a){
this.a=a;
}
public static void main(String []args){
BlankFinal f = new BlankFinal(20);
System.out.println(f.a);
}
}|
//output:20
```

■ Here we can only re-initialize the non-static final variable, if use final static int a; We will get an error.

■ Final Method:

```
class FinalMethod{
    final void display(){
        System.out.println("final method");
    }
    public static void main(String []args){
        FinalMethod f = new FinalMethod();
        f.display();}
    }
    //final method|
```

- We cannot override a method because we cannot modify the method implementation.
- Final Class:
- Similarly, final public class class_name[], we cannot inherit a class which is a final class.
- Nested Class:
- We cannot **declare** a method inside a method and cannot **declare** a constructor inside a constructor but we can declare a class inside a class.

```
public class NestedClass{
    int a=10;
    class Demo2{
        int b=20;
    }
    public static void main(String []args){
        NestedClass n=new NestedClass();
        NestedClass.Demo2 d = n.new Demo2();
        System.out.println(n.a);
        System.out.println(d.b);
    }
}
//10
//20
```

- We can have a special type of object creation for inner class.
- We can declare inner class as private, default, protected, public, static, final, abstract.
- We can declare outer class as public, default, final, abstract only.