12.30PM 1st August 2020 Notes

What is a class keyword?

Note: Java is cases sensitive. All keywords in java are written in lower case.

Class is a factory the generates objects for us whenever request is made to it using "new keyword"

syntax:

```
class  A  {



}
```

What is new Keyword ?

- **It helps us to send a request to the class to create an object**
- **It's get the address of the object created and stores that in reference variable**

Syntax:

```
A a1 = new A();
```

Static and non static members of the class

non static member:

- **Whenever an object is created in your program non static member will get loaded into the object. Static member will never get loaded into the object**
- **These members they belong to the object**
- **Whenever you want to access non static variables firstly create an object and then use the variable. Without creating object non static variables cannot be accessed**
- **It is not mandatory to initialize non static variables, if we do not initialize it then depending on the data type default value gets stored init**

Static Member:

- **These members are loaded into the common memory of the class and they belong to the class**
- **To access static members do not create an object**
- **static members are loaded into the common memory of the class only once**
- **It is not mandatory to initialize static variables, if we do not initialize it then depending on the data type default value gets stored init**
- **static variables has global access and they can be accessed anywhere in the class**

Example 1:

```
public class A {
    int i = 10;//non static
    static int j = 500;//static
    int k = 500;//non static
    static int z = 1000;//static

    public static void main(String[] args){
      A a1 = new A();
      System.out.println(a1.i);
      System.out.println(a1.k);
      System.out.println(A.j);
      System.out.println(A.z);
    }
}
```

Output:

```
10
500
500
1000
```

Example 2:

```
public class A {

    int i  = 10;

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

```
     System.out.println(i);


    }
}
```

Output:

Error because without creating an object non static variable cannot be accessed

Example 3:

```
//What kind of variable is "i" ? 9632882052
public class A {


    int i  = 10;


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


    }
}
```

Output:

10

Example 4:

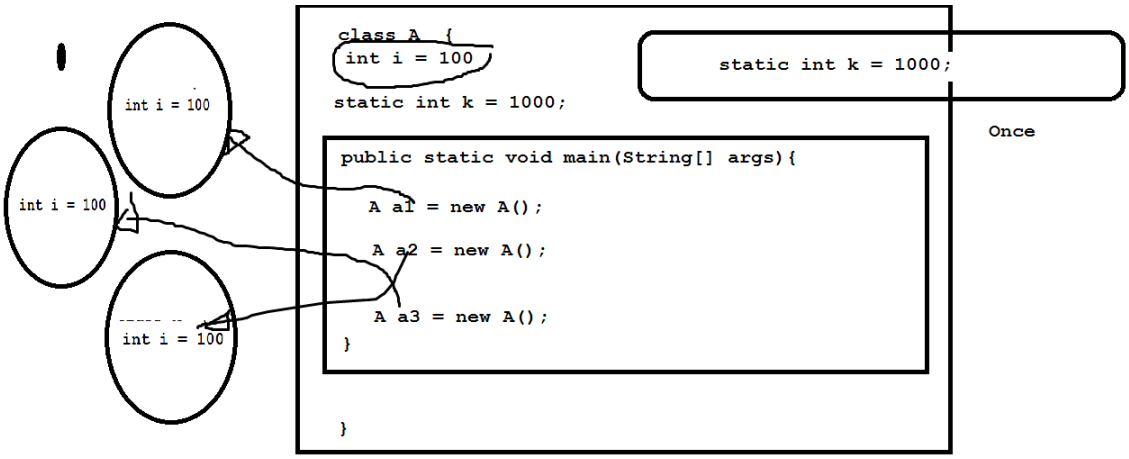
```
public class A {


    static int i  = 10;


    public static void main(String[] args){
      System.out.println(A.i);
```

```
    }
}
```

Output:

10

Example 5: for 3rd point in static section



Example 6:

value of a static variable can be changed as shown in the below program wherein the value of variable "i" is changed from 10 to 500

```
//What kind of variable is "i" ? 9632882052
public class A {
    static int i  = 10;
    public static void main(String[] args){
     System.out.println(A.i);
     A.i = 500;
     System.out.println(A.i);

    }
}
```

Output:

10

500

Example 7:

Note: non static variables values can be changed as shown in the below program

```
public class A {
   int i  = 10;
    public static void main(String[] args){
     A a1 = new A();
     System.out.println(a1.i);
     a1.i = 500;
     System.out.println(a1.i);

    }
}
```

Output:

10

500

Example 8:

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

Output:

0

Example 9:

```
public class A {


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

Output:

0

Example 10:

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


}
```

Output:

0

How to create methods in Java?

Example 1:

```
public class A {
   public static void main(String args[]) {
     A.test();
```

```
    }

    public static void test(){
        System.out.println(500);
    }
}
```

Output:

500

Example 2

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

    public void test(){
        System.out.println(500);
    }
}
```

Output:

500

Example 11:

```
public class A {

    static int i = 100;
```

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

    public void test(){
      System.out.println(A.i);
    }
}
```

Output:

100

100

Types of variables in Java

a. static variables

b. non staticvariables

c. local variables

d. reference variables

Local reference variables:

- **These variables are created inside a method**
- **These variables can be used only within created method**
- **These variables are accessed directly with its name**
- **These variables if not initialized and used then it gives us an error**

Example 1:

```
public class A {
   //Outside method we create static and non static variables
```

```
  public static void main(String args[]) {
   //Inside a method we create local variables
      int i = 10;//created
      System.out.println(i);//using
      A.test();
  }

   //Outside method we create static and non static variables
  public static void test(){
   //Inside a method we create local variables
   System.out.println(i);
  }
  //Outside method we create static and non static variables
}
```

Output:

Error because variable "i" cannot be used outside the created method

Example 2:

```
public class A {
  //Outside method we create static and non static variables

  public static void main(String args[]) {
   //Inside a method we create local variables
      int i = 10;//created
      System.out.println(i);//using

  }
```

```
}
```

Output:

10

Example 3:

```
public class A {

  public static void main(String args[]) {
    A a1 = new A();
    a1.test();

  }

  public void test(){
     int i = 10;
     System.out.println(i);

  }


}
```

Output:

10

Example 4:

```
public class A {

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

```
    A a1 = new A();
    a1.test();
    System.out.println(i);//Error
  }

  public void test(){
     int i = 10;//Created
     System.out.println(i);//Using

  }


}
```

Output:

Error because variable i is local and is used outside created method

Example 5:

```
public class A {

  public static void main(String args[]) {
    A a1 = new A();
    a1.test();
    System.out.println(i);//Error
  }

  public void test(){
     int i ;//Created and not initialized
     System.out.println(i);//Using
```

```
  }


}
```

Output:

Error because variable "i" cannot be used without initializing it

Reference Variable:

- **These variables are used to store objects address**
- **Reference variables are of two types**

1. **Local reference variable: These variables are created inside method and should be used only within created method. Outside created method if accessed it will give us an error**

```
Example 1:
public class A {

  public static void main(String args[]) {
     //Create Local variable here
     A a1 = new A();//Created here and initialized with objects address
     System.out.println(a1);
  }

  public void test(){
     System.out.println(a1);
  }
}
```

Output:Error

```
Example 2:
public class A {

  public static void main(String args[]) {
     A a1; //In this step object is not created and hence objects address is not stored in a1
     System.out.println(a1);
  }

}
```

Output:
Error

2. **Static reference variable: These variables are created outside all the methods but inside a class using static keyword. These variables has global access and can be used anywhere in the program.**
   If we do not initialize static variables then by default null value will be stored in it

Example1:

```
public class A {


  static A a1 = new A();

  public static void main(String args[]) {

    System.out.println(a1);

    a1.test();

  }


  public void test(){

     System.out.println(a1);

  }


}
```

Output:

A@7960847b

A@7960847b

Example 2:

Example 1:

```
public class A {

  public static void main(String args[]) {

      A a1 = 10;

      A a2 = "Pankaj";

  }
```

}

Output:

Error because reference variables can store only objects address

Example 2:

```
public class A {


  static A a1 ;//null value will be stored init

  public static void main(String args[]) {

    System.out.println(a1);


  }



}
```

Output:

null

Data Types In Java

Data Type	Default value	Memory Size
byte	0	1
short	0	2
Int	0	4
long	0	8
float	0.0	4
double	0.0	8
char	Empty space	2
boolean	flase	NA
var	NA	NA
String (class)	null	NA

Note: What is var datatype in java?

- var data type was introduced in JDK 1.10
- When the variable is created as var data type then it can store any kind of value. It can store heterogeneous value.

- var data type cannot be static or non static variable. var data type should be only local variable
- var data type cannot be used without initializing it

Example 1:

```
public class A {

   public static void main(String args[]) {
     var i = new A();
     var j = 10;
     var k = true;
     var z = 10.3;
     System.out.println(i);
     System.out.println(j);
     System.out.println(k);
     System.out.println(z);
   }
}
```

Output:

```
A@7960847b
10
true
10.3
```

Example 2:

```
public class A {
     static var i = new A();
     static var j = 10;
     static var k = true;
     static var z = 10.3;
   public static void main(String args[]) {

     System.out.println(A.i);
     System.out.println(A.j);
     System.out.println(A.k);
     System.out.println(A.z);
   }
}
```

Output:

```
/A.java:2: error: 'var' is not allowed here
     static var i = new A();
          ^
/A.java:3: error: 'var' is not allowed here
     static var j = 10;
          ^
```

/A.java:4: error: 'var' is not allowed here
    static var k = true;
        ^
/A.java:5: error: 'var' is not allowed here
    static var z = 10.3;
        ^
4 errors

Example 3:

```
public class A {

   public static void main(String args[]) {
     var i;
     System.out.println(i);

   }
}
```

Output:

error because variable "i" is var and should be used only after initializing it

Note: Static variables can be accessed in 3 ways

1. className.variableName
2. variableName
3. referenceVariable.variableName

Type casting:

1. variable type casting
2. class casting

Variable Type Casting: Converting a particular data type into required data type is called as typecasting

a.Auto Upcasting:

- Converting smaller data type to bigger data type is called as auto Upcasting
- During Auto Upcasting data loss should not happen

Example 1:

```
public class A {


   public static void main(String args[]) {

    byte i = 10;//Memory size of byte is 1 byte

    int j = i; //Here we are copying the data from 1 byte of memory to 4 bytes of memory

    System.out.println(j);
```

```
    }
}
```

Output:

10

Example 2

```
public class A {

    public static void main(String args[]) {
      float i = 1.3f;//Memory size of float is 4 bytes
      double j = i; //Here we are copying the data from 4 byte of memory to 8 bytes of memory
      System.out.println(j);
    }
}
```

Output:

1.2999999523162842

Example 3:

```
public class A {

    public static void main(String args[]) {
      long i = 10l;//Memory size of float is 8 bytes
      int j = i; //Here we are copying the data from 8 byte of memory to 4 bytes of memory
      System.out.println(j);
    }
}
```

Output:

Error, because data will not be automatically copied from bigger memory to smaller memory

Note:

For var data type type casting happens with the following

1. int

2. long

3. float

4. double

Example 4:

```
public class A {
    public static void main(String args[]) {
    float i = 10.3f;//Memory size of float is 4 bytes
    long j = i; //Here we are copying the data from 4 byte of memory to 8 bytes of memory
    System.out.println(j);
  }
}
```

Output:

```
/A.java:5: error: incompatible types: possible lossy conversion from float to long
    long j = i; //Here we are copying the data from 4 byte of memory to 8 bytes of memory
             ^
1 error
```

b. Explicit Downcasting:

- **Converting bigger data type to smaller data type is called as explicit downcasting**
- **During Explicit Downcasting data loss might happen**

Example 1:

```
public class A {
  public static void main(String args[]) {
    long i = 10;//Memory size of long is 8 bytes
    int  j = (int)i;//We are copying the data from bigger memory to smaller memory
    System.out.println(j);
  }
```

```
}
```

Output:

10

Example 2:

```
public class A {
   public static void main(String args[]) {
     double i = 10.3;//Memory size of double is 8 bytes
     float  j = (float) i;//We are copying the data from bigger memory to smaller memory
     System.out.println(j);
   }
}
```

Output:

10.3

Example 3:

```
public class A {
   public static void main(String args[]) {
     float i = 10.3f;//Memory size of double is 4 bytes
     long  j = (long) i;//Here data loss of .3 value will happen
     System.out.println(j);
   }
}
```

Output

10

Example 4:

```
public class A {
   public static void main(String args[]) {
     int i = 'd';
```

```
    System.out.println(i);
  }
}
```

Output:

100

Example 5:

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

Output:

2319

Example 6:

```
public class A {
  public static void main(String args[]) {
    int i = '৯';
    System.out.println(i);
  }
}
```

Output:

2962

Rules to design and develop methods

- **Rule 1: Program execution in java always begins with opening bracket of main method().**
- **Rule 2: Apply this rule on method calling statement. As per rule 2 transfer the control to the matching method**
- **Rule 3: when closing bracket of user defined method runs control is transferred back to calling statement**

- **Rule 4: When main method() closing bracket runs, then the whole program execution stops**

Example 1:

```
package intern_java_app_2;

public class A {

	public static void main(String[] args) {//Rule 1: STARTS

		A a1 = new A();//NoRule

		a1.test();//Rule 2

		System.out.println(100);

	}//Rule 4 STOPS



	public void test() {

		System.out.println(500);//NO Rule

	}//Rule 3




}
```

Output:

```
500

100
```

Example 2:

```
package app_java_1;

public class A {


	public static void main(String[] args) {//Rule 1 STARTS HERE

		System.out.println(100);//No Rule
```

```
		A a1 = new A();//No Rule
		a1.test();
		System.out.println(700);
		a1.test();//Rule 2
	}//Rule 4 STOPS

	public void test() {
		System.out.println(500);
	}//Rule 3 Rule 3


}
```

Output:

```
100
500
700
500
```

Example 3:

```
package app_java_1;

public class B {

	public static void main(String[] args) {//RULE 1 STARTS HERE
		B b1 = new B();// NO Rule
		b1.test1();//Rule 2

	}//Rule 4 STOPS
```

```
	public void test1() {
		B b2 = new B();
		b2.test2();//Rule 2
	}//Rule 3
	public void test2() {
		System.out.println(1000);
	}//Rule 3

}
```

Output:

1000

Example 4:

```
package app_java_1;


public class C {


	public static void main(String[] args) {//Rule 1 STARTS HERE
		C.test2();//Rule 2
		System.out.println("From Main");
		C c1 = new C();
		c1.test1();//Rule 2
	}//Rule 4 STOPS
	public void test1() {
		System.out.println(500);
	}
	public static void test2() {
		System.out.println(1000);
```

```
	}//Rule 3


}
```

Output:

1000

From Main

500

Example 5:

```
package app_java_1;

public class A {


	public static void main(String[] args) {

		A a1 = new A();

	   int var = a1.test();

	   System.out.println(var);


	}


	public int test() {

		return 500;

	}


}
```

Output:

500

Example 6:

```
package app_java_1;
```

```
public class A {

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

	}

	public String test() {
		return "Pankaj Sir Academy";
	}

}
```

Output:

Pankaj Sir Academy

Example 7:

Note: If a method is void then that method cannot return any value. Please refer the program below:

```
package app_java_1;

public class A {

	public static void main(String[] args) {



	}

	public void test() {
```

```
            return "Pankaj Sir Academy";
        }


}
```

Output:

Error because method is void and it cannot return any value

Example 8:

Note: If a method is void then in such methods we can use only return keyword in it. using only return keyword it means we are returning the control back to calling statement

```
package app_java_1;

public class A {


        public static void main(String[] args) {

                A a1 = new A();

                a1.test();


        }


        public void test() {

                System.out.println(500);

                return;

        }


}
```

Output:

500

Example 9:

Note:

If we right anything after return keyword then that line of code will never execute and hence it would give us an error "unreachable code"

```
package app_java_1;

public class A {


	public static void main(String[] args) {

		A a1 = new A();

		a1.test();


	}


	public void test() {

		return;

		System.out.println(500);

	}


}
```

Output:

Unreachable code

Example 10:

```
package app_java_1;

public class A {


	public static void main(String[] args) {

		A a1 = new A();

		int test = a1.test();

		System.out.println(test);

	}
```

```
	public int test() {
		return 100;
		System.out.println(500);

	}
}
```

Output:

Unreachable Code

Note:

- **If a method is void then such methods can never return any value. In void methods we can use only "return" keyword**
- **if method is not a void then such methods are developed to return a value**

Example 11:

```
package app_java_1;

public class A {

	public static void main(String[] args) {
	A.test(100,"Pankaj",10.3,true);
	}

	public static void test(int i, String s, double d, boolean b) {
		System.out.println(i);
		System.out.println(s);
		System.out.println(d);
		System.out.println(b);
	}
```

```
}
```

Output:

```
100
Pankaj
10.3
true
```

Example 12:

```
package app_java_1;
public class A {

	public static void main(String[] args) {
		A.test(100,200,300,500,600);
	}

	public static void test(int... x) {
		System.out.println(x[0]);
		System.out.println(x[1]);
		System.out.println(x[2]);
		System.out.println(x[3]);
		System.out.println(x[4]);
	}

}
```

Output:

```
100
200
```

300

500

600

Constructors In Java

- **Constructors should have same name as that of class**
- **Whenever an object is created Constructor would be called**
- **Constructors are internally permanently void**
- **We can create more than one constructor in the same class provided they have different number of arguments or different number of arguments**

Example 1:

```
package app_java_contructors;


public class A {


	A(){

		System.out.println("From Constructor");

	}

	public static void main(String[] args) {

		A a1 = new A();

		A a2 = new A();

		A a3 = new A();

	}

}
```

Output:

From Constructor

From Constructor

From Constructor

Example 2:

```
package app_java_contructors;


public class A {


	A(){

		return 100;

	}

	public static void main(String[] args) {

		A a1 = new A();

	}




}
```

Output:

Error because constructor can never return any value

Example 3:

Note: Because constructor is void we can use only return keyword in it

```
package app_java_contructors;


public class A {


	A(){

		System.out.println("A");

		return;

	}

	public static void main(String[] args) {

		A a1 = new A();
```

```
	}



}
```

Output:

A

Example 4:

Note: If we use void while creating a constructor then it is no longer a constructor it is treated as a method

```
package app_java_contructors;


public class A {


	void A(){

		System.out.println("A");

		return;

	}

	public static void main(String[] args) {

		A a1 = new A();

	}



}
```

Output:

In the above program "void A()" is a method and hence upon creating object it would not be called

Example 5:

```
package app_java_contructors;
```

```
public class A {


	void A(){

		System.out.println("A");

		return;

	}

	public static void main(String[] args) {

		A a1 = new A();

		a1.A();

	}




}
```

Output:

A

Example 6:

Note: class name, method name, constructor name all can be same as shown in the below example

```
package app_java_contructors;


public class main {


	void main(){

		System.out.println(500);

	}

	main(){

		System.out.println(100);

	}
```

```
	public static void main(String[] args) {
		main m = new main();
		m.main();

	}

}
```

Output:

100

500

Example 6:

```
package app_java_contructors;

public class A {

	A(int i,String s){
		System.out.println(i);
		System.out.println(s);
	}

	public static void main(String[] args) {

		A a1 = new A(100,"Pankaj Sir Academy");

	}
```

}

Output:

100

Pankaj Sir Academy

Example 7:

```
package app_java_contructors;


public class A {


	A(){//Number of args = 0
		System.out.println("A");
	}
	A(int i){//Number of args = 1
		System.out.println(i);
	}
	public static void main(String[] args) {
		A a1 = new A();
		A a2 = new A(100);




	}



}
```

Output:

A

100

Example 8:

```
package app_java_contructors;


public class A {


	A(){//Number of args = 0
		System.out.println("A");
	}
	A(int i){//Number of args = 1
		System.out.println(i);
	}
	A(int i,int j){//Number of args = 2
		System.out.println(i);
		System.out.println(j);
	}
	public static void main(String[] args) {
		A a1 = new A();
		A a2 = new A(100);
		A a3 = new A(500,1000);




	}
```

```
}
```

Output:

```
A

100

500

1000
```

Note: When we have more than one constructor in the same class we call that as constructor overloading

Example 9:

```
package app_java_contructors;


public class A {

	A(char i){//Number of args = 1 and type = char

		System.out.println(i);

	}

	A(int i){//Number of args = 1 and type = int

		System.out.println(i);

	}


	public static void main(String[] args) {


		A a1 = new A('a');

		A a2 = new A(100);

	}



}
```

Output:

a

100

Example 10:

Note: Calling a constructor from another constructor is called as constructor chaining as shown in the below example

```
package app_java_contructors;


public class A {


	A(){

		System.out.println(500);

	}

	A(int i){

		A a2 = new A();

	}


	public static void main(String[] args) {

		A a1 = new A(100);

	}




}
```

Output:

500

Example 11:

```
public class A {

	A(){
```

```
		System.out.println(100);
	}

	public static void main(String[] args) {
		A a1 = new A();
		a1.test();
	}

	public void test() {
		System.out.println("From test");
	}


}
```

Example 12:

```
package app_java_contructors;

public class A {
	A(){
		System.out.println(1000);
	}

	public static void main(String[] args) {
		System.out.println(new A());
	}
}
/*
```

*** what new keyword does**

*** 1. sending the request to the class to create object**

*** 2. it calls the constructor**

*** 3. Once the constructors is called it will get the address of the object**

***/**

Output:

1000

app_java_contructors.A@15db9742

Example 13:

```
package app_java_contructors;


public class A {


	public static void main(String[] args) {

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

	}

}

/*

 * what new keyword does

 * 1. sending the request to the class to create object

 * 2. it calls the constructor

 * 3. Once the constructors is called it will get the address of the object

 */
```

Output:

app_java_contructors.A@15db9742

Example 14:

```
package app_java_contructors;


public class A {
	int i = 1000;
	A(){
		System.out.println(100);
	}
	public static void main(String[] args) {
		System.out.println(new A().i);


	}
}
```

Output:

100

1000

Example 15:

```
package app_java_contructors;


public class A {


	A(){
		System.out.println(100);
	}
	public static void main(String[] args) {
		new A().test();
	}
```

```
	public void test() {
		System.out.println(5000);
	}
}
```

Output:

100

5000

Instance Initialization Block (IIB)

- **Whenever an object is created IIB will be called**
- **When we have more than one IIB in the same program then they would run in sequence whenever an object is created**
- **The main purpose of IIB is to initialize all non static variables in one place so that we have better readability of the code**

Example 1:

```
package app_java_contructors;


public class A {


	{
		System.out.println("From IIB");
	}
	public static void main(String[] args) {
		A a1 = new A();
		A a2 = new A();
	}


}
```

Output:

From IIB

From IIB

Example 2:

```
package app_java_contructors;


public class A {


	{

		System.out.println(10);

	}

	{

		System.out.println(500);

	}

	public static void main(String[] args) {

		A a1 = new A();


	}


}
```

Output:
10

500

Example 3:

Note: Always IIB will run first and then the constructor would run as shown in the below example

```
package app_java_contructors;


public class A {
```

```
	A(){
		System.out.println(5);
	}

	{
		System.out.println(10);
	}

	public static void main(String[] args) {
		A a1 = new A();

	}

}
```

Output:

10

5

Example 5:

```
package app_java_contructors;


public class A {
	{
		System.out.println(500);
	}
	A(){
		System.out.println(5);
	}
```

```
	{
		System.out.println(10);
	}

	public static void main(String[] args) {
		A a1 = new A();

	}

}
```

Output:

500

10

5

Example 5:

```
package app_java_contructors;

public class A {
	{
		System.out.println(500);
	}
	A(int i){
		System.out.println(i);
	}

	{
		System.out.println(10);
```

```
	}

	public static void main(String[] args) {
		A a1 = new A(100);

	}

}
```

Output:

500

10

100

Example 6:

```
package app_java_contructors;
public class A {
	int i,j,k;

	{
		i = 10;
		j = 20;
		k = 30;
	}

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

```
            System.out.println(a1.j);
            System.out.println(a1.k);
        }

}
```

Output:

10

20

30

Example 7:

Note: In IIB static variables can also be initialized, but it is advised not to be performed

```
package app_java_contructors;
public class A {
        static int i,j,k;

        {
            i = 10;
            j = 20;
            k = 30;
        }

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

```
	}


}
```

Output:

10

20

30

Static Initialization Block (SIB)

- **These block they run on its own and they run before main method**
- **If we have more than one SIB then it would execute in sequence**

Example 1:

```
package app_java_contructors;

public class A {

	static {

		System.out.println(500);

	}

	public static void main(String[] args) {

		System.out.println(5000);

	}


}
```

Output:

500

5000

Example 2:

```
package app_java_contructors;
```

```
public class A {
	static {
		System.out.println(500);
	}

}
```

Output:

without main method class cannot run

Example 3:

```
package app_java_contructors;
public class A {
	static {
		System.out.println(500);
	}
	static {
		System.out.println(6);
	}
	static {
		System.out.println(9);
	}
	public static void main(String[] args) {
		System.out.println(4);
	}

}
```

Output:

500

6

9

4

Example 4:

Note: Write a program to call main method more than once:

```
package app_java_contructors;

public class A {

	static {

		A.main(null);

		A.main(null);

	}

	public static void main(String[] args) {

		System.out.println(4);


	}


}
```

Output:

4

4

4

Note:

- **Always SIB runs first and then main method, if object created then IIB will run and finally the constructor would execute**

Example 1:

```
package app_java_contructors;
```

```
public class A {
	static {
		System.out.println(100);
	}
	{
		System.out.println(6);
	}
	A(){
		System.out.println(5);
	}

	public static void main(String[] args) {
		System.out.println(4);

	}

}
```

Output:

100

4

Example 2:

```
package app_java_contructors;

public class A {
	static {
		System.out.println(100);
```

```
	}
	{
		System.out.println(6);
	}
	A(){
		System.out.println(5);
	}

	public static void main(String[] args) {
		System.out.println(4);
		A a1 = new A();
	}

}
```

Output:

```
100
4
6
5
```

Example 3:

```
package app_java_contructors;

public class A {
	static {
		System.out.println(100);
	}
	{
```

```
		System.out.println(6);
	}
	A(){
		System.out.println(5);
	}
	{
		System.out.println(9);
	}
	static {
		System.out.println(0);
	}

	public static void main(String[] args) {
		System.out.println(4);
		A a1 = new A();
	}

}
```

Output:

100

0

4

6

9

5

Example 4:

```
package app_java_contructors;
```

```
public class A {
	static {
		System.out.println(100);
		A a1 = new A();
	}
	{
		System.out.println(6);
	}
	A(){
		System.out.println(5);
	}
	public static void main(String[] args) {
		System.out.println(4);
	}

}
```

Output:

100

6

5

4

Example 5:

```
package app_java_contructors;


public class A {
	static {
```

```
        System.out.println(100);

        A a1 = new A(1000);

    }

    {

        System.out.println(6);

    }

    A(int i){

        System.out.println(i);

    }

    public static void main(String[] args) {

        System.out.println(4);

    }


}
```

Output:

100

6

1000

4

Arrays:

- **Arrays are collection of elements**
- **In java arrays are special objects created to store elements in sequence starting with index number '0'**

Example 1:

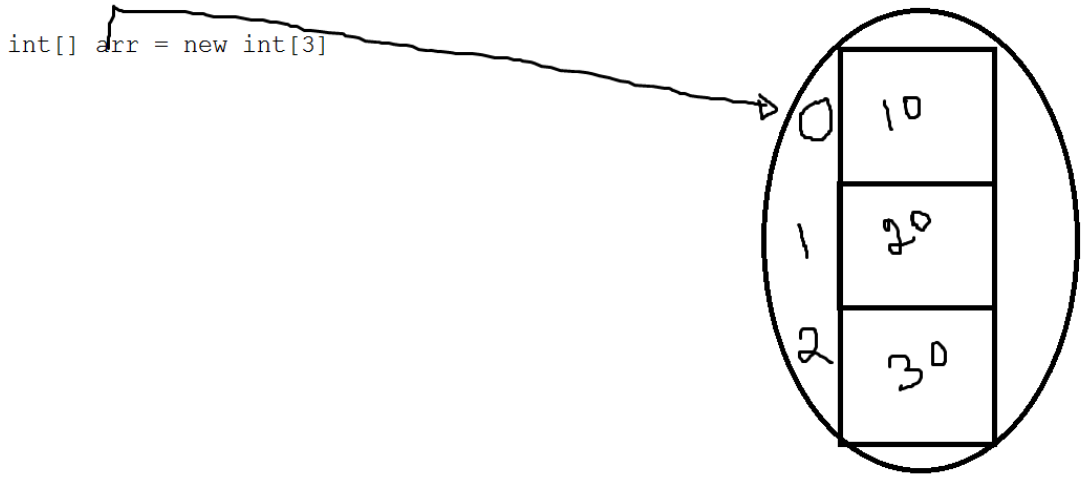
```
package app_java_contructors;


public class A {
```

```
        public static void main(String[] args) {
                int[] arr = new int[3];
                System.out.println(arr);
        }

}
```

Output:

[I@15db9742

Example 2:



Example 3:

```
package app_java_contructors;


public class A {
        public static void main(String[] args) {
                int[] arr = new int[3];
                arr[0] = 10;
                arr[1] = 20;
                arr[2] = 30;
                System.out.println(arr[0]);
```

```
                System.out.println(arr[1]);

                System.out.println(arr[2]);

        }


}
```

Output:

10

20

30

Example 4:

```
package app_java_contructors;


public class A {

        public static void main(String[] args) {

                String[] arr = new String[3];

                System.out.println(arr[0]);

                System.out.println(arr[1]);

                System.out.println(arr[2]);

        }


}
```

Output:
null

null

null

Example 5:

```
package app_java_contructors;
```

```
public class A {
	public static void main(String[] args) {
		int[] arr = {100,200,300};
		System.out.println(arr[0]);
		System.out.println(arr[1]);
		System.out.println(arr[2]);

	}

}
```

Output:

100

200

300

Example 6:

```
package app_java_contructors;

public class A {
	public static void main(String[] args) {
		int[] arr = new int[10];
		System.out.println(arr.length);


	}

}
```

Output:

10

Example 7:

```
package app_java_contructors;


public class A {
	public static void main(String[] args) {
		int[][] arr = new int[3][5];
		System.out.println(arr.length);
		System.out.println(arr[2].length);



	}

}
```

Output:

3

5

Example 8:

```
package arrays_demo;


public class A {
	public static void main(String[] args) {
		int[] arr = new int[5];
		arr[0] = 10;
		arr[1] = 100;
		arr[2] = 1000;
```

```
		for (int i = 0; i < arr.length; i++) {
			System.out.println(arr[i]);

		}
	}

}
```

Output:

```
10
100
1000
0
0
```

Example 9:

```
package arrays_demo;

public class A {
	public static void main(String[] args) {
		int[] arr = new int[5];
		arr[0] = 10;
		arr[1] = 100;
		arr[2] = 1000;

		//for each loop

		for (int i : arr) {
```

```
                System.out.println(i);
            }

        }

}
```

Output:

10

100

1000

0

0

Assignment: Find duplicate elements in an array and print only unique value of it

Example 10:

```
package arrays_demo;


public class A {
    public static void main(String[] args) {
        int[][] arr = new int[3][3];
        arr[0][0] = 10;

        for(int i=0;i<arr.length;i++) {
            for(int j=0;j<arr[2].length;j++) {
            System.out.println(arr[i][j]);
            }
        }
```

```
        }


}
```

Output:

10

0

0

0

0

0

0

0

0

public static void main(String[] args) Explanation ?

String[] args: It is command line argument that helps us to supply value to the main method during run time.

void: means main method cannot return any value

static: It means belongs to class common memory and main method is loaded into the common memory only once

Different ways to create main method in java program?

In java variable name can be $ and _ as well

Example 1:

```
package arrays_demo;


public class A {

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

```
            System.out.println("From main");

        }

}
```

Output:

From main

Example 2:

```
public class A {
        public static void main(String x[]) {

            System.out.println("From main");

        }

}
```

Output:

From main

Example 3:

```
package arrays_demo;

public class A {
        public static void main(String... x) {

            System.out.println("From main");

        }
```

}

Output:

From main

Example 4:

```
package arrays_demo;


public class A {
	public  void main(String... x) {


		System.out.println("From main");


	}


}
```

Output:

Error because main has to be static

Example 5:

```
package arrays_demo;


public class A {
	static void main(String... x) {


		System.out.println("From main");


	}
```

}

Output:

Error because main method has to be public

Note: If command line argument is not supplied to main method then the size of an array is ZERO. Depending on the number of command line arguments supplied during run time which dynamically allocate the size of an array in main method.

Naming Conventions to be followed in Java:

- **If a method consist of only one letter then it should be written in lower case. But in case if the method name has more than one word the we use camel casing as shown below:**
  Example
  addNumberValue()
- **If a variable consist of only one letter then it should be written in lower case. But in case if the method name has more than one word then we use camel casing as shown below:**
  Example
  int addNumberValue = 100;
- **Class Name Every word first Letter should be capital letter**

  Example: class BankStatement{ }

- **All keyword in java should be written in lower case**

What is this keyword in Java

- **this keyword is a special reference variable created in Java automatically to store objects address and it points to the current object running**
- **Using this keyword we can access non static members of object**
- **We cannot use this keyword inside static methods**
- **Using this keyword we can call constructor of the class but this calling should be done from another constructor of same class and it should be the first statement**

Example 1:

```
package app_java_2;


public class A {


    public static void main(String[] args) {

        A a1 = new A();

        System.out.println(a1);
```

```
		a1.test();
	}

	public void test() {
		System.out.println(this);
	}

}
```

Output:

```
app_java_2.A@15db9742
app_java_2.A@15db9742
```

Example 2:

```
package app_java_2;

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

	public void test() {
		System.out.println(this.i);
	}
```

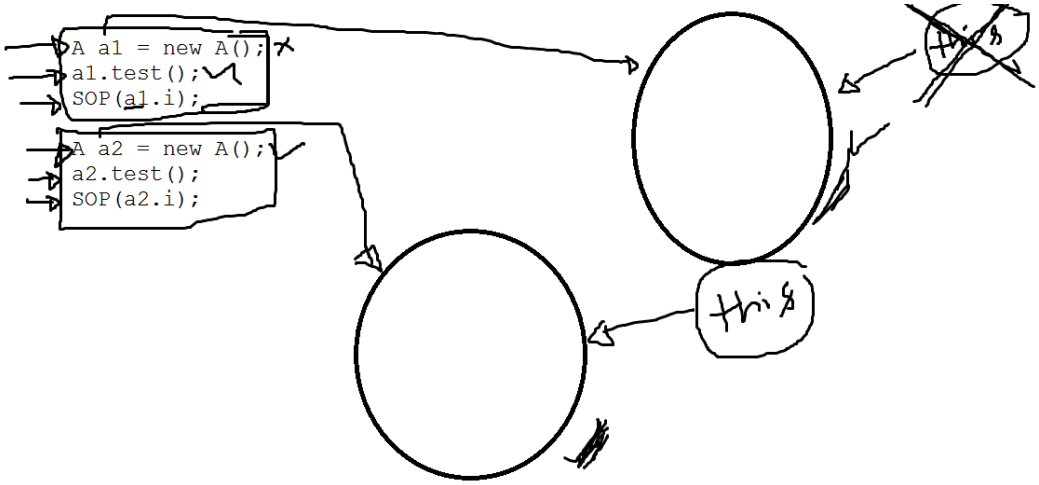
}

Output:

10

10

Note: this keyword points to current object running in your program



Example 3:

package app_java_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();

}

public void test() {

```
            System.out.println(this);
        }

}
```

Output:

```
app_java_2.A@15db9742
app_java_2.A@15db9742
app_java_2.A@6d06d69c
app_java_2.A@6d06d69c
```

Example 4:

Note: this keyword cannot be use in static context. Please refer the example below:

```
package app_java_2;

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

}
```

Output:

Error

Example 5:

```
package app_java_2;
```

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

}
```

Output:

Error because this keyword cannot be used in static context

Example 6:

Note: Using this keyword we can access static members of the class

```
package app_java_2;

public class A {
	static int i = 100;
	public static void main(String[] args) {
		A a1 = new A();
		a1.test();

	}
	public void test() {
		System.out.println(this.i);
	}

}
```

Output:

100

Example 7:

```
package app_java_2;


public class A {


	public static void main(String[] args) {

		A a1 = new A();

		a1.test1();


	}

	public  void test1() {

		this.test2();

	}

	public void test2() {

		System.out.println("From test 2");

	}


}
```

Output:

```
From test 2
```

Example 8:

```
package app_java_2;


public class A {


	public static void main(String[] args) {

		A a1 = new A();
```

```
		a1.test1();


	}
	public  void test1() {
		this.test2();
	}
	public static void test2() {
		System.out.println("From test 2");
	}


}
```

Output:

From test 2

Example 9:

Note:

In non static member if this keyword is not added then automatically java compiler would add that to access non static members of your class as shown in the below example

```
package app_java_2;


public class A {
	int i = 10;
	public static void main(String[] args) {
		A a1 = new A();
		a1.test1();


	}
	public  void test1() {
		System.out.println(i);
```

```
	}


}
```

Output:

10

Example 10:

```
package app_java_2;


public class A {
	int i = 10;
	public static void main(String[] args) {
		A a1 = new A();
		a1.test1();


	}
	public static void test1() {
		System.out.println(i);
	}


}
```

Output:

Error because in static methods this keyword cannot get added automatically.

Note:

Static variables / methods in java can be accessed in 3 ways:

1. className.memberName

2. memberName

3. referenceVariable.memberName

Example :

```
package app_java_2;


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


}
```

Output:

```
10
10
10
```

Example 11:

```
package app_java_2;


public class A {
	A(){
		System.out.println("From A");
	}
	A(int i){
		this();
	}
```

```
	public static void main(String[] args) {
		A a1 = new A(100);
	}

}
```

Output:

From A

Example 12:

```
package app_java_2;


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

}
```

Output:

100

Example 13:

```
package app_java_2;
```

```
public class A {
	A(){
		System.out.println(5);
		this(100);
	}
	A(int i){
		System.out.println(i);
	}
	public static void main(String[] args) {
		A a1 = new A();
	}

}
```

Output:

Error because this keyword cannot be the second statement while calling the constructor

Example 14:

```
package app_java_2;

public class A {
	A(){
		this(100);
		System.out.println(5);

	}
	A(int i){
		System.out.println(i);
	}
```

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

}
```

Output:

100

5

Inheritance In Java

- **Here we inherit non static members from the parent class to child class object**
- **The main purpose of inheritance is to create re-usability of parent class members in child class**
- **In java multiple inheritance is not allowed because multiple inheritance results in complex designing of the software**

Example 1:

```
package inheritance_examples;

public class A {//Parent , Super class
        int i = 10;
}
```

```
package inheritance_examples;


public class B extends A{
        public static void main(String[] args) {
                B b1 = new B();
                System.out.println(b1.i);
        }
}
```

Output:

10

Example 2:

```
package inheritance_examples;


public class A {//Parent,Super class
	int i = 10;
	public void test() {
		System.out.println("From Test");
	}
}
```

```
package inheritance_examples;


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

Example 3:

```
package inheritance_examples;


public class A {
	public void test1() {
		System.out.println("From test 1");
	}
}
```

```
package inheritance_examples;
```

```
public class B extends A{//test1() & test2()


	public void test2() {

		System.out.println("From test 2");

	}


}
```

```
package inheritance_examples;


public class C extends B{//test1() + test2() + test3()


	public void test3() {

		System.out.println("From test 3");

	}

	public static void main(String[] args) {

		C c1 = new C();

		c1.test1();

		c1.test2();

		c1.test3();

	}


}
```

Output:

```
From test 1

From test 2

From test 3
```

Example 4:

```
package inheritance_examples;


public class A {


}
```

```
package inheritance_examples;


public class B {




}
```

```
package inheritance_examples;


public class C extends A,B{


}
```

Output:

Error because multiple inheritance in java is not allowed

Example 5:

Note: In the below example class A and class B are non sub classes.

```
package inheritance_examples;


public class A {

        int i = 10;

}
```

```
package inheritance_examples;


public class B {

	public static void main(String[] args) {

		A a1 = new A();

		System.out.println(a1.i);

	}


}
```

Output:

10

Example 6:

Note: static members in Java are not inherited. In the below example b1.i is converted to A.i during run time as shown below:

```
package inheritance_examples;


public class A {

static int i = 10;

}
```

```
package inheritance_examples;


public class B extends A {

	public static void main(String[] args) {

		B b1 = new B();

		System.out.println(b1.i);//A.i

	}
```

}

Output:

10

Example 7:

Note: Static members are not inherited. In the below example during run time B.i will be converted to A.i and hence we get the output as 10

```
package inheritance_examples;


public class A {

static int i = 10;

}
```

```
package inheritance_examples;


public class B extends A {

        public static void main(String[] args) {

                System.out.println(B.i);//A.i

        }


}
```

Output:

10

Polymorphism in java

- **Developing a feature in java that can take more than one form is called as polymorphism**

There are two types of polymorphism:

- **Overriding**
- **Overloading**

Overriding:

- **Here we inherit a method from parent class and then we replace the methods logic by once again creating a method with same signature in child class**
- **Overriding helps to modify the logic of inherited method based on our requirement.**

Note: Polymorphism cannot be applied on variables

Example 1:

```
package polymorphism;


public class A {

	public void test() {

		System.out.println("From class A test() method");

	}

}
```

```
package polymorphism;


public class B extends A{

	public void test() {

		System.out.println(500);

	}


	public static void main(String[] args) {

		B b1 = new B();

		b1.test();

	}


}
```

Output:

500

Example 2:

```
package polymorphism;


public class A {
	public void test() {
		System.out.println("From class A test() method");
	}
}
package polymorphism;


public class B extends A{
	public void test() {
		System.out.println(500);
	}


	public static void main(String[] args) {
		B b1 = new B();
		b1.test();


		A a1 = new A();
		a1.test();
	}


}
```

Output:

500

From class A test() method

@Override: It check whether the method signature in the parent class and the child class it matches, if the match does not happen then it would give us an error as shown in the below example.

Example 3:

```
package polymorphism;


public class A {

	public void test() {

		System.out.println("From class A test() method");

	}

}
```

```
package polymorphism;


public class B extends A{


	@Override

	public void tests() {

		System.out.println(500);

	}


	public static void main(String[] args) {

		B b1 = new B();

		b1.test();


		A a1 = new A();

		a1.test();

	}
```

}

Output:

Error

Example 4:

```
package polymorphism;


public class A {
	public void test() {
		System.out.println("From class A test() method");
	}
}
package polymorphism;


public class B extends A{


	@Override
	public void test() {
		System.out.println(500);
	}


	public static void main(String[] args) {
		B b1 = new B();
		b1.test();


		A a1 = new A();
		a1.test();
	}
```

```
}
```

Output:

```
500
From class A test() method
```

Example 5:

```
package polymorphism;

public class A {
	public void test1() {
		System.out.println("From class A test1() method");
	}
	public void test2() {
		System.out.println("From class A test2() method");
	}
}
```

```
package polymorphism;

public class B extends A{
	@Override
	public void test1() {
		System.out.println("From class B test1() method");
	}
	public static void main(String[] args) {
		B b1 = new B();
```

```
        b1.test1();

        b1.test2();

    }


}
```

Output:

From class B test1() method

From class A test2() method

Overloading

- **Develop methods in the same class with same name provided they have different number of arguments or different type of arguments.**

Example 1:

```
package polymorphism;


public class Email {


    // 1. Transactional Emailer

    // 2. Promotional Emailer


    public void sendEmail(String TCID, String emailId) {

        System.out.println("Transaction emailer");

    }


    public void sendEmail(String emailId) {

        System.out.println("promotional emailer");

    }

    public static void main(String[] args) {

        Email email = new Email();
```

```
            email.sendEmail("pankaj@gmail.com");

            email.sendEmail("xvy100", "pankaj@gmail.com");


        }


}
```

Output:

promotional emailer

Transaction emailer

What are packages in Java?

- **Packages in java are nothing but folders created to store programs in organized manner**
- **Packages resolves naming convention problem**

Example 1:

```
package p1;

public class A {


}
```

Example 2:

```
package p2;

public class C {


}
```

Example 3:

```
package p3.p4.p5;

public class D {


}
```

Example 4:

```
package p1;

public class A {

	int i = 10;

}

package p1;

public class B extends A{

	public static void main(String[] args) {

		B b1 = new B();

		System.out.println(b1.i);

	}


}
```

output:

10

Example 5:

```
package p1;

public class A {

	int i = 10;

}

package p2;

public class C extends A{

	public static void main(String[] args) {

		C c1 = new C();

		System.out.println(c1.i);

	}

}
```

Output:

Error because accessing a class present in different package cannot be done without importing it

Example 6:

```
package p1;

public class A {

	public int i = 10;

}
```

```
package p2;


import p1.A;


public class C extends A{

	public static void main(String[] args) {

		C c1 = new C();

		System.out.println(c1.i);

	}

}
```

Output:

10

Example 7:

```
package p3.p4.p5;


public class D {

	public int i = 10;

}
```

```
package p2;


import p3.p4.p5.D;
```

```
public class C extends D{
	public static void main(String[] args) {
		C c1 = new C();
		System.out.println(c1.i);
	}
}
```

Output:

10

Example 8:

```
package p1;
public class A {
	public int i = 10;
}
```

```
package p2;

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

Example 9:

```
package p3.p4.p5;

public class D {
	public int i = 10;
}
```

```
package p2;


public class C extends p3.p4.p5.D{
	public static void main(String[] args) {
		p3.p4.p5.D d1 = new p3.p4.p5.D();


	}
}
```

Example 10:

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


}
package p2;
import p1.*;


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

```
        A a1 = new A();

        B b1 = new B();

    }

}
```

Access Specifier / modifier

	private	default	protected	public
Same class	yes	yes	yes	yes
Same package sub class	No	yes	yes	yes
Same package non sub class	No	yes	yes	yes
different package sub class	No	No	yes	yes
different package non sub class	No	no	no	yes

Private Members:

Example 1:

```
package p1;


public class A {


    private int i = 10;

    private void test() {

        System.out.println("from test");

    }


    public static void main(String[] args) {

        A a1 = new A();

        System.out.println(a1.i);

        a1.test();
```

```
	}


}
```

Output:

```
10
from test
```

Example 2:

```
package p1;


public class A {


	private int i = 10;

	private void test() {

		System.out.println("from test");

	}
}
```

```
package p1;


public class B extends A{


	public static void main(String[] args) {

		B b1 = new B();

		System.out.println(b1.i);

		b1.test();

	}


}
```

Output:

Error

Example 3:

```
package p1;


public class A {


	private int i = 10;

	private void test() {

		System.out.println("from test");

	}

}
```

```
package p1;


public class B {


	public static void main(String[] args) {

		A a1 = new A();

		System.out.println(a1.i);

		a1.test();

	}


}
```

Output:

Error

Example 4:

```
package p1;
```

```
public class A {

	private int i = 10;
	private void test() {
		System.out.println("from test");
	}
}
```

```
package p2;

import p1.A;

public class C extends A{
	public static void main(String[] args) {
		C c1  = new C();
		System.out.println(c1.i);
		c1.test();
	}
}
```

Output:

Error

Example 5:

```
package p1;

public class A {

	private int i = 10;
```

```
	private void test() {
		System.out.println("from test");
	}
}
```

```
package p2;

import p1.A;

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

Output:

Error

Note: If a member is made private then that member is accessible only in same class

default members:

Example 1:

```
package p1;

public class A {

	int i = 10;
	void test() {
```

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

Output:

10

from test

Example 2:

```
package p1;

public class A {

        int i = 10;
        void test() {
                System.out.println("from test");
        }

}
```

```
package p1;

public class B extends A{

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

```
		B b1 = new B();
		System.out.println(b1.i);
		b1.test();
	}

}
```

Output:

```
10
from test
```

Example 3:

```
package p1;

public class A {

	int i = 10;
	void test() {
		System.out.println("from test");
	}

}
```

```
package p1;

public class B {

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

```
		a1.test();
	}

}
```

Output:

10

from test

Example 4:

```
package p1;

public class A {

	int i = 10;
	void test() {
		System.out.println("from test");
	}

}
```

```
package p2;

import p1.A;

public class C extends A{
	public static void main(String[] args) {
		C c1 = new C();
		System.out.println(c1.i);
		c1.test();
```

```
	}
}
```

Output:

Error

Example 5:

```
package p1;


public class A {


	int i = 10;
	void test() {
		System.out.println("from test");
	}


}
```

```
package p2;


import p1.A;


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

Output:

Error

Note: If you make a member default then it is accessible only in same class, outside that package it is not accessible

Protected :

Example 1:

```
package p1;


public class A {


	protected int i = 10;

	protected void test() {

		System.out.println("from test");

	}

	public static void main(String[] args) {

		A a1 = new A();

		System.out.println(a1.i);

		a1.test();

	}


}
```

Output:

10

from test

Example 2:

```
package p1;


public class A {
```

```
	protected int i = 10;
	protected void test() {
		System.out.println("from test");
	}

}
```

```
package p1;

public class B extends A{

	public static void main(String[] args) {
		B b1 = new B();
		System.out.println(b1.i);
		b1.test();
	}

}
```

Output:

```
10
from test
```

Example 3:

```
package p1;

public class A {

	protected int i = 10;
	protected void test() {
```

```
		System.out.println("from test");
	}

}
package p1;

public class B {

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

}
```

Output:

```
10
from test
```

Example 4:

```
package p1;

public class A {

	protected int i = 10;
	protected void test() {
		System.out.println("from test");
	}
```

```
}
package p2;


import p1.A;


public class C extends A{
	public static void main(String[] args) {
		C c1 = new C();
		System.out.println(c1.i);
		c1.test();
	}
}
```

Output:

```
10
from test
```

Example 5:

```
package p1;


public class A {


	protected int i = 10;
	protected void test() {
		System.out.println("from test");
	}


}
```

```
package p2;


import p1.A;


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

Output:

Error

Protected: When you make a member protected then it is accessible in same package and also accessible in different package but only through inheritance

Question 1: What all access specifier does a class supports ?

a. private

b. default

c. protected

d. public

Question 2: Which of the following class file names are valid

program:

```
class A{
}
public class B{
}
class c{
}
```

class D{

}

Options:

a. A.java

b. B.java - Correct answer

c. C.java

d. D.java

e. Hello.java

Which of the above options are correct

Question 3:

Which of the access specifier does a constructor supports

a. private

b. default

c. protected

d. public

Explain encapsulation in java ?

Note:

- **When you make a class default then it would be accessible only in the same package**
- **When you make a class public then it would be accessible in any same package**

What is encapsulation in java?

- **Design a class in a way where in all the variables of the class is made private and the variables are accessed only through getters and setters.**
- **Encapsulation is used to hide the values or state of variables**
- **We are developing security around class data members**
- **Direct access to these variables from another class is not allowed**

Example 1:

```
package app_java_encapsulation;


public class A {


	private int id;


	public int getId() {

		return id;

	}


	public void setId(int id) {

		this.id = id;

	}


	public static void main(String[] args) {

		A a1 = new A();

		a1.setId(500);

		System.out.println(a1.getId());

	}




}
```

Example 2:

```
package app_java_encapsulation;

//POJO
```

```
public class A {

	private int id;

	private String firstName;

	private String lastName;

	private String emailId;

	private String mobileNumber;


	public int getId() {

		return id;

	}

	public void setId(int id) {

		this.id = id;

	}

	public String getFirstName() {

		return firstName;

	}

	public void setFirstName(String firstName) {

		this.firstName = firstName;

	}

	public String getLastName() {

		return lastName;

	}

	public void setLastName(String lastName) {

		this.lastName = lastName;

	}

	public String getEmailId() {
```

```
		return emailId;
	}
	public void setEmailId(String emailId) {
		this.emailId = emailId;
	}
	public String getMobileNumber() {
		return mobileNumber;
	}
	public void setMobileNumber(String mobileNumber) {
		this.mobileNumber = mobileNumber;
	}



}
package app_java_encapsulation;

public class B {

	public static void main(String[] args) {
		A a1 = new A();
		a1.setId(100);
		a1.setFirstName("Pankaj");
		a1.setLastName("mutha");
		a1.setEmailId("pankaj@gmail.com");
		a1.setMobileNumber("9632882052");
```

```
            System.out.println(a1.getId());

            System.out.println(a1.getFirstName());

            System.out.println(a1.getLastName());

            System.out.println(a1.getEmailId());

            System.out.println(a1.getMobileNumber());

        }


}
```

Output:

100

Pankaj

mutha

pankaj@gmail.com

9632882052

Interfaces In Java

- **An interface can consist of only incomplete methods**
- **When a class implements an interface then it means that the class should complete all incomplete methods inherited from an interface or else you would get an error.**
- **Interfaces are just like a contract that the class gets into and the class should complete all incomplete methods of an interface**
- **In java interfaces supports multiple inheritance**

Example 1:

```
package app_interfaces_examples;


public interface A {


        public void test() ;


        public void x() {
```

```
        }

}
```

Output:

Error, Because an interface cannot consist of complete method

Example 2:

```
package app_interfaces_examples;


public interface A {


        public void test() ;


}
```

Output:

It will compile

Example 3:

```
package app_interfaces_examples;


public interface A {


        public void test() ;


}
```

```
package app_interfaces_examples;


public class B implements A{
```

```
}
```

Output:

Example 4:

```
package app_interfaces_examples;


public interface A {


	public void test() ;


}
package app_interfaces_examples;


public class B implements A{


	@Override
	public void test() {
		System.out.println(1000);


	}
	public static void main(String[] args) {
		B b1 = new B();
		b1.test();
	}
}
```

Output:

1000

Example 5:

```
package app_interfaces_examples;


public interface A {


	public void test1() ;

	public void test2();


}
package app_interfaces_examples;


public class B implements A{


	@Override

	public void test1() {

		System.out.println("From Test 1");

	}


	@Override

	public void test2() {

		System.out.println("From Test 2");

	}

	public static void main(String[] args) {

		B b1 = new B();

		b1.test1();
```

```
            b1.test2();
        }


}
```

Output:

From Test 1

From Test 2

Example 6:

```
package app_interfaces_examples_1;

public interface A {

        public void createCustomerRecord();
        public void deleteCustomerRecord();
        public void updateCustomerRecord();
        public void readCustomerRecord();

}
package app_interfaces_examples_1;

public class B implements A{

        @Override
        public void createCustomerRecord() {
                System.out.println("Create Record");
```

```
	}

	@Override
	public void deleteCustomerRecord() {
		System.out.println("Delete Record");
	}

	@Override
	public void updateCustomerRecord() {
		System.out.println("Update Record");
	}

	@Override
	public void readCustomerRecord() {
		System.out.println("Read Customer Record");
	}
	public static void main(String[] args) {
		B b1 = new B();
		b1.createCustomerRecord();
		b1.deleteCustomerRecord();
		b1.updateCustomerRecord();
		b1.readCustomerRecord();
	}

}
```

Output:

```
Create Record
```

Delete Record

Update Record

Read Customer Record

Note:

- **When we do inheritance from class to class we use extends keyword**
- **when we do inheritance from interface to class we use implements keyword**
- **when we do inheritance from interface to interface we use extends keyword**

Example 7:

```
package app_interfaces_example_2;


public interface A {


	public void test1();


}
```

```
package app_interfaces_example_2;


public interface B extends A{//test1() + test2()

	public void test2();

}
```

```
package app_interfaces_example_2;


public interface C extends B{//test1() + test2() + test3()-

	public void test3();

}
```

```
package app_interfaces_example_2;
```

```
public class D implements C{

	@Override
	public void test1() {
		System.out.println("From test 1");

	}

	@Override
	public void test2() {
		System.out.println("From test 2");
	}

	@Override
	public void test3() {
		System.out.println("From test 3");

	}
	public static void main(String[] args) {
		D d1 = new D();
		d1.test1();
		d1.test2();
		d1.test3();
	}

}
```

Example 8:

```
package app_interfaces_example_2;


public interface A {


	public void test1();


}
package app_interfaces_example_2;


public interface B {
	public void test2();
}
package app_interfaces_example_2;


public interface C extends A,B{//test1() + test2() + test3()-
	public void test3();
}
package app_interfaces_example_2;


public class D implements C{


	@Override
	public void test1() {
		System.out.println("From test 1");


	}
```

```
	@Override
	public void test2() {
		System.out.println("From test 2");
	}

	@Override
	public void test3() {
		System.out.println("From test 3");

	}
	public static void main(String[] args) {
		D d1 = new D();
		d1.test1();
		d1.test2();
		d1.test3();
	}

}
```

Output:

```
From test 1
From test 2
From test 3
```

abstract keyword in java

- This keyword is used to define a method or a class is incomplete

Example 9:

```
package app_interfaces_example;


public interface A {

	public abstract void test1();

}
```

```
package app_interfaces_example;


public interface B {


	public void test2();


}
```

```
package app_interfaces_example;


public interface C {

	public abstract void test3();

}
```

```
package app_interfaces_example;


public class D implements A,B,C{


	@Override

	public void test3() {

		System.out.println("From test 3");
```

```
	}

	@Override
	public void test2() {
		System.out.println("From test 2");
	}

	@Override
	public void test1() {
		System.out.println("From test 1");
	}
	public static void main(String[] args) {
		D d1 = new D();
		d1.test1();
		d1.test2();
		d1.test3();
	}

}
```

Output:

From test 1

From test 2

From test 3

Example 10:

Note: On a particular class we can use extends and implements both of these keyword but ensure extends is used first and then implements is used as shown in the below example

```
package app_interfaces_example;


public interface A {


	public abstract void test1();


}
package app_interfaces_example;


public interface B {

	public abstract void test2();

}
package app_interfaces_example;


public class C {


	public void test3() {

		System.out.println("From test 3");

	}


}
package app_interfaces_example;


public class D extends C implements A,B {


	@Override

	public void test2() {
```

```
		System.out.println("From test 2");
	}

	@Override
	public void test1() {
		System.out.println("From test 1");
	}
	public static void main(String[] args) {
		D d1 = new D();

	}

}
```

Output:

From test 1

From test 2

From test 3

Example 11:

```
package app_interfaces_example;

public interface A {

	public abstract void test1();

}
```

```
package app_interfaces_example;
```

```
public class B {

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


}
```

Example 12:

```
package app_interfaces_example;

public interface A {

	public abstract void test1();

}
```

```
package app_interfaces_example;

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


}
```

Output:

null

Summary of Interfaces

- Interfaces are 100% abstract / incomplete
- When a class implements an interface then the class should complete all of the incomplete method which is inherited from an interface
- We cannot create an object for an interface but a reference variable of an interface can be created
- main method cannot be created in an interface

Note:

Class up casting: Here we create reference variable of parent class / interface and store child class object address in it

Example

```
package app_interfaces_example;


public interface A {




}
```

```
package app_interfaces_example;


public class B implements A{

	static A a1 ;

	public static void main(String[] args) {

		//Class up casting


		A a1 = new B();

		System.out.println(a1);

	}
```

```
}
```

Output:

```
app_interfaces_example.B@15db9742
```