Intern batch 4PM Notes

What is a class?

- class is a factory that generates objects for us whenever it receives a request from "new keyword"

Syntax

```
class    name {




}
```

What is new keyword?

- It helps us to send a request to the class to create an object
- Once the class creates an object new keyword will get the address of the object and store that in reference variable

Syntax:

```
new    className();
```

Example 1:

```
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@7960847b

A@6a6824be

A@5c8da962
```

Static member versus non static member in java

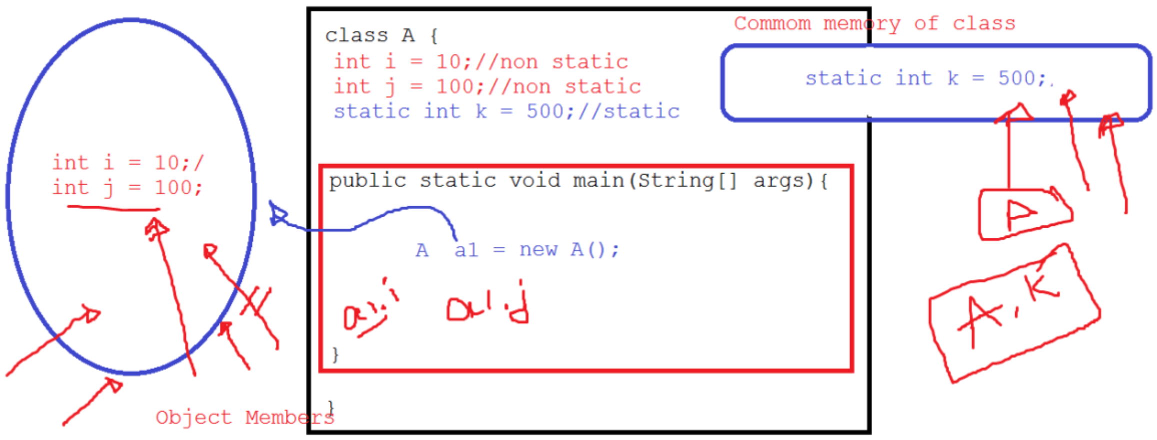
non static:

- Whenever an object is created only non static member would get loaded into the object
- non static members are also called as Object member
- It is not mandatory to initialize non static variables, if we do not initialize then depending on the data type auto initialization would happen

static:

- These members belongs the class and is loaded into the class common memory
- static members are loaded into the class common memory only once
- When we create static variable it has to created outside the method but inside a class using static keyword
- It is not mandatory to initialize static variables, if we do not initialize then depending on the data type auto initialization would happen
- If a variable is static then it means we can use that variable anywhere in the program

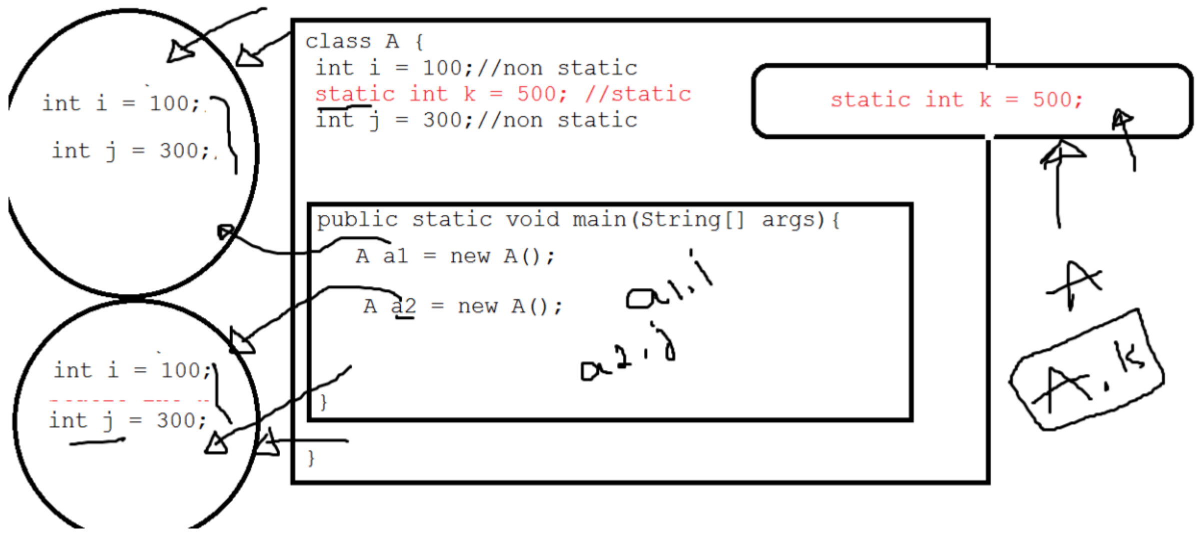
Example 1:



```
class A {
 int i = 10;//non static
 int j = 100;//non static
 static int k = 500;//static

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

Example 2:



```
class A {
 int i = 100;//non static
 static int k = 500; //static
 int j = 300;//non static

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

   A a2 = new A();

 }
}
```

Example 3:

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

Output:

```
10
500
```

Example 4:

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

Output:

Error because non static member cannot be used without creating object

Example 5:

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

Output:

```
10
```

Example 6:

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

Output:

```
10
```

Example 7:

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

        A.i = 100;

        System.out.println(A.i);

    }

}
```

Output:

```
10

100
```

Example 8:

```
public class A {

    int i = 10;

    public static void main(String[] args){

        A a1 = new A();

        System.out.println(a1.i);

        a1.i = 100;

        System.out.println(a1.i);

    }

}
```

Output:

```
10

100
```

Example 9:

```
public class A {

    static int i;

    public static void main(String[] args){

        System.out.println(A.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
```

Types of variables in java

1. **Local variables - Created Inside Method**
2. **static variables - Are class variables and created outside method but inside class**
3. **non static variables - Are Object variables and only after object creation we can use it & are created outside method but inside class**
4. **reference variables - They are created to store objects address.**

Note:

Lets understand what are methods:

```
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
```

Local Variables In Java

- Local variables are created inside a method and should be used only within created method
- Local variables are used directly with its name
- We can use local variables only after initializing it

  Example 1:

  ```
  public class A {

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

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

Output:

Error because variables is created in main method but is being used outside the created method

Example 2:

```
public class A {

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

}
```

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);
  }
```

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

Output:

Error because variable "i" is created inside test() method but it is being used inside main() method

Example 5:

```
public class A {

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

}
```

Output:

Error because variable "i" is local variable and without initializing it we cannot use it

Example 6:

```
public class A {

   static int k = 500;//static variable has global access

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

   public static void test(){
      System.out.println(A.k);
   }

}
```

Output:

```
500
500
```

Reference Variables in Java

- Reference variable can store only objects memory address and the data type of reference variable is always class name

Types of reference variables

Local reference variables:

- Local reference variables are created inside a method and should be used only inside created method

Example 1:

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

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

}
```

Output:

Error because reference variable "a1" is created in main method and should be used only within main method

Static reference variable:

- These variables are created outside all the methods but inside a class using static keyword and these variables have global access

Example 1:

```
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:

```
public class A {

   static A a1 ;

   public static void main(String args[]) {

         System.out.println(a1);

    }

 }
```

Output:

```
null
```

Example 3:

```
public class A {


   public static void main(String args[]) {

         A a1  ;

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

Output:

Error

Note:

Static variables are not mandatory to be initialized but it is mandatory to initialize local variables

Data Type In Java

Data Type	Memory Size	Default Values
var	NA	NA
byte-Integer	1 byte	0
short-Integer	2 bytes	0
int-integer	4 bytes	0
long-integer	8 bytes	0
float-decimal	4 bytes	0.0
double-decimal	8 bytes	0.0
char	2 bytes	Empty Space
boolean	NA	false
String (Class)	NA	null

Note:

- var data type was introduced in JDK 1.10
- var data type can store any kind of value in it.
- var data type can be only local variable. It cannot be static / non static variable

Example 1:

```
public class A {

   public static void main(String args[]) {

     var i = 10;

     var j = 10.3;

     var k = "Pankaj Sir Academy";

     var z = new A();

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

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

      System.out.println(k);

      System.out.println(z);

   }

}
```

Output:

```
10

10.3

Pankaj Sir Academy

A@311d617d
```

Example 2:

```
public class A {

     static var i = 10;

     var j = 500;

   public static void main(String args[]) {




   }

}
```

Output:

```
/A.java:2: error: 'var' is not allowed here

     static var i = 10;

            ^

/A.java:3: error: 'var' is not allowed here

     var j = 500;

     ^
```

```
2 errors
```

Example 3:

```
public class A {


   public static void main(String args[]) {


    var i ;

    System.out.println(i);




   }

}
```

Note:

Error because var data type can be only local variable and hence initializing it becomes mandatory

Example 4:

```
public class A {


   public static void main(String args[]) {

    long mobileNumber = 9632882052l;

    System.out.println(mobileNumber);




   }

}
```

Output:

```
9632882052
```

Example 5:

```
public class A {


   public static void main(String args[]) {

    int i = 1_00_000;

    System.out.println(i);




   }

}
```

Output:

```
100000
```

Type Casting:

- Converting a particular data type into required data type is called as type casting

1. Auto Upcasting

- Converting a smaller data type to bigger data type is called as auto up casting
- During Upcasting if data loss happens then auto Upcasting will not happen.

Example 1:

```
public class A {


   public static void main(String args[]) {

    int i = 10; //Memory Size = 4 bytes

    long j = i; //Memory Size = 8 bytes

    System.out.println(j);




   }

}
```

Output:

```
10
```

Example 2:

```
public class A {


   public static void main(String args[]) {
    float i = 10.3F; //Memory Size = 4 bytes
    double j = i; //Memory Size = 8 bytes
    System.out.println(j);



   }
}
```

Output:

```
10.300000190734863
```

Example 3:

```
public class A {


   public static void main(String args[]) {
    long i = 10L; //Memory Size = 8 bytes
    int j = i; //Memory Size = 4 bytes
    System.out.println(j);



   }
}
```

Note:

Copying data from bigger to smaller memory is not done automatically by our java compiler

Example 4:

```
public class A {


   public static void main(String args[]) {

    float i = 10.3F; //Memory Size = 4 bytes

    long j = i; //Memory Size = 8 bytes

    System.out.println(j);




   }

}
```

Output:

```
A.java:5: error: during conversion data loss is happening hence auto Upcasting cannot take place

    long j = i; //Memory Size = 8 bytes

             ^

1 error
```

2. Explicit Downcasting

- **Here we convert bigger data type to smaller data type**
- **Explicit Downcasting might result in data loss**

Example 1:

```
public class A {

   public static void main(String args[]) {

     long i = 10l;//Memory = 8 bytes

     int  j =(int) i; //Memory = 4 bytes

     System.out.println(j);

   }

}
```

Output:

```
10
```

Example 2:

```java
public class A {
   public static void main(String args[]) {
    double i = 10.3;//Memory = 8 bytes
    float j =(float)  i; //Memory = 4 bytes
    System.out.println(j);
   }
}
```

Output:

```
10.3
```

Example 3:

```java
public class A {
   public static void main(String args[]) {
    float i = 10.3f;//Memory = 4 bytes
    short j =(short)  i; //Memory = 2 bytes
    System.out.println(j);
   }
}
```

Output:

```
10
```

Example 4:

```java
public class A {
   public static void main(String args[]) {
    float i = 10.3f;//Memory = 4 bytes
    int   j =(int)  i; //Memory = 4 bytes
    System.out.println(j);
   }
}
```

Output:

```
10
```

Example 5:

```
public class A {
   public static void main(String args[]) {
    float i = 10.3f;//Memory = 4 bytes
    long  j = (long) i; //Memory = 8 bytes
    System.out.println(j);
   }
}
```

Output:

```
10
```

Type casting :

Note: In the below example character value is converted to Unicode value

```
int i = 'a';
System.out.println(i);
```

Output:

```
97
```

Download and install JDK and eclipse:

Creating Java Project in Eclipse

- Option 1: control + n and then type java in the wizards and then select java project, then click on next, then give project name, Under JRE select second radio button and click on finish
- Option 2: Go to file>>new>>others>> then type java in the wizards and then select java project, then click on next, then give project name, Under JRE select second radio button and click on finish

- Option 3: Right click in project explorer >>Select new>> others>> then type java in the wizards and then select java project, then click on next, then give project name, Under JRE select second radio button and click on finish

Creating Java Class in your project

- Go to SRC folder of your project>>Right click on src folder go to new>>select class>>Then give your class a name and click on finish
- Select your java project>>Press control + n>>type class>> select class and give a name and click on finish

Explore Important shortcuts in eclipse

- Shortcut to create main method(): type "main" in lower case and then press control + space bar then enter
- shortcut to create System.out.println(): type syso then control + space bar
- Code formatting in eclipse: control + shift + f
- Shortcut to get eclipse suggestions: Control + 1
- Short cut to delete a line in eclipse: Control + D

Rules to design and develop methods in eclipse:

Rule 1: Always program execution begins with opening bracket of main method

Rule 2: Whenever method calling statement executes, program control will be transferred to the matching method

Rule 3: Whenever a user defined method closing bracket runs then the control will be transferred back to the calling statement

Rule 4: When the closing bracket of main method runs then the complete program execution would stop

Example 1:

```
package app_java_1;


public class A {

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

		System.out.println(100);//(2)

		A a1 = new A();//(3)

		a1.test();//(4) (7)

	}//(8)
```

```
        public void test() {//(5)
                System.out.println("From test");//(6)
        }//(7)
}
```

Output:

```
100
From test
```

Example 2:

```
package app_java_1;

public class A {

        public static void main(String[] args) {//Rule 1: Starts Here (1)
                A a1 = new A();//(2)
                a1.test();//(3) (7) Rule 2
                System.out.println(1000);//(8)
                a1.test();//Rule 2 (9)
        }//Rule 4 STOPS here

        public void test() {//(4)(10)
                System.out.println(500);//(5)(11)
        }//(6)(12) Rule 3

}
```

Output:

```
500
1000
```

```
500
```

Example 3:

```
package app_java_1;


public class A {


	public static void main(String[] args) {//(1) STARTS HERE, RULE 1
		A a1 = new A();//No Rule
		a1.test1();//Rule 2
	}//Rule 4 And program STOPS HERE


	public void test1() {
		A a2 = new A();//No Rule
		a2.test2();//Rule 2
	}//Rule 3


	public void test2() {
		System.out.println(500);//500
	}//Rule 3


}
```

Output:

```
500
```

Example 4:

```
package app_java_1;


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


}
```

Output:

```
100
```

Note: If a method is returning value then ensure that it is not void

Example 5:

```
package app_java_1;

public class A {

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

```
	}



}
```

Output:

Error because method is void and hence cannot return any value

Example 6:

```
package app_java_1;

public class A {

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



}
```

Output:

```
Pankaj Sir Academy
```

Example 7:

```
package app_java_1;
```

```
public class A {


	public static void main(String[] args) {
		A a1 = new A();
		float i = a1.test();
		System.out.println(i);
	}
	public float test() {
		return 10.3f;
	}



}
```

Output:

```
10.3
```

Example 8:

```
package app_java_1;


public class A {


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

```
        return 10.3f;

    }



}
```

Output:

```
Pankaj Sir Academy

10.3
```

Example 9:

Note: If we write anything after return keyword then that line of code will never execute and hence will give us an error unreachable code as shown in the below example.

```
package app_java_1;


public class A {


    public static void main(String[] args) {

        A a1 = new A();

        String i = a1.test();

        System.out.println(i);

    }

    public String test() {

        return "Pankaj";

        System.out.println("Pankaj Sir Academy");//Unreachable code error

    }



}
```

Output:

error

Example 10:

Note: If a method is void then it cannot return any value. In void methods we can use only "return " keyword. Usage of return keyword inside void methods is optional

```
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(1000);

		return;

	}



}
```

Output:

```
1000
```

Example 11:

```
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(1000);
    }



}
```

Output:

error because of unreachable code.

Example 12:

```
package app_java_1;

public class A {

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



}
```

Output:

```
100
```

Example 13:

```java
package app_java_1;


public class A {


	public static void main(String[] args) {

		A a1 = new A();

		a1.test(100,'a',"Pankaj",true);

	}

	public void test(int i, char c, String s, boolean b) {

		System.out.println(i);

		System.out.println(c);

		System.out.println(s);

		System.out.println(b);

	}



}
```

Output:

```
100

a

Pankaj

true
```

Example 14:

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

                A a1 = new A();

                a1.test(100,200,300,400);

        }

        public void test(int... x) {

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

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

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

                System.out.println(x[3]);

        }



}
```

Output:

```
100

200

300

400
```

Example 15:

```
package app_java_1;


public class A {


        public static void main(String[] args) {

                A a1 = new A();

                a1.test(100,'a',"pankaj",true);

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


}
```

Output:

```
100
a
pankaj
true
```

Example 16:

```
package app_java_1;

public class A {
        static Object x = 10;
        static Object y = 10.3;
        static Object z = true;
        public static void main(String[] args) {
                System.out.println(A.x);
                System.out.println(A.y);
                System.out.println(A.z);
        }
}
```

Output:

```
10

10.3

true
```

Definition: Methods helps us to break our programs into reusable modules

Example :

```
package app_java_1;


public class A {


        public static void main(String[] args) {


                A.test(100,200,300,400);

                A.test(500,600,700,800);

                A.test(100,200,300,400);

                A.test(500,600,700,800);

                A.test(100,200,300,400);

                A.test(500,600,700,800);

        }

        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]);

        }
}
```

Output:

Constructors in Java

- Constructors should have same name as that of class
- Whenever an object is created constructor would be called
- Constructors are internally void. It means constructors can never return any value
- We can create more than one constructor in the same class but ensure they have different number of arguments or different type of arguments. It is called as constructor overloading

Example 1:

```
package app_java_2;


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 3:

```java
package app_java_2;

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

	public static void main(String[] args) {
		A a1 = new A(100);
		A a2 = new A(200);
		A a3 = new A(300);
	}

}
```

Output:

```
100
200
300
```

Example 3:

```java
package app_java_one;
public class A {
	A(){
		System.out.println("From constructor");
		return 100;
	}
	public static void main(String[] args) {
		A a1 = new A();
	}

}
```

Output:

Error, because constructors are void and hence it can never return any value

Example 4:

```
package app_java_one;
public class A {
	A(){
		System.out.println("From constructor");
		return;
	}
	public static void main(String[] args) {
		A a1 = new A();
	}

}
```

Output:

```
From constructor
```

Example 5:

Note: If you void keyword while creating a constructor then it will be treated as method. In the below program when object is created method "void A()" will not be called

```
package app_java_one;
public class A {
        void A(){
                System.out.println("From constructor");
                return;
        }
        public static void main(String[] args) {
                A a1 = new A();
        }

}
```

Output:

```
Will compile and run but will print nothing
```

Example 6:

```
package app_java_one;
public class A {
        void A(){
                System.out.println("From constructor");
                return;
        }
        public static void main(String[] args) {
                A a1 = new A();
                a1.A();
        }

}
```

Output:

```
From constructor
```

Example 7:

```
package app_java_one;

public class A {

	A(){//No Of Args = 0

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

	}

	A(int i){//No Of Args = 1

		System.out.println(i);

	}

	A(int i,int j){//No 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:

```
From Constructor A

100

500

1000
```

Example 8:

```
package contructors_example;
```

```
public class A {


	A(int i){

		System.out.println(i);

	}

	A(char j){

		System.out.println(j);

	}

	public static void main(String[] args) {

		A a1 = new A(100);

		A a2 = new A('a');


	}

}
```

Output:

```
100

a
```

IIB - Instance Initialization block

- Whenever an object is created IIB will be called
- The main purpose of IIB is to initialize all non static variables in one place, so that it creates better readability of the code
- IIB initializes all the variables during runtime

Example 1:

```
package contructors_example;


public class A {

	{

		System.out.println("From IIB 3");
	}
	{
		System.out.println("From IIB 1");
	}
	{
		System.out.println("From IIB 2");
	}
	public static void main(String[] args) {

		A a1 = new A();

	}
}
```

Output:

```
From IIB 3
From IIB 1
From IIB 2
```

Example 2:

Note: Always IIB runs first and then the constructor

```
package contructors_example;

public class A {
	A()
	{
		System.out.println("From Contructor");
	}
	{
		System.out.println("From IIB ");
	}

	public static void main(String[] args) {

		A a1 = new A();

	}
}
```

Output:

```
From IIB
From Contructor
```

Example 3:

```
package contructors_example;

public class A {
	{
		System.out.println("From IIB z");
	}
	A()
	{
		System.out.println("From Contructor");
	}
	{
		System.out.println("From IIB g");
	}
	{
		System.out.println("From IIB h");
	}

	public static void main(String[] args) {

		A a1 = new A();

	}
}
```

Output:

```
From IIB z
From IIB g
From IIB h
From Contructor
```

Example 4:

Note: When an object with argument is created still it will be calling IIB

```
package contructors_example;

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

```
                System.out.println("From IIB g");
        }
        {
                System.out.println("From IIB h");
        }

        public static void main(String[] args) {

                A a1 = new A(100);


        }
}
```

Output:

```
From IIB z
From IIB g
From IIB h
100
```

Example 5:

```
package contructors_example;

public class A {
        int i,j,k;

        {
                i = 100;
                j = 200;
                k = 300;
```

```
	}



	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:

```
100

200

300
```

Example 6:

Note: static variables can also be initialized in IIB, but it is not recommended!!

```
package contructors_example;


public class A {

	static int i,j,k;


	{

		i = 100;

		j = 200;

		k = 300;
	}



	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:

```
100

200

300
```

Static Initialization Block (SIB):

- **Always SIB runs first before main method**
- **SIB runs automatically, we need not call it**
- **The main purpose of SIB is to initialize all static variables in One place**

Example 1:

```
package contructors_example;


public class A {


	static

	{

		System.out.println("SIB");
	}

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

Output:

```
SIB
main
```

Example 2:

```
package contructors_example;

public class A {

	static
	{
		System.out.println("SIB");
	}

}
```

Output:

Error, because there is not main method present in the above program

Example 3:

Write a java code to call main method twice?

```
package contructors_example;

public class A {
        static {
                A.main(null);
                A.main(null);
                A.main(null);
        }

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

}
```

Output:

```
From main
From main
From main
From main
```

Example 4:

```
package contructors_example;

public class A {

        static int i,j,k;

        static
        {
                i = 10;
                j = 20;
```

```
		k = 30;
	}

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

}
```

Output:

```
10
20
30
```

All Mixed IIB, SIB & Constructor Example

Note:

- **Always SIB runs first then main method will run and then if object is created IIB will run and finally constructor**

Example 1:

```
package contructors_example;

public class A {
	A()
	{
		System.out.println(5);
	}
	static
	{
			System.out.println(100);
		}
		{
			System.out.println(21);
		}

		public static void main(String[] args) {
			System.out.println(31);
		}

}
```

Output:

```
100
31
```

Example 3:

```
package contructors_example;

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

```
		System.out.println(21);
	}

	public static void main(String[] args) {
		System.out.println(31);
		A a1 = new A();
	}

}
```

Output:

```
100
31
21
5
```

this keyword in java

- It is a special reference variables that holds current objects address and it is created automatically by java compiler
- Non static member of the class can be accessed using this keyword

Example 1:

```
package app_this_keyword;

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_this_keyword.A@15db9742
app_this_keyword.A@15db9742
```

Example 2:

```
package app_this_keyword;

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
```

Example 3:

```
package app_this_keyword;
```

```
public class A {


	public static void main(String[] args) {

		A a1 = new A();

		System.out.println("Object Address 1 : "+a1);

		a1.test();

		A a2 = new A();

		System.out.println("Object Address 2 : "+a2);

		a2.test();

	}

	public void test() {

		System.out.println("This Keyword address : "+this);

	}

}
```

Output:

```
Object Address 1 : app_this_keyword.A@15db9742

This Keyword address : app_this_keyword.A@15db9742

Object Address 2 : app_this_keyword.A@6d06d69c

This Keyword address : app_this_keyword.A@6d06d69c
```

Limitations of this keyword

- this keyword cannot be used inside static methods

Example 1:

```
package app_this_keyword;


public class A {


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

```
                A.test();


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

Output:

Error because this keyword cannot be used inside static methods

Example 2:

```
package app_this_keyword;


public class A {


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


        }


}
```

Output:

Error

Note: Using this keyword we can access static members as well. But it is not recommended!!

Example 3:

```
package app_this_keyword;
```

```
public class A {

        static int i = 10;

        int j = 200;

        public static void main(String[] args) {

                A a1 = new A();

                a1.test();

        }

        public void test() {

                System.out.println(this.i);

                System.out.println(this.j);

        }

}
```

Output:

```
10

200
```

Example 4:

```
package app_this_keyword;


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 5:

```
package app_this_keyword;


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
```

Other benefits of this keyword

- **Using this keyword we can call constructor of a class, but this call should happen from another constructor of same class and it should be the first statement inside another constructor**

Example 1:

```
package app_this_keyword;


public class A {

	A(){

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

	}

	A(int i){

		this();

	}


	public static void main(String[] args) {

			A a1 = new A(100);

	}



}
```

Example 2:

```
package app_this_keyword;


public class A {

	A(){
```

```
                this(200);
        }
        A(int i){
                System.out.println(i);
        }

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



}
```

Output:

```
200
```

Example 3:

```
package app_this_keyword;

public class A {
        A(){
                System.out.println("A");
                this(200);

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

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

			A a1 = new A();

	}



}
```

Output:

Error because this keyword is not first statement inside another constructor

Example 4:

```
package app_this_keyword;


public class A {

	A(){

		this(200);

		System.out.println("A");




	}

	A(int i){

		System.out.println(i);

	}


	public static void main(String[] args) {

			A a1 = new A();

	}
}
```

Output:

```
200

A
```

Example 5:

```
package app_this_keyword;


public class A {

	int i;


	public static void main(String[] args) {

		A a1 = new A();

		a1.test();


	}

	public void test() {

		int i = 50;

		this.i = i;

		System.out.println(this.i);

	}



}
```

Output:

```
50
```

Example 6:

```
package app_this_keyword;
```

```
public class A {

        int i= 600;

        public static void main(String[] args) {

                A a1 = new A();

                a1.test();



        }

        public void test() {

                System.out.println(i);//It gets added automatically this.i

        }

}
```

Output:

```
600
```

Example 7:

```
package app_this_keyword;
public class A {
        int i= 600;
        public static void main(String[] args) {
                A a1 = new A();
                a1.test();

        }
        public static void test() {
                System.out.println(i);//this keyword does not gets added automatically
        }
}
```

Output:

```
Error
```

Inheritance In Java:

- Here we inherit non static members of parent class into child class object
- With inheritance we are able to re-use non static members of parent class into child class object
- In java at class level multiple inheritance is not allowed because multiple inheritance results in complex designing of the software

Example 1:

```
package app_inheritance;

public class A {//Parent, Super
```

```
        int i = 10;
}

package app_inheritance;

public class B extends A{ //Child

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

}
```

Output:

```
10
```

Example 2:

```
package app_inheritance;

public class A {

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

}
package app_inheritance;

public class B extends A{

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

}
```

Output:

```
From test method
10
```

Example 3:

```
package app_inheritance_example_1;

public class A {

        public void test1() {
                System.out.println("From test 1");
        }
}
package app_inheritance_example_1;

public class B extends A{

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


}
package app_inheritance_example_1;

public class C extends B{
	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 app_inheritance_example_1;

public class A {

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

}
package app_inheritance_example_1;

public class B {

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


}
package app_inheritance_example_1;
```

```
public class C extends A,B{


}
```