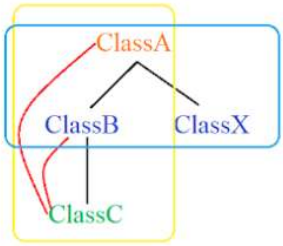
Types Of Inheritance:

1) Single Inheritance ➡ One level of Inheritance [OneParent-One Child]
Every Java class by default exhibit single inheritance, because Object class is the parent Class for all the Classes by default.

2) Multi-Level Inheritance ➡ In Multilevel inheritance a child class object should be able to access both parent class methods and grandparent class methods.

3) Heirarchal Inheritance ➡ Sharing the properties of one class into multiple child classes

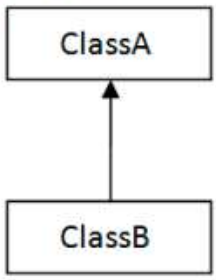


4) Hybrid Inheritance ➡ It is a Combination of '2' or more Inheritances

5) Multiple Inheritance ➡ Java does not support multiple inheritance through **Classes** because of ambiguity problem. But we can achieve this multiple inheritance in Java by using **Interfaces**. ***(One child multiple parents)***

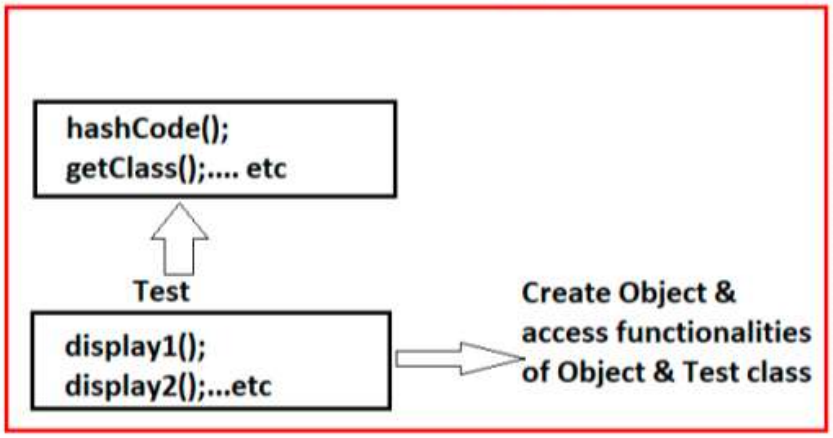
Single Inheritance

- Every java class by default inherits from "java.lang.Object" class.
- By this extension only, every user object gets the behavior of real Object.
- Hence every java class exhibits by default "Single Inheritance".
- Single Inheritance enables a derived class(Sub class) to inherit properties and behavior from a single parent class



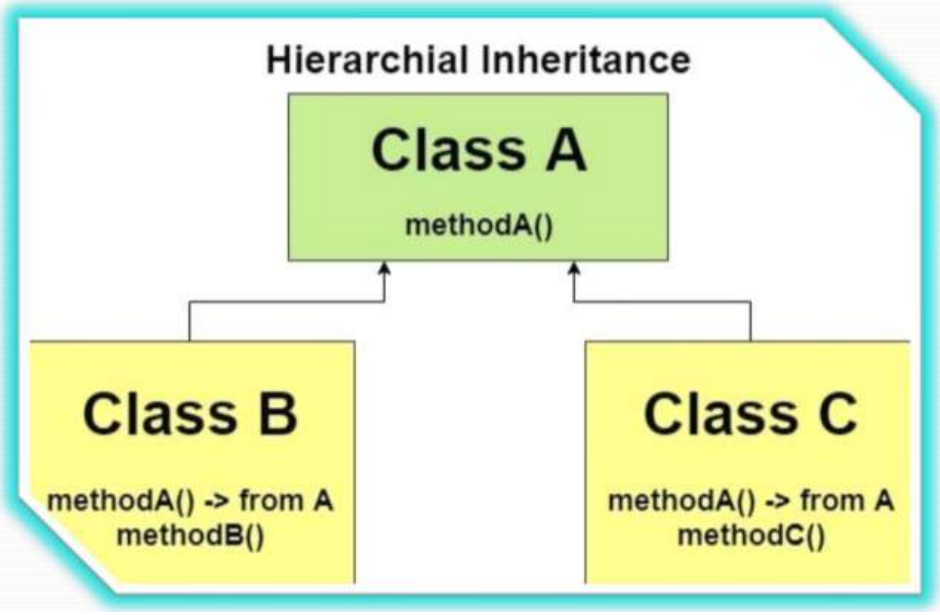
Multi-Level Inheritance

- Accessing the functionality of objects in more than one level is called "Multi-Level Inheritance".
- Child class accessing the functionality of grand parent.



Hierarchal Inheritance

- Sharing the properties of object to multiple child objects is called "Hierarchal Inheritance".



Understanding 'Super'

- It is a keyword which is predefined and non-static variable.
- It is used to access the complete functionality of the parent class.
- It must be used in non-static context.

this vs super

this	super
It is Keyword	It is a Keyword
Non-static variable	Non-static variable
Used to access current class object functionality	Used to access the parent object functionality from child class
Must be used in non-static context.	Must be used in non-static context.
It holds object address	It doesn't hold any object address

super() vs this()

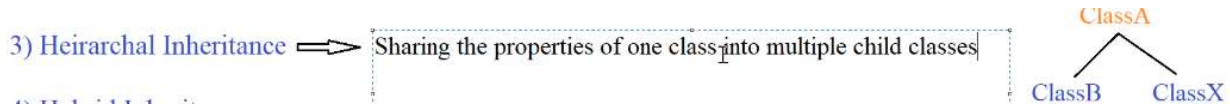
super()	this()
Used to invoke parent class constructor	Used to invoke current class constructor
Must be used only inside the child class constructor.	Must be used inside another constructor of same class.
It must be first statement	It must be first statement

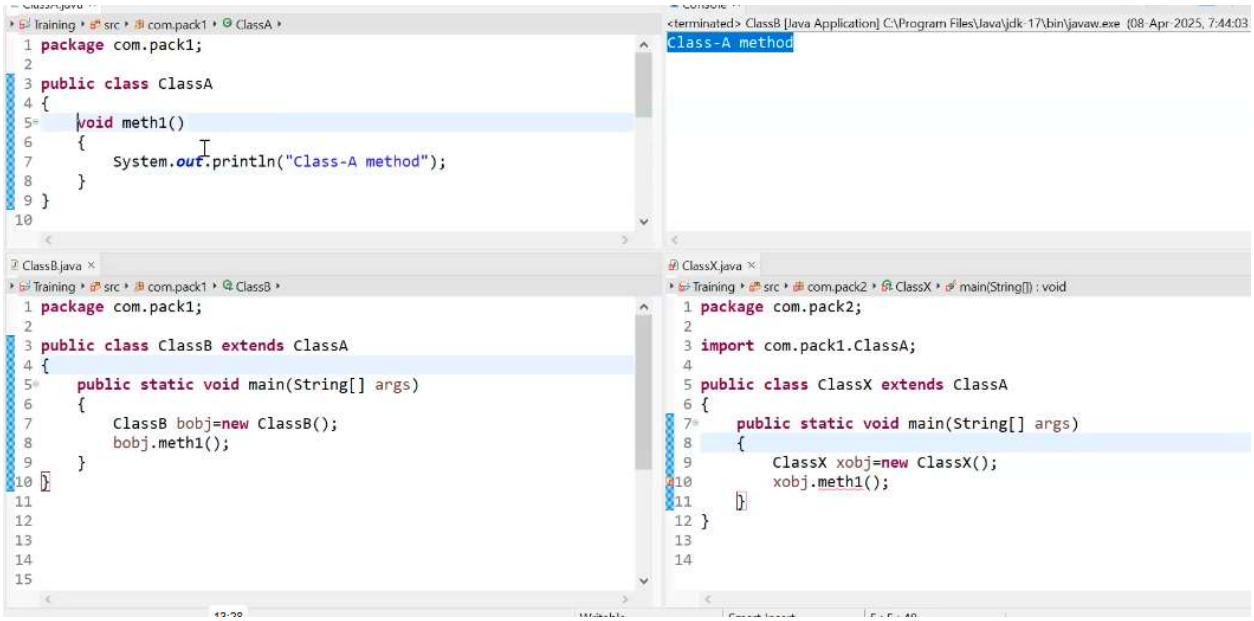
super() , this() Vs super, this

super(),this()	super, this
These are constructors calls.	These are keywords
We can use these to invoke super class & current constructors directly	We can use these to refers parent class and current class instance members.
We should use only inside constructors as first line, if we are using outside of constructor we will get compile time error	We can use anywhere (i.e., instance area) except static area , other wise we will get compile time error .

super	super()
1) It is a Keyword 2) It is used to call PARENT class variables or methods 3) We cannot use 'super' keyword inside a static area.	1) It is a Constructor call 2) It is used to call PARENT class constructors. 3) We need to use **super()** ONLY inside a constructor, that too as a FIRST statement. 4) In every Java constructor by default compiler is going to provide **super()** as its a first statement.

Hierarchical inheritance





```java
package com.pack1;

public class ClassA
{
    public void meth1()
    {
        System.out.println("Class-A method");
    }
}
```

```
Class-A method
```

ClassB.java

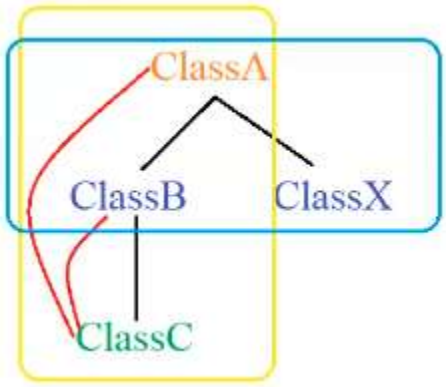
```java
package com.pack1;

public class ClassB extends ClassA
{
    public static void main(String[] args)
    {
        ClassB bobj=new ClassB();
        bobj.meth1();
    }
}
```

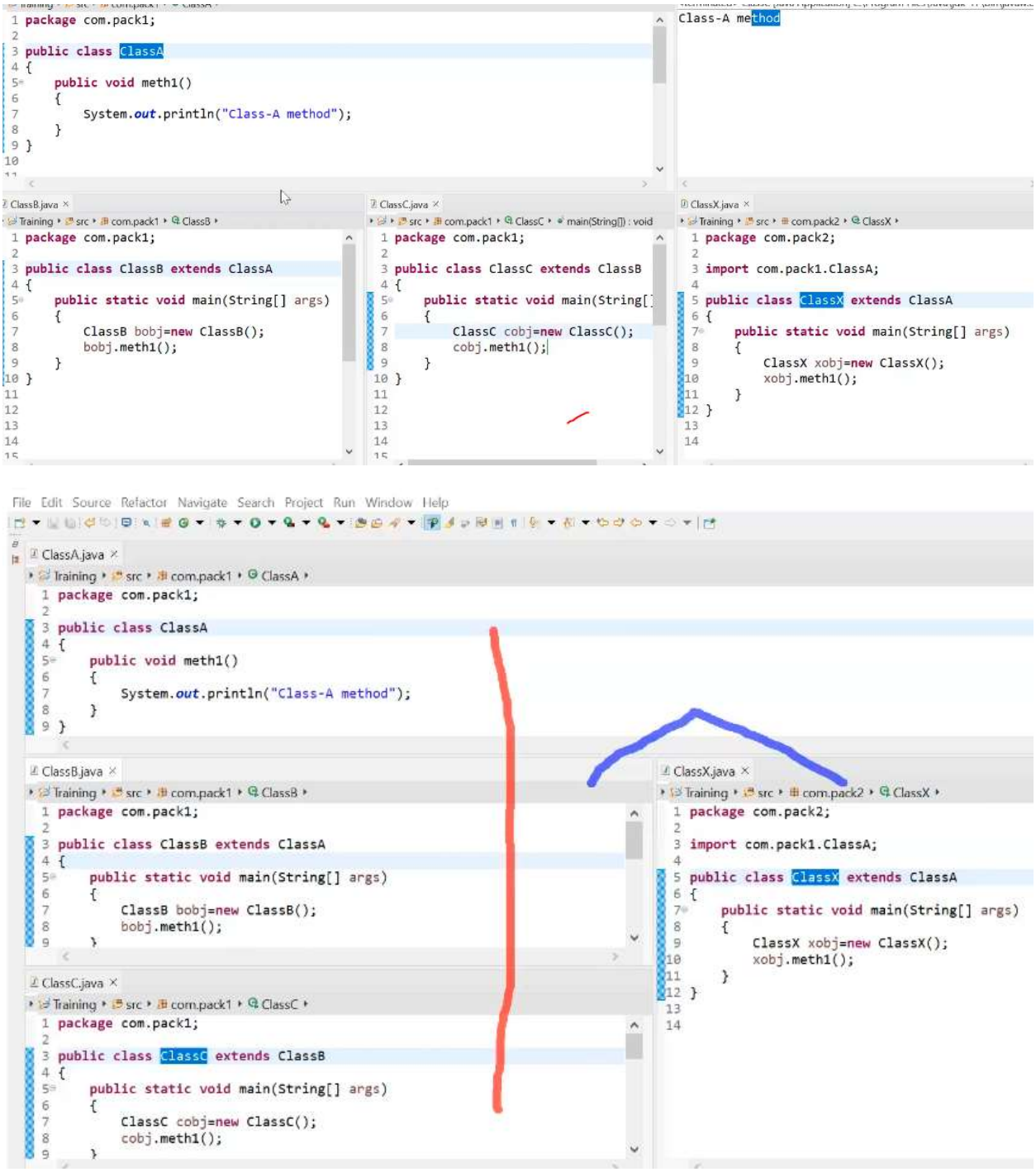
ClassX.java

```java
package com.pack2;

import com.pack1.ClassA;

public class ClassX extends ClassA
{
    public static void main(String[] args)
    {
        ClassX xobj=new ClassX();
        xobj.meth1();
    }
}
```

Hybrid inheritance

4) Hybrid Inheritance ⟹ It is a Combination of '2' or more Inheritances





Java does not support multiple inheritance through classes but by using interfaces we can achieve.

```java
package com.pack1;

public class ClassA
{
    public void meth1()
    {
        System.out.println("Class-A method");
    }
}
```

```java
package com.pack1;


public class ClassC extends ClassA, ClassB
{
    public static void main(String[] args)
    {
        ClassC cobj=new ClassC();
        cobj.meth1();
    }
}
```

ClassB.java

Training ▸ src ▸ com.pack1 ▸ ClassB ▸ meth1() : void

```java
package com.pack1;

public class ClassB
{
    public void meth1()
    {
        System.out.println("Class-B method");
    }
}
```

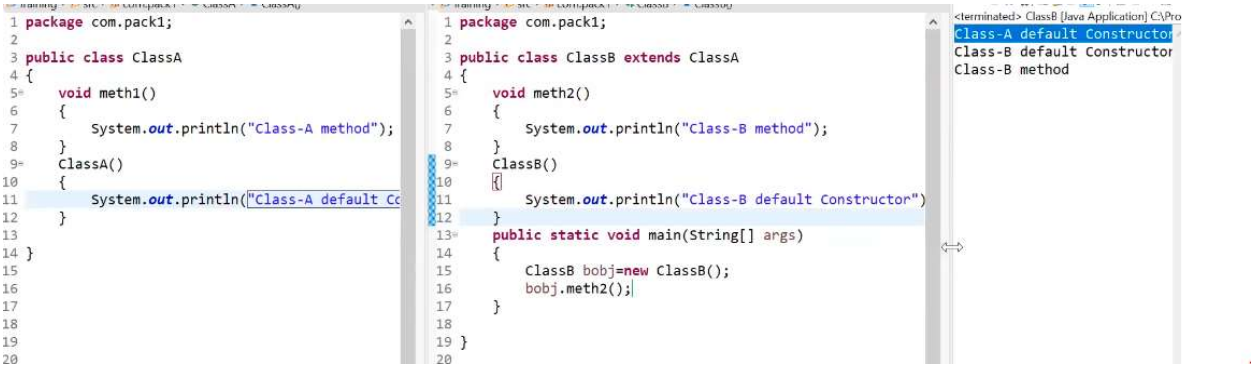
Ambiguity in meth1()

5) Multiple Inheritance ⇨ Java does not support multiple inheritance through **Classes** because of ambiguity problem. But we can achieve this multiple inheritance in Java by using **Interfaces**. ***(One child multiple parents)***

Note

Constructors do not participate in inheritance

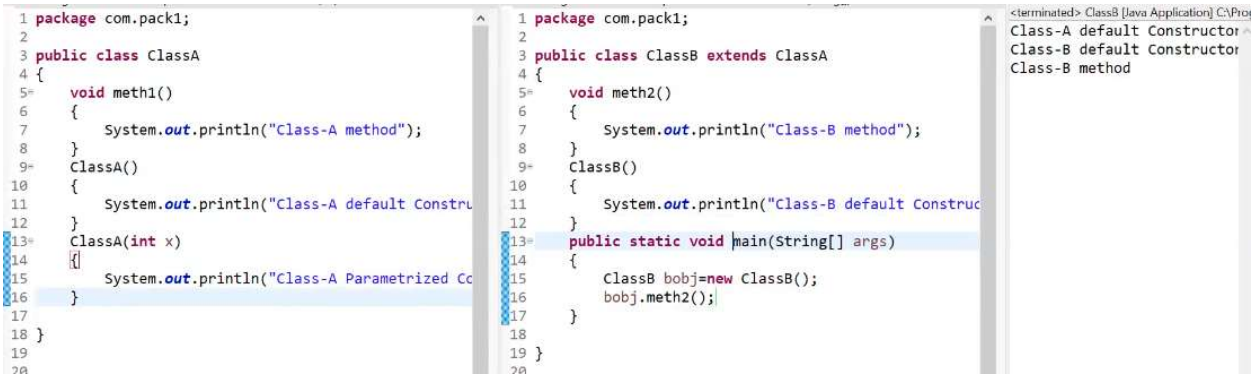
super	super()
1) It is a Keyword	1) It is a Constructor call
2) It is used to call PARENT class variables or methods	2) It is used to call PARENT class constructors.
3) We cannot use 'super' keyword inside a static area.	3) We need to use **super()** ONLY inside a constructor, that too as a FIRST statement.
	4) In every Java constructor by default compiler is going to provide **super()** as its a first statement.



```
package com.pack1;

public class ClassA
{
    void meth1()
    {
        System.out.println("Class-A method");
    }
    ClassA()
    {
        System.out.println("Class-A default Cc
    }

}
```

```
package com.pack1;

public class ClassB extends ClassA
{
    void meth2()
    {
        System.out.println("Class-B method");
    }
    ClassB()
    {
        System.out.println("Class-B default Constructor")
    }
    public static void main(String[] args)
    {
        ClassB bobj=new ClassB();
        bobj.meth2();
    }

}
```

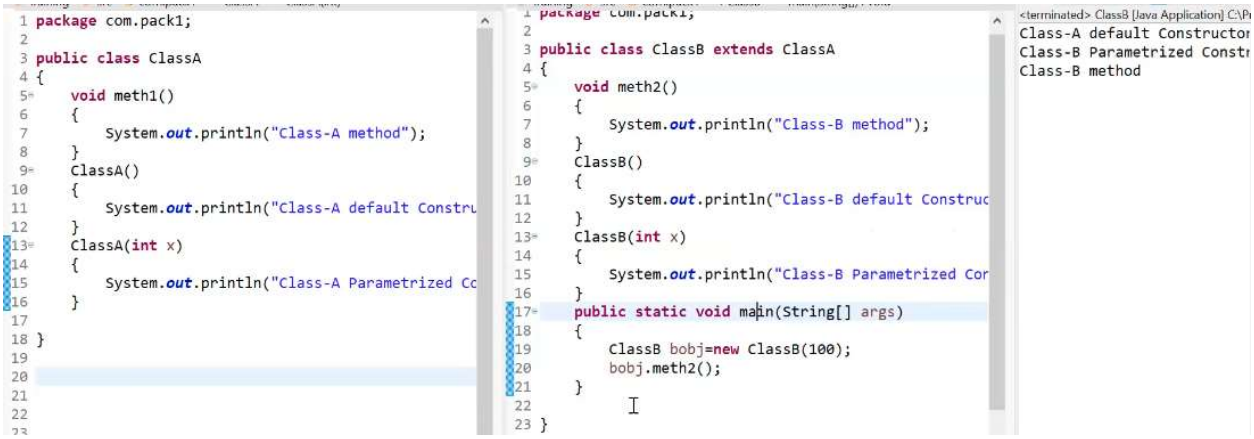
```
<terminated> ClassB [Java Application] C:\Pro
Class-A default Constructor
Class-B default Constructor
Class-B method
```



```
package com.pack1;

public class ClassA
{
    void meth1()
    {
        System.out.println("Class-A method");
    }
    ClassA()
    {
        System.out.println("Class-A default Constru
    }
    ClassA(int x)
    {
        System.out.println("Class-A Parametrized Cc
    }

}
```

```
package com.pack1;

public class ClassB extends ClassA
{
    void meth2()
    {
        System.out.println("Class-B method");
    }
    ClassB()
    {
        System.out.println("Class-B default Construc
    }
    public static void main(String[] args)
    {
        ClassB bobj=new ClassB();
        bobj.meth2();
    }

}
```

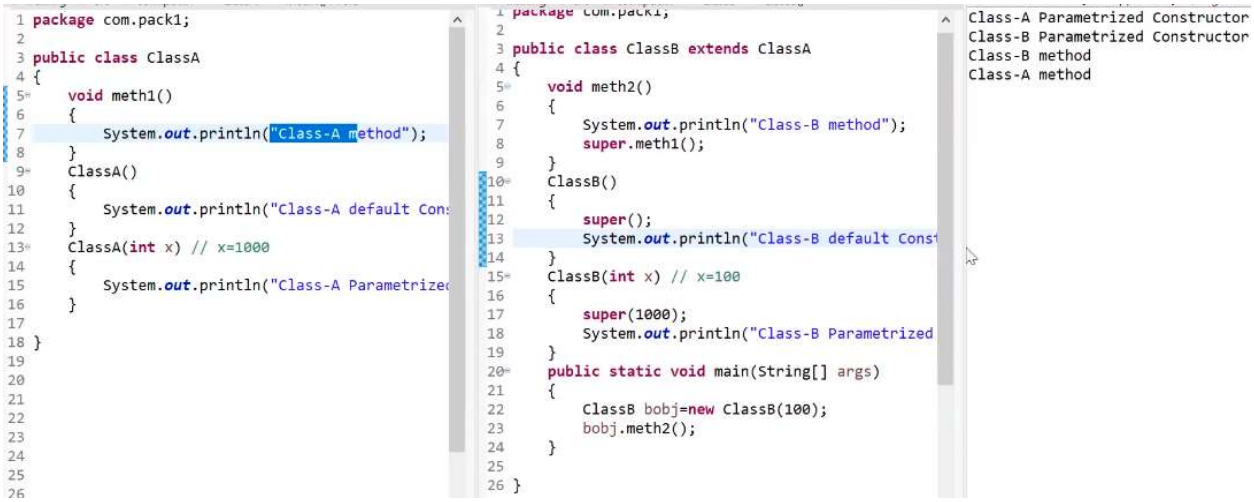
```
<terminated> ClassB [Java Application] C:\Pro
Class-A default Constructor
Class-B default Constructor
Class-B method
```



```
package com.pack1;

public class ClassA
{
    void meth1()
    {
        System.out.println("Class-A method");
    }
    ClassA()
    {
        System.out.println("Class-A default Constru
    }
    ClassA(int x)
    {
        System.out.println("Class-A Parametrized Cc
    }

}
```

```
package com.pack1;

public class ClassB extends ClassA
{
    void meth2()
    {
        System.out.println("Class-B method");
    }
    ClassB()
    {
        System.out.println("Class-B default Construc
    }
    ClassB(int x)
    {
        System.out.println("Class-B Parametrized Cor
    }
    public static void main(String[] args)
    {
        ClassB bobj=new ClassB(100);
        bobj.meth2();
    }
}
```

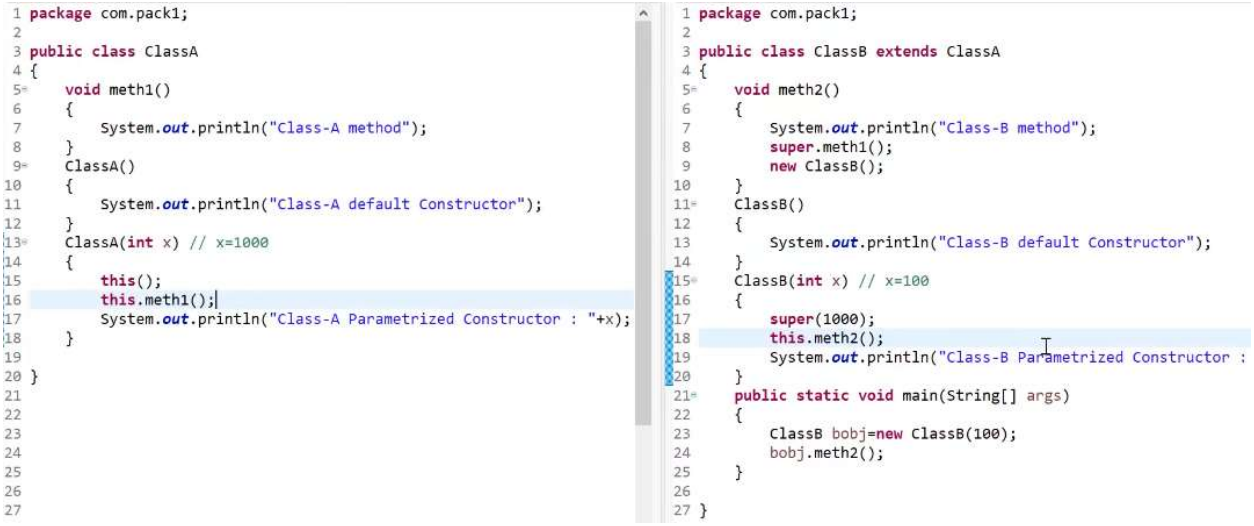
```
<terminated> ClassB [Java Application] C:\Pr
Class-A default Constructor
Class-B Parametrized Constr
Class-B method
```



```
package com.pack1;

public class ClassA
{
    void meth1()
    {
        System.out.println("Class-A method");
    }
    ClassA()
    {
        System.out.println("Class-A default Cons
    }
    ClassA(int x) // x=1000
    {
        System.out.println("Class-A Parametrize
    }

}
```

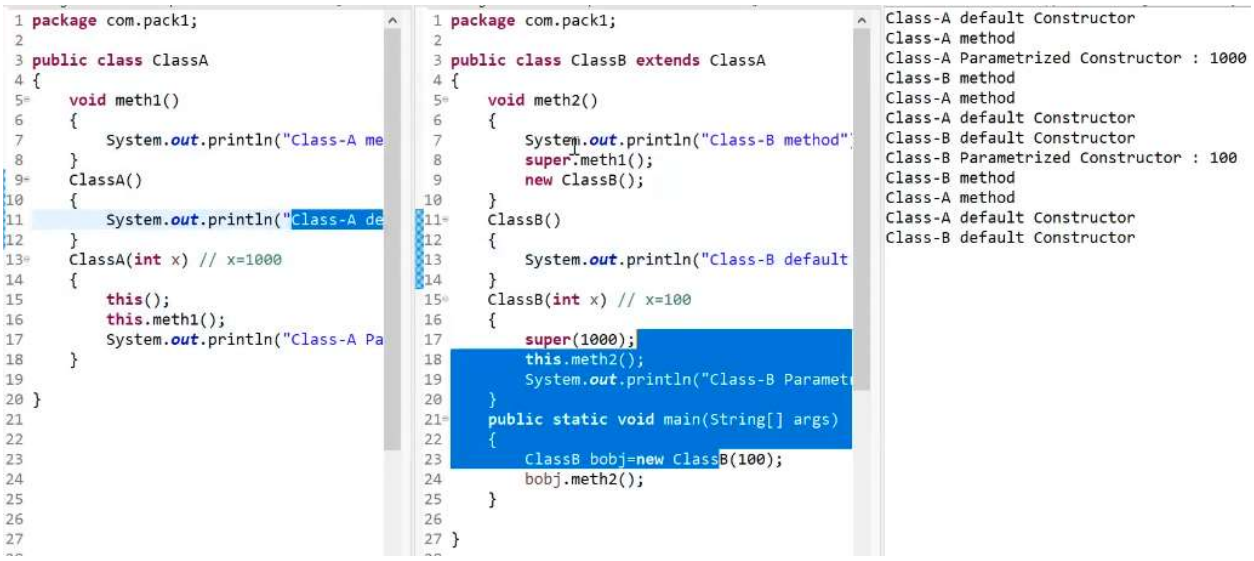
```
package com.pack1;

public class ClassB extends ClassA
{
    void meth2()
    {
        System.out.println("Class-B method");
        super.meth1();
    }
    ClassB()
    {
        super();
        System.out.println("Class-B default Const
    }
    ClassB(int x) // x=100
    {
        super(1000);
        System.out.println("Class-B Parametrized
    }
    public static void main(String[] args)
    {
        ClassB bobj=new ClassB(100);
        bobj.meth2();
    }

}
```

```
Class-A Parametrized Constructor
Class-B Parametrized Constructor
Class-B method
Class-A method
```



```
package com.pack1;

public class ClassA
{
    void meth1()
    {
        System.out.println("Class-A method");
    }
    ClassA()
    {
        System.out.println("Class-A default Constructor");
    }
    ClassA(int x) // x=1000
    {
        this();
        this.meth1();
        System.out.println("Class-A Parametrized Constructor : "+x);
    }

}
```

```
package com.pack1;

public class ClassB extends ClassA
{
    void meth2()
    {
        System.out.println("Class-B method");
        super.meth1();
        new ClassB();
    }
    ClassB()
    {
        System.out.println("Class-B default Constructor");
    }
    ClassB(int x) // x=100
    {
        super(1000);
        this.meth2();
        System.out.println("Class-B Parametrized Constructor :
    }
    public static void main(String[] args)
    {
        ClassB bobj=new ClassB(100);
        bobj.meth2();
    }

}
```



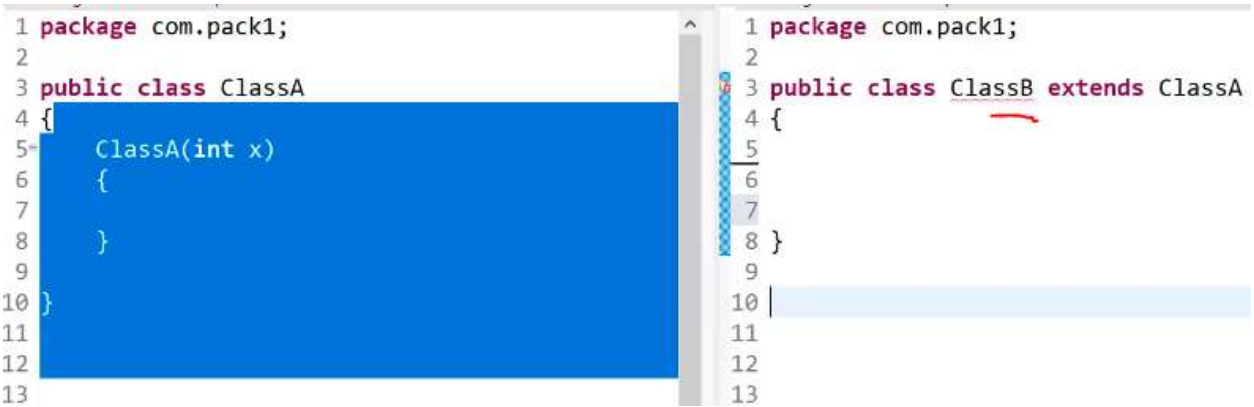
```
1 package com.pack1;
2
3 public class ClassA
4 {
5     void meth1()
6     {
7         System.out.println("Class-A me
8     }
9     ClassA()
10    {
11        System.out.println("Class-A de
12    }
13    ClassA(int x) // x=1000
14    {
15        this();
16        this.meth1();
17        System.out.println("Class-A Pa
18    }
19
20 }
```

```
1 package com.pack1;
2
3 public class ClassB extends ClassA
4 {
5     void meth2()
6     {
7         System.out.println("Class-B method"
8         super.meth1();
9         new ClassB();
10    }
11    ClassB()
12    {
13        System.out.println("Class-B default
14    }
15    ClassB(int x) // x=100
16    {
17        super(1000);
18        this.meth2();
19        System.out.println("Class-B Paramet
20    }
21    public static void main(String[] args)
22    {
23        ClassB bobj=new ClassB(100);
24        bobj.meth2();
25    }
26
27 }
```

```
Class-A default Constructor
Class-A method
Class-A Parametrized Constructor : 1000
Class-B method
Class-A method
Class-A default Constructor
Class-B default Constructor
Class-B Parametrized Constructor : 100
Class-B method
Class-A method
Class-A default Constructor
Class-B default Constructor
```

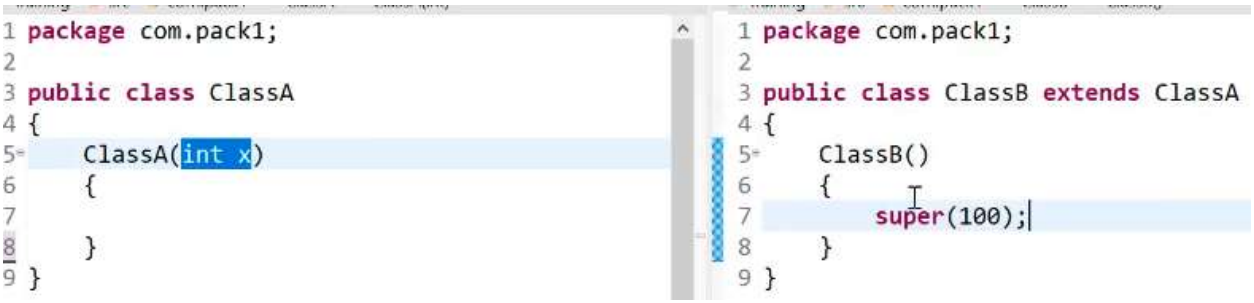
In inheritance if child class and parents' class does not have any constructor is no problem, if parent class is having an

Default constructor is no problem but if parent class is having a parameter constructor and in child class, we are not calling parameterized constructor(super(parameter)), then 100% parent class should have a default constructor.



```
1 package com.pack1;
2
3 public class ClassA
4 {
5     ClassA(int x)
6     {
7
8     }
9
10 }
11
12
13
```

```
1 package com.pack1;
2
3 public class ClassB extends ClassA
4 {
5
6
7
8 }
9
10
11
12
13
```



```java
package com.pack1;

public class ClassA
{
    ClassA(int x)
    {

    }
}
```

```java
package com.pack1;

public class ClassB extends ClassA
{
    ClassB()
    {
        super(100);
    }
}
```