3.4. Inheritance

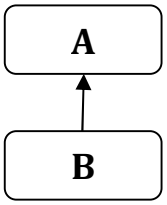
- **Inheritance is the process of writing new class by inheriting commonly used state and behavior of existing class.**
- **Existing Class is called as Super class or Base class or Parent Class.**
- **Newly defined Class is called as Sub class or Derived class or Child class.**
- **Inheritance is an important property or features of OOPS.**
- **Main goal of inheritance is code reusability i.e. Inheritance can be used to re-use the properties and operations of the existing class in the newly defined class.**

Types of Inheritance

1) **Simple Inheritance**
2) **Multilevel Inheritance**
3) **Hierarchical Inheritance**
4) **Multiple Inheritance**
5) **Hybrid Inheritance**

Simple Inheritance

- **In Simple inheritance,**
  - **There will be only one super class and one sub class.**

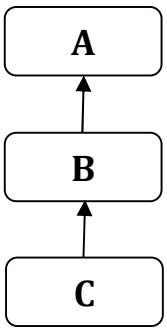


Ex:

```
class A{ ...}

class B extends A{...}
```

Multilevel Inheritance

- **In Multilevel inheritance,**
  - **One super class can have only one direct sub class and many indirect sub classes.**
  - **One sub class can have only one direct super class and many indirect super classes.**

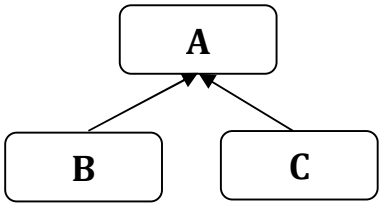


Ex:

```
class A{ ...}

class B extends A{...}

class C extends B{...}
```

Hierarchical Inheritance

- In Hierarchical inheritance,
  - One super class can have many direct sub classes.
  - One sub class can have only one direct super class.

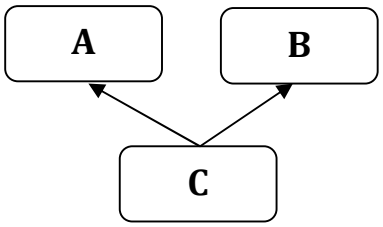


Ex:

```
class A{ ...}

class B extends A{...}

class C extends A{...}
```

Multiple Inheritance

- In Multiple inheritance,
  - One sub class can have many direct super classes.
  - One super class can have only one direct sub class.



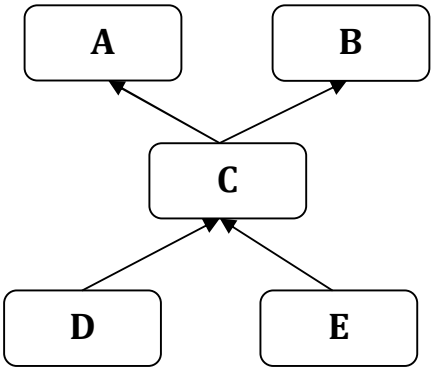
Ex:

```
class A{ ...}

class B {...}

class C extends A,B{...}     // INVALID
```

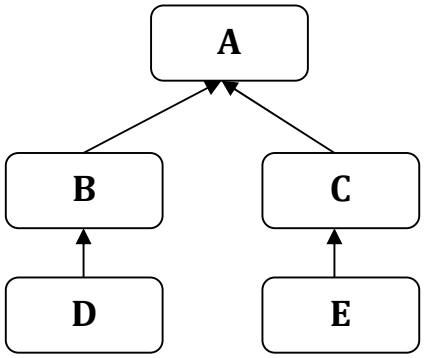
- Java does not support Multiple inheritance with classes.
- Java supports Multiple inheritance with interfaces.

Hybrid Inheritance

- Hybrid inheritance is the combination of two or more inheritance types.



Combination of Multiple inheritance, Hierarchical inheritance and Multilevel inheritance. ... Invalid



Combination of Hierarchical inheritance and Multilevel inheritance.

... Valid

Lab387.java

```
class Hai{
static{
System.out.println("Hai - S.B");
}
}

class Hello extends Hai{
static {
System.out.println("Hello - S.B");
}
}

class Lab387{
static public void main(String args[]){
Hai hai=new Hai();
}
}
```

Lab388.java

```
class Hai{
static{
System.out.println("Hai - S.B");
}
}

class Hello extends Hai{
static {
System.out.println("Hello - S.B");
}
}

class Lab388{
static public void main(String args[]){
Hello hello=new Hello();
}
}
```

Lab389.java

```
class Hai{
int a=10;
}

class Hello{
int b=20;
}

class Lab389{
static public void main(String args[]){
Hai hai=new Hai();
System.out.println(hai.a);
System.out.println(hai.b); //

Hello hello=new Hello();
System.out.println(hello.a); //
System.out.println(hello.b);
}
}
```

Lab390.java

```
class Hai{
int a=10;
}

class Hello extends Hai{
int b=20;
}

class Lab390{
static public void main(String args[]){
Hai hai=new Hai();
System.out.println(hai.a);
System.out.println(hai.b);

Hello hello=new Hello();
System.out.println(hello.a);
System.out.println(hello.b);
}
}
```

Lab391.java

```
class Hai{
int a=10;
}

class Hello extends Hai{
int b=20;
}

class Lab391{
static public void main(String args[]){
Hai hai=new Hai();
System.out.println(hai.a);

Hello hello=new Hello();
System.out.println(hello.a);
System.out.println(hello.b);
}
}
```

Lab392.java

```
class Hai{
int a=10;
void m1(){
System.out.println("m1 - a = "+ a);
System.out.println("m1 - b = "+ b);
}
}
class Hello extends Hai{
int b=20;
void m2(){
System.out.println("m2  - a = "+ a);
System.out.println("m2  - b = "+ b);
}
}
class Lab392{
static public void main(String args[]){
Hello hello=new Hello();
hello.m1();
hello.m2();
}
}
```

Lab393.java

```
class Hai{
int a=10;
void m1(){
System.out.println("m1 - a = "+ a);
}
}

class Hello extends Hai{
int b=20;
void m2(){
System.out.println("m2  - a = "+ a);
System.out.println("m2  - b = "+ b);
}
}
```

Lab394.java

```
class A{
void m1(){
System.out.println("A - m1()");
}
}
class B extends A{
void m2(){
System.out.println("B - m2()");
}
}
class C extends B{
void m3(){
System.out.println("C - m3()");
}
}
```

```
class Lab393{
static public void main(String args[]){
Hello hello=new Hello();
hello.m1();
hello.m2();
}
}
```

```
class Lab394{
static public void main(String args[]){
C co = new C();
co.m1();
co.m2();
co.m3();
}
}
```

Lab395.java

```
class A{
void m1(){
System.out.println("A - m1()");
}
}
class B extends A{
void m2(){
System.out.println("B - m2()");
}
}
class C extends A{
void m3(){
System.out.println("C - m3()");
}
}

class Lab395{
static public void main(String args[]){
B bo = new B();
bo.m1();
bo.m2();
//bo.m3();

C co = new C();
co.m1();
//co.m2();
co.m3();

}
}
```

Lab396.java

```
class A{ }
class B { }
class C extends A,B{ }

class Lab396{
static public void main(String args[]){
System.out.println("Hello Guys");
}
}
```

Lab397.java

```
class Hai { }

class Hello extends Object,Hai { }

class Lab397{
static public void main(String args[]){
System.out.println("Hello Guys");
}
}
```

Lab398.java

```
class Hello extends Hello { }

class Lab398{
static public void main(String args[]){
System.out.println("Hello Guys");
}
}
```

Lab399.java

```
class Hello extends Hai { }

class Hai extends Hello { }

class Lab399{
static public void main(String args[]){
System.out.println("Hello Guys");
}
}
```

Lab400.java

```
final class Hai {   }

class Hello extends Hai {   }

class Lab400{
static public void main(String args[]){
System.out.println("Hello Guys");
}
}
```

3.4.1 Inheritance and Blocks

Lab401.java

```
class Hai {
int a=10;
{
System.out.println("Hai- I.B : a = "+a);
}
}
class Hello extends Hai {
int b=20;
{
System.out.println("Hello- I.B : a = "+a);
System.out.println("Hello- I.B : b = "+b);
}
}
class Lab401{
static public void main(String args[]){
Hello hello=new Hello();
}
}
```

Lab402.java

```
class Hai {
static int a=10;
static {
System.out.println("Hai- S.B : a = "+a);
}
}
class Hello extends Hai {
static int b=20;
static {
System.out.println("Hello- S.B : a = "+a);
System.out.println("Hello- S.B : b = "+b);
}
}
class Lab402{
static public void main(String args[]){
int x = Hello.b;
}
}
```

Lab403.java

```
class A{
{
System.out.println("A- I.B");
}
static{
System.out.println("A- S.B");
}
}
class B extends A{
{
System.out.println("B- I.B");
}
static{
System.out.println("B- S.B");
}
}
class C extends B{
{
System.out.println("C- I.B");
}
static{
System.out.println("C- S.B");
}
}

class Lab403{
static public void main(String args[]){
C co = new C();
}
}
```

3.4.2 Inheritance and Constructors

Lab404.java

```
class Hai{
Hai(){
System.out.println("Hai- 0 arg");
}
}

class Hello extends Hai{
Hello(){
System.out.println("Hello- 0 arg");
}
}

class Lab404{
static public void main(String args[]){
new Hello();
}
}
```

Lab405.java

```
class A{
A(){
System.out.println("A- 0 arg");
}
}

class B extends A{
B(){
System.out.println("B- 0 arg");
}
}

class C extends B{
C(){
System.out.println("C- 0 arg");
}
}

class Lab405{
static public void main(String args[]){
new C();
}
}
```

Lab406.java

```
class A{
int a=10;
static int b=20;
static{
System.out.println("A- S.B");
}
{
System.out.println("A- I.B");
}
A(){
System.out.println("A- 0 arg");
}
}
class B extends A{
int c=30;
static int d=40;
static{
System.out.println("B- S.B");
}
{
System.out.println("B- I.B");
}
B(){
System.out.println("B- 0 arg");
}
}
class C extends B{
int e=50;
static int f=60;
static{
System.out.println("C- S.B");
}
{
System.out.println("C- I.B");
}
C(){
System.out.println("C- 0 arg");
}
}
class Lab406{
static public void main(String args[]){
new C();
}
}
```

A) Class Loading Tasks

i) Load Class A

1) Static Variable - b will get memory.
2) Static Block of A class will be executed.

ii) Load Class B

3) Static Variable - d will get memory.
4) Static Block of B class will be executed.

iii) Load Class C

5) Static Variable - f will get memory.
6) Static Block of C class will be executed.

B) Instance Creation Tasks

iv) Invokes C class Constructor
v) Invokes B class Constructor
vi) Invokes A class Constructor

vii) Tasks of A - Instance Creation

7) Instance Variable - a will get memory.
8) Instance Block of A class will be executed.
9) Constructor of A will be executed

viii) Tasks of B - Instance Creation

10) Instance Variable - c will get memory.
11) Instance Block of B class will be executed.
12) Constructor of B will be executed

ix) Tasks of C - Instance Creation

13) Instance Variable - e will get memory.
14) Instance Block of C class will be executed.
15) Constructor of C will be executed

3.4.3 super keyword

- super is a keyword and it will be used to refer the members of immediate super class.
- super keyword cannot be accessed from static context.
- super keyword can be used in three ways:

  - **To access the immediate super class variables.**
    Syntax:
    super.<variableName>
    Ex:
    super.a super.b

  - **To access the immediate super class methods.**
    Syntax:
    super.<methodName>();
    Ex:
    super.m1(); super.m2();

  - **To access the immediate super class constructors.**
    Syntax:
    super (params);
    Ex:
    super() -> Invokes immediate super class Default Constructor
    super(99) -> Invokes immediate super class 1-Arg Constructor
    super(99,88) -> Invokes immediate super class 2-Arg Constructor

Lab407.java

```
class Hai{
int a=10; //Super
}
class Hello extends Hai{
int a=20; // INstance
void show(){
int a=30;  // Local
System.out.println(a);
System.out.println(this.a);
System.out.println(super.a);
}
}
class Lab407{
static public void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab408.java

```
class Hai{
static int a=10; //Super
}
class Hello extends Hai{
static int a=20; // INstance
void show(){
int a=30;  // Local
System.out.println(a);
System.out.println(this.a);
System.out.println(super.a);
}
}
class Lab408{
static public void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab409.java

```
class Hai{
static int a=10; //Super
}
class Hello extends Hai{
static int a=20; // INstance
void show(){
int a=30;  // Local
System.out.println(a);
System.out.println(Hello.a);
System.out.println(Hai.a);
}
}
class Lab409{
static public void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab410.java

```
class Hello {
void m1(){
System.out.println("Hello-m1()");
}
void show(){
System.out.println("show-begin");
m1();
this.m1();
System.out.println("show-end");
}
}
class Lab410{
static public void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab411.java

```
class Hai{
void m1(){
System.out.println("Hai-m1()");
}
}
class Hello  extends Hai {
void m1(){
System.out.println("Hello-m1()");
}
void show(){
System.out.println("show-begin");
m1();
this.m1();
super.m1();
System.out.println("show-end");
}
}
class Lab411{
static public void main(String args[]){
Hello h= new Hello();
h.show();
}  }
```

Lab412.java

```
class Hai{
Hai(int a){
System.out.println("Hai-1 arg ");
}
}
class Hello  extends Hai {
Hello(){
//super();
System.out.println("Hello-0 arg");
}
}

class Lab412{
static public void main(String args[]){
new Hello();
}
}
```

Lab413.java

```
class Hai{
Hai(int a){
System.out.println("Hai-1 arg ");
}
}
class Hello  extends Hai {
Hello(){
super(10);
System.out.println("Hello-0 arg");
}
}
class Lab413{
static public void main(String args[]){
new Hello();
}  }
```

Lab414.java

```
class Hai{
Hai(int a){
System.out.println("Hai-1 arg ");
}
}
class Hello  extends Hai {
Hello(){
System.out.println("Hello-0 arg");
super(10);
}
}
class Lab414{
static public void main(String args[]){
new Hello();
} }
```

Lab415.java

```
class A{
int a;
A(){
System.out.println("A - 0 arg");
}
A(int a){
System.out.println("A - 1 arg");
this.a=a;
}
}
class B  extends A {
int b;
B(){
super();
System.out.println("B-0 arg");
}
B(int a,int b){
super(a);
System.out.println("B-2 arg");
this.b=b;
}
}
class C  extends B {
int c;
C(){
super();
System.out.println("C-0 arg");
}
C(int a,int b,int c){
super(a,b);
System.out.println("C-3 arg");
this.c=c;
}
void show(){
System.out.println(a);
System.out.println(b);
System.out.println(c);
}
}
```

```
class Lab415{
static public void main(String args[]){
C co1= new C(10,20,30);
co1.show();
System.out.println("----------");
C co2= new C();
co2.show();

}
}
```

Lab416.java

```
class A{
A(){
System.out.println("A - 0 arg");
}
}

class B  extends A {
B(){
super();
System.out.println("B-0 arg");
}
B(int a){
this();
System.out.println("B-1 arg");
}
}

class Lab416{
static public void main(String args[]){
new B(10);
}
}
```

Lab417.java

```
class B  extends A {
B(){
this(10);
System.out.println("B-0 arg");
}
B(int a){
this();
System.out.println("B-1 arg");
}
}

class Lab417{
static public void main(String args[]){
new B(10);
}
}
```

Lab418.java

```
class Hello {
Hello(){
this();
}
}

class Lab418{
static public void main(String args[]){
new Hello();
}
}
```

SUMMARY

1) **Java does not support multiple inheritance with classes.**
2) **Java supports multiple inheritance with interfaces.**
3) **Java does not support Hybrid inheritance which is using multiple inheritance.**
4) **You can use extends keyword to inherit super class functionalities in sub class.**
5) **You can access only super class members by using super class object.**
6) **You can access super class members as well as sub class members by using sub class object.**
7) **All the super class members can be accessed from sub class directly.**
8) **Sub class members cannot be accessed from super class directly.**
9) **Object is the default super class for all the Java classes.**
10) **When a class is not extending any class then Object class becomes direct super class.**

    class Hello{} // Object is direct super class

11) **When a class is extending any super class then Object class becomes indirect super class.**

    class Hai{}

    class Hello extends Hai{} // Object is indirect super class

12) Java does not support cyclic inheritance.

13) Cyclic inheritance involves in two cases:

   a. class A extends A{}

   b. class A extends B{}

      class B extends A{}

14) final classes cannot be sub classed.

15) private members of super class will not be inherited to sub class.

16) When JVM loads the class then:

   a. It checks whether super class is loaded or not.

   b. If super class is not loaded then it loads super class then subclass.

   c. If super class is already loaded then it loads the subclass directly.

17) When you access static members of super class using subclass name then subclass will not be loaded.

18) When JVM creates the object then:

   a. It allocates the memory of instance variables of super class.

   b. It allocates the memory of instance variables of subclass.

   c. It executes the instance blocks and constructors of super class.

   d. It executes the instance blocks and constructors of sub class.

19) Constructors of super class will not be inherited to subclass.

20) You can use same name for super class variables, sub class variables or local variables.

21) When you access any variable directly then following things will happen:

   a. Checks whether that variable is declared in the local scope or not.

   b. If found in the local scope, that local variable will be used.

   c. If not found in the local scope then checks whether that variable is declared in the class scope or not.

   d. If found, that class level variable will be used.

   e. If not found in the class scope then checks whether that variable is inherited from super classes or not.

   f. If found, that inherited variable will be used.

22) When you have same name for local variables, class level variables and super class variables then do the following:

a. Refer the local variable directly.

b. Refer the class level variable using this keyword.

c. Refer the super class level variable using super keyword.

23) super keyword can be used to access both instance and static members.

24) super keyword cannot be used from static context.

25) When you are not writing any constructor inside the class then one default constructor will be inserted by the Java Compiler.

26) When you are writing any constructor inside the class then default constructor will not be inserted by the Java Compiler.

27) When you are not writing any super statement inside the constructor then default super will be inserted by the Java Compiler.

28) When you are writing any super statement inside the constructor then default super will not be inserted by the Java Compiler.

29) You can invoke super class constructor explicitly using super keyword.

30) call to super constructor must be first statement from subclass constructor.

31) call to this constructor must be first statement from constructor.

32) The first statement of any constructor can be either this or super, can't be both at a time.

33) Order of constructor invocation is from subclass to super class.

34) Order of constructor execution is from super class to subclass.

35) private members of super class cannot be accessed using super keyword also.

Assignment #10

Q1) What is Inheritance?

Q2) What is Base class?

Q3) What is Derived class?

Q4) What is direct super class or sub class?

Q5) What is indirect super class or sub class?

Q6) How to achieve code reusability in Java?

Q7) How to inherit the functionalities of super class in sub class?

Q8) What are the types of Inheritance available in Java?

Q9) What is Simple Inheritance? Explain with example?

Q10) What is Multilevel Inheritance? Explain with example?

Q11) What is Hierarchical Inheritance? Explain with example?

Q12) What is Multiple Inheritance? Explain with example?

Q13) What is Hybrid Inheritance? Explain with example?

Q14) Can I use Multiple Inheritance with classes?

Q15) Can I use Multiple Inheritance with interfaces?

Q16) Can I use Hybrid inheritance which is using Multiple Inheritance?

Q17) Can I access sub class members by using super class object?

Q18) How can I access sub class members from super class?

Q19) What is the default super class for all the Java classes?

Q20) What is super?

Q21) What are the ways to use super keyword in java?

Q22) How to invoke super class constructor from subclass?

Q23) Can I access super keyword from static block?

Q24) Can I access super keyword from static method?

Q25) Can I access instance members using super keyword?

Q26) Can I access static members using super keyword?

Q27) Can I access local variable using super keyword?

Q28) Can I access current class variable using super keyword?

Q29) How to stop inheritance?

Q30) Can I access private members of super class from sub class.?

Q31) Can I inherit main () method?

Q32) Can I take same name for super class variables, sub class variables or local variables?

Q33) What are the checks happening when I access any variable directly?

Q34) What are the checks happening when I access any variable using this keyword?

Q35) What are the checks happening when I access any variable using super keyword?

Q36) How can I access indirect super class members from sub class?

Q37) What will happen when I am not writing super statement in constructor?

Q38) Can I invoke super class constructor explicitly?

Q39) What is the first statement of any constructor?

Q40) What is the order of constructor invocation?

Q41) What is the order of constructor execution?

Practice Test #10

Q.No	Question	Options	Answer

1	class Test1 extends Object{
	public static void main(String args[]){
	System.out.println("Main ");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) Main
		D) No Output	

2	class Test2{
	public static void main(String args[]){
	Object obj="JLC";
	System.out.println(obj);
	} }
		A) Compilation Error
		B) Runtime Error
		C) JLC
		D) No Output	

3	class Test3{
	public static void main(String args[]){
	Object obj="JLC";
	System.out.println(obj);
	}
	}
	class Object{}
		A) Compilation Error
		B) Runtime Error
		C) JLC
		D) No Output
		E) None of above	

4	class A{} class B extends A{}	A) Object is direct super class for A B) Object is direct super class for B C) A is direct super class for B D) Object and A both are super class for B E) Object and A both are direct super class for B	

5	class Test4{
	public static void main(String args[]){
	B ref=new B();
	System.out.println(ref.y);
	} }
	class A{
	int x=99;
	}
	class B {
	int y=88;
	}
		A) Compilation Error
		B) Runtime Error
		C) 99
		D) 88
		E) None of above	

6	class Test5{
	public static void main(String args[]){
	B ref=new B();
	System.out.println(ref.x);
	}
	}
	
	class A{
	int x=99;
	}
	class B {
	int y=88;
	}
		A) Compilation Error
		B) Runtime Error
		C) 99
		D) 88
		E) None of above	

7	class Test6{
	public static void main(String args[]){
	B ref=new B();
	System.out.println(ref.x);
	} }
	class A{
	int x=99;
	}
	class B extends A{
	int y=88;
	}
		A) Compilation Error
		B) Runtime Error
		C) 99
		D) 88
		E) None of above	

8	class Test7{
	public static void main(String args[]){
	B ref=new B();
	ref.x=101;
	System.out.println(ref.ref.x);
	}
	}
	class A{
	int x=99;
	}
	class B extends A{
	int y=88;
	A ref=new A();
	}
		A) Compilation Error
		B) Runtime Error
		C) 99
		D) 88
		E) None of above	

9	class Test8{
	public static void main(String args[]){
	D ref=new D();
	System.out.println(ref.x);
	}
	}
	class A{
	int x=99;
	}
	class B extends A{ }
	class C extends B{ }
	class D extends B{ }
		A) Compilation Error
		B) Runtime Error
		C) 99
		D) JLC
		E) false
		F) None of above	

10	class Test9{
	public static void main(String args[]){
	D ref=new D();
	System.out.println(ref.x);
	} }
	class A{ int x=99; }
	class B extends A{ }
	class C extends B{
	String x="JLC";
	}
	class D extends C{
	boolean x=false;
	}
		A) Compilation Error
		B) Runtime Error
		C) 99
		D) JLC
		E) false
		F) None of above	

11	class Test10{
	public static void main(String args[]){
	D ref=new D();
	System.out.println(ref.x);
	} }
	class A{
	int x=99;
	}
	class B extends A{ }
	class C extends B{
	String x="JLC";
	}
	class D extends C{ }
		A) Compilation Error
		B) Runtime Error
		C) 99
		D) JLC
		E) false
		F) None of above	

12	class Test11{
	public static void main(String args[]){
	D ref=new D();
	System.out.println(ref.x);
	} }
	class A{ int x=99; }
	class B extends A{ }
	class C extends B{
	String x="JLC";
	}
	class D extends B{ }
		A) Compilation Error
		B) Runtime Error
		C) 99
		D) JLC
		E) false
		F) None of above	

13	class Test12{
	public static void main(String args[]){
	new B();
	}
	}
	class A{
	static{
	System.out.println("A S.B.");
	} }
	class B{
	static{
	System.out.println("B S.B.");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A S.B.
		B S.B.
		D) A S.B.
		E) B S.B.
		F) None of above	

14	class Test13{
	public static void main(String args[]){
	new B();
	} }
	class A{
	static{
	System.out.println("A S.B.");
	} }
	class B extends A{
	static{
	System.out.println("B S.B.");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A S.B.
		B S.B.
		D) A S.B.
		E) B S.B.
		F) None of above	

15	class Test14{
	public static void main(String args[]){
	B ref=null;
	ref.x=101;
	}
	}
	
	class A{
	static int x=99;
	static{
	System.out.println("A S.B.");
	}
	}
	
	class B extends A{
	static{
	System.out.println("B S.B.");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A S.B.
		B S.B.
		D) A S.B.
		E) B S.B.
		F) None of above	

16	class Test15{
	public static void main(String args[]){
	B ref=null;
	ref.x=101;
	}
	}
	
	class A{
	static int x=99;
	{
	System.out.println("A I.B.");
	}
	}
	
	class B extends A{
	{
	System.out.println("B I.B.");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A I.B.
		B I.B.
		D) A I.B.
		E) B I.B.
		F) None of above	

17	class Test16{
	public static void main(String args[]){
	new B().x=101;
	} }
	class A{
	static int x=99;
	{
	System.out.println("A I.B.");
	}
	}
	class B extends A{
	{
	System.out.println("B I.B.");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A I.B.
		B I.B.
		D) A I.B.
		E) B I.B.
		F) None of above	

18	class Test17{
	public static void main(String args[]){
	B.show();
	}
	}
	class A{
	static void show(){
	System.out.println("A -> show ()");
	}
	}
	class B extends A{ }
		A) Compilation Error
		B) Runtime Error
		C) A -> show()
		D) None of above	

19	class Test18{
	public static void main(String args[]){
	B ref=new B();
	ref.show();
	} }
	class A{
	int x=10;
	}
	class B extends A{
	boolean x=true;
	void show(){
	System.out.println(x);
	} }
		A) Compilation Error
		B) Runtime Error
		C) true
		D) 10
		E) 99
		F) None of above	

20	class Test19{
	public static void main(String args[]){
	B ref=new B();
	ref.show();
	}
	}
	class A{ int x=10; }
	class B extends A{
	void show(){
	A ref=new A();
	System.out.println(ref.x);
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) true
		D) 10
		E) 99
		F) None of above	

21	class Test20{
	public static void main(String args[]){
	B ref=new B();
	ref.x=99;
	ref.show();
	}
	}
	class A{ int x=10; }
	class B extends A{
	void show(){
	A ref=new A();
	System.out.println(ref.x);
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) true
		D) 10
		E) 99
		F) None of above	

22	class Test21{
	public static void main(String args[]){
	B ref=new B();
	ref.x=99;
	ref.show();
	} }
	class A{ int x=10; }
	class B extends A{
	void show(){
	A ref=new A();
	System.out.println(super.x);
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) true
		D) 10
		E) 99
		F) None of above	

23	class A{ int x=10; }
	class Test22 extends A{
	public static void main(String args[]){
	System.out.println(super.x);
	} }
		A) Compilation Error
		B) Runtime Error
		C) 10
		D) None of above	

24	class Test23{
	public static void main(String args[]){
	C ref=new C();
	ref.x=98;
	System.out.println(ref.x);
	}
	}
	class A{ int x=99; }
	class B extends A{ String x="JLC"; }
	class C extends B{ float x=11.1F; }
		A) Compilation Error
		B) Runtime Error
		C) 98
		D) 98.0
		E) 99
		F) 99.0
		G) JLC
		H) 11.1	

25	class Test24{
	public static void main(String args[]){
	C ref=new C();
	ref.show();
	} }
	class A{ int x=99; }
	class B extends A{ String x="JLC"; }
	class C extends B{
	float x=11.1F;
	void show(){
	System.out.println(super.x);
	} }
		A) Compilation Error
		B) Runtime Error
		C) 98
		D) 98.0
		E) 99
		F) 99.0
		G) JLC
		H) 11.1	

26	class Test25{
	public static void main(String args[]){
	B ref=new B();
	ref.super.x=9090;
	ref.show();
	} }
	class A{ int x=99; }
	class B extends A{
	String x="JLC";
	void show(){
	System.out.println(super.x);
	}}
		A) Compilation Error
		B) Runtime Error
		C) 99
		D) 9090
		E) JLC
		F) None of above	

27	class Test26{
	public static void main(String args[]){
	B ref=new B();
	}
	}
	class A{
	A(int a){
	System.out.println("A Par. C");
	} }
	class B extends A{
	B(){
	System.out.println("B Def. C");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A Par. C
		B Def. C
		D) A Par. C
		E) B Def. C
		F) None of above	

28	class Test27{
	public static void main(String args[]){
	B ref=new B();
	}
	}
	class A{
	A(int a){
	System.out.println("A Par. C");
	}
	}
	class B extends A{
	B(){
	super(10);
	System.out.println("B Def. C");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A Par. C
		B Def. C
		D) A Par. C
		E) B Def. C
		F) None of above	

29	class Test28{
	public static void main(String args[]){
	A ref=new A();
	}
	}
	class A extends Object{
	A(){
	super(10);
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) None of above
		D) No Output	

30	class Test29{
	public static void main(String args[]){
	A ref=new A();
	}
	}
	class A extends Object{
	A(){
	super(10);
	}
	}
	class Object{
	Object(int a){ }
	}
		A) Compilation Error
		B) Runtime Error
		C) None of above
		D) No Output	

31	class Test30{
	public static void main(String args[]){
	B ref=new B();
	ref.x=90;
	System.out.println(ref.x);
	}
	}
	class A{
	int x;
	}
	class B extends A{
	String x;
	}
		A) Compilation Error
		B) Runtime Error
		C) 90
		D) 0
		E) JLC
		F) null
		G) None of above	

32	class Test31{
	public static void main(String args[]){
	B ref=new B();
	ref.x="JLC";
	System.out.println(ref.x);
	}}
	class A{
	int x;
	}
	class B extends A{
	String x;
	}
		A) Compilation Error
		B) Runtime Error
		C) 90
		D) 0
		E) JLC
		F) null
		G) None of above	

33	class Test32{
	public static void main(String args[]){
	B ref=new B();
	ref.setX(90);
	System.out.println(ref.x);
	}
	}
	
	class A{
	int x;
	void setX(int x){
	this.x=x;
	}
	}
	
	class B extends A{
	String x;
	}
		A) Compilation Error
		B) Runtime Error
		C) 90
		D) 0
		E) JLC
		F) null
		G) None of above	

34	class Test33{
	public static void main(String args[]){
	B ref=new B();
	ref.setX(90);
	System.out.println(ref.getX());
	}
	}
	
	class A{
	int x;
	void setX(int x){
	this.x=x;
	}
	int getX(){
	return x;
	}
	}
	
	class B extends A{
	String x;
	}
		A) Compilation Error
		B) Runtime Error
		C) 90
		D) 0
		E) JLC
		F) null
		G) None of above	

35	class Test34{
	public static void main(String args[]){
	B ref=new B();
	ref.setX("JLC");
	System.out.println(ref.getX());
	}
	}
	
	class A{
	int x;
	int getX(){
	return x;
	}
	}
	class B extends A{
	String x;
	void setX(String x){
	this.x=x;
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) 90
		D) 0
		E) JLC
		F) null
		G) None of above	

36	class Test35{
	public static void main(String args[]){
	B r1=new B();
	B r2=new B();
	r1.x=99;
	r2.x=88;
	r1.show();
	r2.show();
	}
	}
	class A{
	int x;
	}
	class B extends A{
	void show(){
	System.out.println(super.x);
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) 99
		88
		D) 88
		99
		E) 0
		0
		F) 99
		99
		G) 88
		88	

37	class Test36{
	public static void main(String args[]){
	B r1=new B();
	B r2=new B();
	r1.x=99;
	r2.x=88;
	r1.show();
	r2.show();
	}
	}
	class A{
	static int x;
	}
	class B extends A{
	void show(){
	System.out.println(super.x);
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) 99
		88
		D) 88
		99
		E) 0
		0
		F) 99
		99
		G) 88
		88	