3.11 Abstract Classes and Abstract Methods

- abstract is a keyword.
- abstract keyword can be used in two ways:
  - with classes
  - with methods
- Classes defined without abstract modifier are called as concrete class. Concrete class can be instantiated.
- Classes defined with abstract modifier are called as abstract class. Abstract class cannot be instantiated but can be sub classed.
- Methods defined without abstract modifier are called as concrete method. Concrete method must have body.
- Methods defined with abstract modifier are called as abstract method. Abstract method must not have body.
- Abstract class can be used when you want to define default behavior for sub classes.

Lab469.java

```
class Shape{ // Concrete Class

void area(){
System.out.println("Shape - area");
}

void draw(){
System.out.println("Shape - draw");
}
}

public class Lab469{
public static void main(String args[]){
Shape myshape = new Shape();
myshape.area();
myshape.draw();
}
}
```

Lab470.java

```
class Shape{
void area();
void draw();
}

public class Lab470{
public static void main(String args[]){
Shape myshape = new Shape();
myshape.area();
myshape.draw();
}
}
```

Lab471.java

```
class Shape{
abstract void area();
abstract void draw();
}
public class Lab471{
public static void main(String args[]){
Shape myshape = new Shape();
myshape.area();
myshape.draw();
} }
```

Lab472.java

```
abstract class Shape{
abstract void area();
abstract void draw();
}
public class Lab472{
public static void main(String args[]){
Shape myshape = new Shape();
myshape.area();
myshape.draw();
} }
```

Lab473.java

```
abstract class Shape{
abstract void area();
abstract void draw();
}

class Circle extends Shape{

}
public class Lab473{
public static void main(String args[]){
Shape myshape = new Circle();
myshape.area();
myshape.draw();
}
}
```

Lab474.java

```
abstract class Shape{
abstract void area();
abstract void draw();
}
abstract class Circle extends Shape{
void draw(){
System.out.println("Circle - draw");
}
}
class MyCircle extends Circle {
void area(){
System.out.println("MyCircle  - area");
}
}
public class Lab474{
public static void main(String args[]){
Shape myshape = new MyCircle();
myshape.area();
myshape.draw();
} }
```

Lab475.java

```
abstract class Shape{
abstract void draw();
}

class Circle extends Shape{
void draw(){
System.out.println("Circle - draw");
}
}
class Square extends Shape{
void draw(){
System.out.println("Square  - draw");
}
}
public class Lab475{
public static void main(String args[]){
Shape myshape  = null;
myshape = new Circle();
myshape.draw();//1
myshape = new Square();
myshape.draw(); //2
}
}
```

Lab476.java

```
abstract class Hello {
int a=10;
static int b=20;
{
System.out.println("Hello - I.B");
}
static{
System.out.println("Hello - S.B :"+b);
}
Hello(){
System.out.println("Hello - D.C");
}
void m1(){
System.out.println("Hello - m1()");
}
static void m2(){
System.out.println("Hello - m2()");
}
abstract void m3();
}

public class Lab476{
public static void main(String args[]){
Hello.m2();
}
}
```

Lab477.java

```
abstract class Hello {
int a=10;
static int b=20;
{
System.out.println("Hello - I.B");
}
static{
System.out.println("Hello - S.B :"+b);
}
Hello(){
System.out.println("Hello - D.C");
}
void m1(){
System.out.println("Hello - m1()");
}
static void m2(){
System.out.println("Hello - m2()");
}
abstract void m3();
}

class Hai extends Hello{
void m3(){
System.out.println("Hai - m3()");
}
}

public class Lab477{
public static void main(String args[]){
Hello hello = new Hai();
hello.m1();
hello.m2();
hello.m3();
}
}
```

Lab478.java

```
public abstract class Lab478{
public static void main(String args[]){
System.out.println("Hello Guys !!!");
}
}
```

Lab478.java

```
public abstract class Lab478{
public static void main(String args[]){
System.out.println("Hello Guys !!!");
}
}
```

SUMMARY

Abstract classes and Methods

1) When you don't know the implementation of method then you can avoid the body of method.
2) When you are not writing body of method then that method must be declared as abstract.
3) When you want to force the subclasses to provide their own implementation for some methods then that method can be declared as abstract.
4) When a class contains one or more abstract methods then that class must be declared as abstract but the reverse is not true i.e. we can have an abstract class without having any abstract methods also.
5) When you don't want to instantiate the class then you can declare that class as abstract.
6) When a class is extending abstract class then that class has to override all the abstract method of that abstract super class.
7) When subclass is not overriding one or more abstract method then subclass must be declared as abstract.
8) Abstract class cannot be instantiated i.e. you cannot create the object of abstract class but you can declare reference variable.
9) Abstract keyword must be used for class and method only. It cannot be used for variable either instance or static or local primitive or reference.
10) Abstract class can contain all the members that can be written in concrete class.
11) Abstract class cannot be final.
12) You cannot use following modifier with abstract methods:

private	static	final
strictfp	synchronized	native

13) Static members of an abstract class can be accessed with class name.
14) Instance members of an abstract class will be accessed with sub class object.
15) When you create the object of subclass then instance block and constructor of abstract class will be called.

3.12 Interfaces

- interface is a keyword which is used to define User Defined Datatypes.
- Interface is a special class which is fully abstracted.
- Interface can be used to achieve the multiple inheritance in java.
- One interface can extend one or more interfaces.
- One class can implement one or more interfaces.

Syntax to declare interface:

```
<modifiers> interface <InterfaceName>
{
  // Members
}
```

- Interface can contain following members:
  - public final static variables
  - public abstract methods
  - public static inner classes
- Variables declared in the interface are by default public final and static. So it is unnecessary to use these modifiers for variables.
- Methods declared in the interface are by default public and abstract. So it is unnecessary to use these modifiers for methods.

Lab479.java

```
interface Inter1{
int A=10;
public final static int B=20;
void m1();
public abstract void m2();
}

public class Lab479{
public static void main(String args[]){
System.out.println(Inter1.A);
System.out.println(Inter1.B);
//Inter1.A=99;
//Inter1.B=88;
}
}
```

Lab480.java

```
interface Inter1{
int A=10;
void m1();
}

public class Lab480{
public static void main(String args[]){
Inter1 iref = new Inter1();
}
}
```

Lab481.java

```
interface Inter1{
int A=10;
void m1();
}

class Hello extends Inter1{

}
public class Lab481{
public static void main(String args[]){
Inter1 iref = null;
}
}
```

Lab482.java

```
interface Inter1{
int A=10;
void m1();
}

class Hello implements Inter1{

}

public class Lab482{
public static void main(String args[]){
Inter1 iref = null;
}
}
```

Lab483.java

```
interface Inter1{
int A=10;
void m1();
}

class Hello implements Inter1{
void m1(){
System.out.println("Hello - m1");
}
}

public class Lab483{
public static void main(String args[]){
Inter1 iref = new Hello();
iref.m1();
}
}
```

Lab484.java

```
interface Inter1{
int A=10;
void m1();
}

class Hello implements Inter1{
public void m1(){
System.out.println("Hello - m1");
}
}

class Lab484{
public static void main(String args[]){
Inter1 iref = new Hello();
iref.m1();
}
}
```

Lab485.java

```
interface Inter1{
int A=10;
void m1();
}

abstract class Hello implements Inter1{

}

class Lab485{
public static void main(String args[]){
System.out.println("Hello Guys");
}
}
```

Lab486.java

```
interface Inter1{
int A=10;
int B=20;
void m1();
void m2();
}

abstract class Hai implements Inter1{

public void m1(){
System.out.println("Hai - m1");
System.out.println(A);
System.out.println(B);
}

}

class Hello extends Hai {

public void m2(){
System.out.println("Hello - m2");
System.out.println(A);
System.out.println(B);
}

}

class Lab486{
public static void main(String args[]){
Inter1 iref = new Hello();
iref.m1();
iref.m2();
}
}
```

Lab487.java

```
interface Inter1{
int A=10;
int B=20;
void m1();
void m2();
}

abstract class Hai implements Inter1{

public void m1(){
System.out.println("Hai - m1");
System.out.println(A);
System.out.println(B);
}

}

class Hello extends Hai {

public void m2(){
System.out.println("Hello - m2");
System.out.println(A);
System.out.println(B);
}
void m3(){
System.out.println("Hello - m3");
}

}

class Lab487{
public static void main(String args[]){
Inter1 iref = new Hello();
iref.m1();
iref.m2();
iref.m3(); //
}
}
```

Lab488.java

```
interface Inter1{
void m1();
}

interface Inter2{
void m2();
}

class Hello implements Inter1,Inter2{
public void m1(){
System.out.println("Hello - m1");
}
public void m2(){
System.out.println("Hello - m2");
}
}

class Lab488{
public static void main(String args[]){
Hello hello= new Hello();
hello.m1();
hello.m2();
}
}
```

Lab489.java

```
interface Inter1{
void m1();
void m3();
}

interface Inter2{
void m2();
void m3();
}

class Hello implements Inter1,Inter2{

public void m1(){
System.out.println("Hello - m1");
}
public void m3(){
System.out.println("Hello - m3");
}
public void m2(){
System.out.println("Hello - m2");
}

}

class Lab489{
public static void main(String args[]){

Inter1 iref1 = new Hello();
iref1.m1();
//iref1.m2();
iref1.m3();

Inter2 iref2 = new Hello();
//iref2.m1();
iref2.m2();
iref2.m3();

}
}
```

Lab490.java

```
interface Inter1{
int A=10;
int B=20;
}

interface Inter2{
int A=99;
int C=30;
}

class Hello implements Inter1,Inter2{
public void show(){
System.out.println("Hello - show");
System.out.println(A);
System.out.println(B);
System.out.println(A);
System.out.println(C);
}

}

class Lab490{
public static void main(String args[]){
Hello h = new Hello();
h.show();

}
}
```

Lab491.java

```
interface Inter1{
int A=10;
int B=20;
}

interface Inter2{
int A=99;
int C=30;
}

class Hello implements Inter1,Inter2{
public void show(){
System.out.println("Hello - show");
System.out.println(Inter1.A);
System.out.println(B);
System.out.println(Inter2.A);
System.out.println(C);
}

}

class Lab491{
public static void main(String args[]){
Hello h = new Hello();
h.show();

}
}
```

Lab492.java

```
interface Inter1{
void show();
}

class Hai{
public void show(){
System.out.println("Hai - show");
}
}

class Hello implements Inter1 extends Hai{

}

class Lab492{
public static void main(String args[]){
Hello h = new Hello();
h.show();
}
}
```

Lab493.java

```
interface Inter1{
void show();
}

class Hai{
public void show(){
System.out.println("Hai - show");
}
}

class Hello extends Hai implements Inter1 {

}

class Lab493{
public static void main(String args[]){
Hello h = new Hello();
h.show();
}
}
```

Lab494.java

```
interface Inter1{
int A=10;
}

class Hello implements Inter1 {
void show(){
System.out.println(A);
System.out.println(Inter1.A);
System.out.println(super.A); //
}
}

class Lab494{
public static void main(String args[]){
Hello h = new Hello();
h.show();
}
}
```

Lab495.java

```
interface Inter1{
int A=10;
}
class Hai{
int A=20;
}
class Hello extends Hai implements Inter1 {
void show(){
//System.out.println(A);
System.out.println(super.A);
System.out.println(Inter1.A);
}
}

class Lab495{
public static void main(String args[]){
Hello h = new Hello();
h.show();
}
}
```

Lab496.java

```
interface Inter1{
int A=10;
}
class Hai{
int A=20;
}
class Hello extends Hai implements Inter1 {
int A=30;
void show(){
int A=40;
System.out.println(A);
System.out.println(this.A);
System.out.println(super.A);
System.out.println(Inter1.A);
}
}
class Lab496{
public static void main(String args[]){
Hello h = new Hello();
h.show();
}
}
```

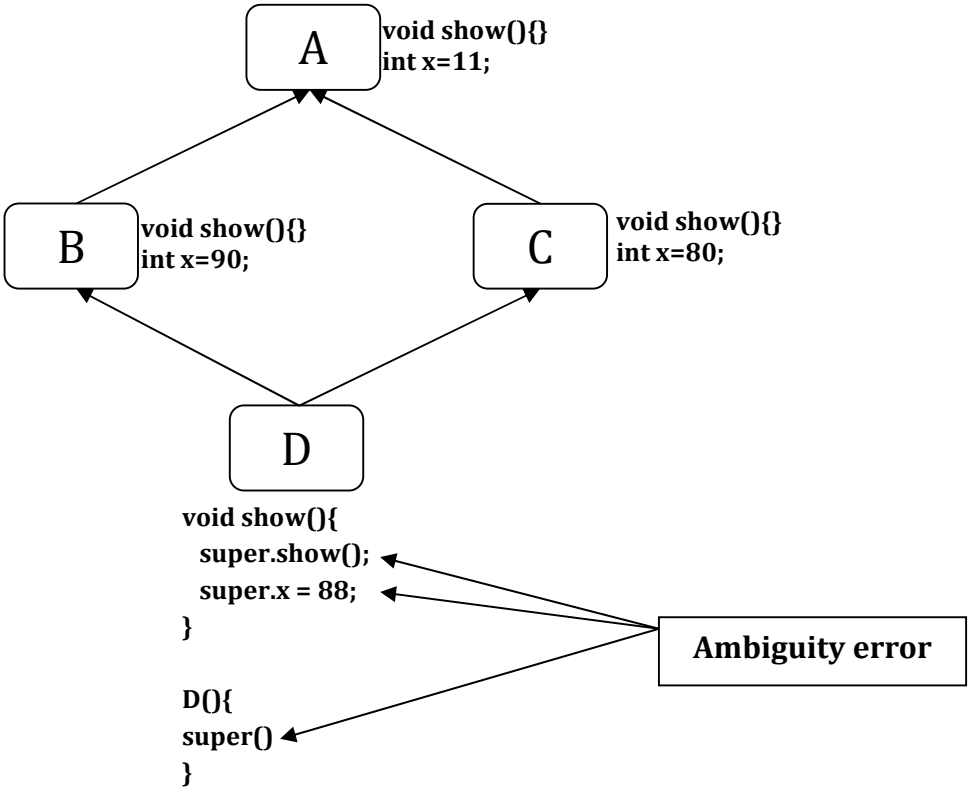
Lab497.java

```
interface Inter1{
int A=10;
static void show(){
System.out.println("Inter1 - show()");
System.out.println(A);
}
}

class Lab497{
public static void main(String args[]){
Inter1.show();

}
}
```

Lab498.java

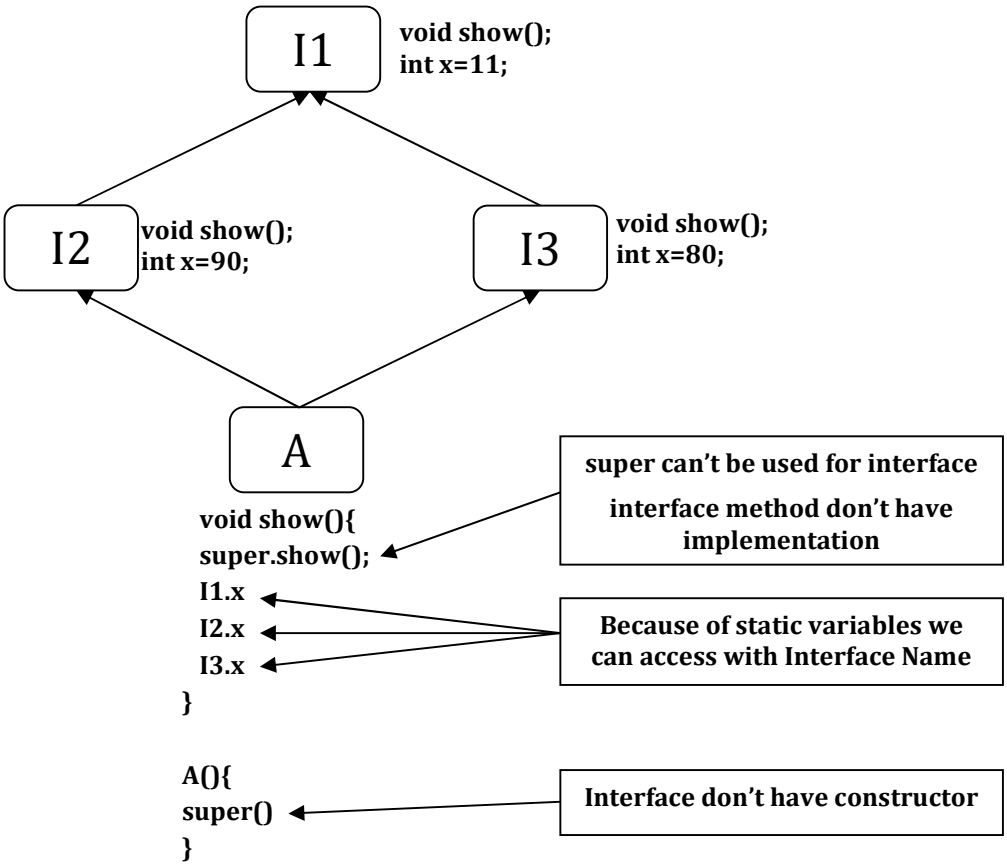
```
interface Inter1{ }
interface Inter2{ }

class A{}
class B{}

class Hello1 extends A{}  //OK
// class Hello2 extends A,B{} //Not OK
class Hello3 implements Inter1{} //OK
class Hello4 implements Inter1,Inter2{} //OK

interface Inter3 extends Inter1{}
interface Inter4 extends Inter1,Inter2{}

//interface Inter5 implements Inter1{}  //Not OK
//interface Inter6 implements Inter1,Inter2{} //Not OK

//interface Inter7 extends A{}
//interface Inter8 implements A{}
```

Problems with Multiple Inheritance using Classes



- When a class extends multiple classes then some members may common in super classes. When you refer those common members from the sub class then it results in ambiguity problem.
- When you create sub class object then first super class constructor should be invoked. When multiple super classes are there then which super class constructor will be invoked first.

Multiple Inheritance using Interfaces



SUMMARY

Interfaces

16) You can use implements keyword to inherit the functionality of an interface.

17) When you write sub class of an interface then:

   a. Subclass can use all the final static variables.

   b. Subclass has to implement all the methods of implemented interfaces.

   c. When subclass is not implementing one or more methods then subclass must be declared as abstract.

18) You cannot instantiate interface i.e. object of interface cannot be created.

19) You can declare the reference variable of interface.

20) Interface methods are by default public i.e. while implementing these methods remember to use public modifier in subclasses.

21) When you are accessing interface members by interface reference variable which contains subclass object then that members should be available in interface.

22) Interface variables can be accessed with interface name also because of static.

23) When you are implementing two or more interfaces which contains common methods then in subclass you have to provide the implementation only once and single implementation will be valid for all the interfaces.

24) When you are implementing two or more interfaces which contains common variables then reference to that variable in subclass will be ambiguous. So to resolve ambiguity you have to refer that common variable with interface name.

25) You cannot use following modifiers for interface variables:

private	protected	volatile	transient

26) You cannot use following modifiers for interface methods:

private	protected	static	final
strictfp	synchronized	native	

27) You cannot define following members in interface:

Instance Variables	Static Methods
Constructors	Initialization Blocks

28) When you define an interface without any members then it is called as MARKER INTERFACE or EMPTY INTERFACE or TAGGED INTERFACE.

Assignment #13

Method Overriding

Q1) What is method overriding?

Q2) What is the use of method overriding?

Q3) What do you mean by "Classes are closed for modifications"?

Q4) What are the purposes for writing sub class?

Q5) What are the rules of method overriding?

Q6) What is covariant return type?

Q7) Can I override private methods?

Q8) Can I override final methods?

Q9) Can I override static methods?

Q10) How to stop method overriding?

Q11) When super class method is private then while overriding in sub class what are the access modifiers are allowed?

Q12) When super class method is default then while overriding in sub class what are the access modifiers are allowed?

Q13) When super class method is protected then while overriding in sub class what are the access modifiers are allowed?

Q14) When super class method is public then while overriding in sub class what are the access modifiers are allowed?

Q15) When super class method is throwing some method level checked exception then what rules to follow in sub class method?

Q16) When super class method is throwing some method level unchecked exception then what rules to follow in sub class method?

Polymorphism

Q17) What is dynamic dispatch?

Q18) What is Polymorphism?

Q19) What are the types of Polymorphism available in Java?

Q20) What is Static Polymorphism?

Q21) What is Early Binding?

Q22) What is Compile time Polymorphism?

Q23) How to achieve Static Polymorphism?

Q24) Can I achieve Static Polymorphism with instance methods?

Q25) Can I achieve Static Polymorphism with static methods?

Q26) What is Dynamic Polymorphism?

Q27) What is Late Binding?

Q28) What is Runtime Time Polymorphism?

Q29) How to achieve Runtime Polymorphism?

Q30) Can I achieve Dynamic Polymorphism with instance methods?

Q31) Can I achieve Dynamic Polymorphism with static methods?

Abstract Classes and Methods

Q32) What is the usage of abstract modifier?

Q33) What is concrete class?

Q34) What is abstract class?

Q35) What is concrete method?

Q36) What is abstract method?

Q37) What is the usage of abstract classes?

Q38) What is the usage of abstract methods?

Q39) When I have abstract method then why the class must be defined as abstract?

Q40) Can I write subclass of an abstract class?

Q41) Can I write an abstract class without having abstract methods?

Q42) Can I create the object of abstract class?

Q43) Can I declare the reference variable of abstract class?

Q44) Can I declare abstract class as final?

Q45) Can I declare abstract class as static?

Q46) Can I declare abstract method as final?

Q47) Can I declare abstract method as private?

Q48) Can I declare abstract method as protected?

Q49) Can I declare abstract method as static?

Interfaces

Q50) What is an interface?

Q51) What is the usage of an interface?

Q52) How to achieve the multiple inheritance in java?

Q53) What are rules followed by subclasses of an interface?

Q54) What are the members allowed in interface?

Q55) What are modifiers allowed for variables declared in an interface?

Q56) What are modifiers allowed for methods declared in an interface?

Q57) What is the ambiguity problem with interfaces?

Q58) What are the limitations of multiple inheritance?

Q59) Why multiple inheritance is not allowed with classes?

Q60) Can I instantiate interface?

Q61) Can I declare the reference variable of interface?

Q62) Can I access interface variable with interface name from outside?

Q63) What are the modifiers not allowed with interface variables?

Q64) What are the modifiers not allowed with interface methods?

Q65) Can I define constructors or blocks inside the interface?

Q66) Can I define instance methods inside the interface?

Q67) What is marker interface?

Q68) What are the differences between Abstract class and Interface?

Practice Test #13

Q.No	Question	Options	Answer

1	class Test1{
	public static void main(String args[]){
	A ref=new A();
	System.out.println("Hello Guys");
	}
	}
	
	class A{ }
		A) Compilation Error
		B) Runtime Error
		C) Hello Guys
		D) None of above	

2	class Test2{
	public static void main(String args[]){
	A ref=new C();
	System.out.println("Hello Guys");
	}
	}
	
	class A{ }
	class B extends A{ }
	class C extends B{ }
		A) Compilation Error
		B) Runtime Error
		C) Hello Guys
		D) None of above	

3	class Test3{
	public static void main(String args[]){
	byte by=123;
	new A().show(by);
	}
	}
	
	class A{
	void show(int ab){
	System.out.println("show -> int");
	}
	void show(char ch){
	System.out.println("show -> char");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) show -> int
		D) show -> char
		E) None of above	

4	class Test4{
	public static void main(String args[]){
	A ref=new B();
	System.out.println("Hello Guys");
	}
	}
	class A{ }
	class B{ }
		A) Compilation Error
		B) Runtime Error
		C) Hello Guys
		D) None of above	

5	class Test5{
	public static void main(String args[]){
	new A().show(65);
	}
	}
	class A{
	void show(short ab){
	System.out.println("show -> short");
	}
	void show(char ch){
	System.out.println("show -> char");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) show -> short
		D) show -> char
		E) None of above	

6	class Test6{
	public static void main(String args[]){
	A ref=new A();
	ref.show(65);
	}
	}
	
	class A{
	void show(int ab){
	System.out.println("A.show -> int");
	}
	}
	
	class B extends A{
	void show(int ab){
	System.out.println("B.show -> int");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A.show -> int
		D) B.show -> int
		E) None of above	

7	class Test7{
	public static void main(String args[]){
	A ref=new B();
	ref.show(65);
	}
	}
	class A{
	void show(int ab){
	System.out.println("A.show -> int");
	}
	}
	class B extends A{
	void show(int ab){
	System.out.println("B.show -> int");
	} }
		A) Compilation Error
		B) Runtime Error
		C) A.show -> int
		D) B.show -> int
		E) None of above	

8	class Test8{
	public static void main(String args[]){
	A ref=new B();
	ref.show(65);
	}
	}
	class A{
	static void show(int ab){
	System.out.println("A.show -> int");
	}
	}
	class B extends A{
	static void show(int ab){
	System.out.println("B.show -> int");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A.show -> int
		D) B.show -> int
		E) None of above	

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

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

11	class Test11{
	public static void main(String args[]){
	A ref=new B();
	ref.x=88;
	System.out.println(ref.x);
	}
	}
	class A{
	int x=99;
	}
	class B extends A{
	String x="JLC";
	}
		A) Compilation Error
		B) Runtime Error
		C) 88
		D) JLC
		E) 99
		F) None of above	

12	class Test12{
	public static void main(String args[]){
	A ref=new A();
	ref.m1();
	}
	}
	class A {
	abstract void m1(){
	System.out.println("A-> m1");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A-> m1
		D) None of above	

13	class Test13{
	public static void main(String args[]){
	A.m1();
	}
	}
	abstract class A {
	abstract static void m1(){
	System.out.println("A-> m1");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A-> m1
		D) None of above	

14	class Test14{
	public static void main(String args[]){
	A ref=new B();
	ref.m1();
	}
	}
	abstract class A {
	public abstract void m1();
	}
	class B extends A{
	void m1(){
	System.out.println("B-> m1");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) B-> m1
		D) None of above	

15	class Test15{
	public static void main(String args[]){
	System.out.println(B.x);
	} }
	abstract class A {
	static int x=99;
	A(){
	System.out.println("A-> D.C");
	}
	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
		99
		D) B-> S.B
		99
		E) A-> S.B
		99
		F) None of above	

16	class Test16{
	public static void main(String args[]){
	System.out.println(new B().x);
	}
	}
	
	abstract class A {
	int x=99;
	A(){
	System.out.println("A-> D.C");
	}
	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
		A-> D.C
		99
		D) B-> S.B
		A-> D.C
		99
		E) A-> S.B
		A-> D.C
		99
		F) None of above	

17	class Test17{
	public static void main(String args[]){
	new B().m1();
	System.out.println(C.x);
	}
	}
	
	abstract class A {
	static int x=90;
	void m1(){}
	}
	class B extends A{
	B(){ x=80; }
	void m1(){
	System.out.println("B-> m1");
	}
	}
	
	class C extends A{ }
		A) Compilation Error
		B) Runtime Error
		C) B-> m1
		90
		D) B-> m1
		80
		E) None of above	

18	class Test18{
	public static void main(String args[]){
	Inter1 in=new A();
	in.m1();
	}
	}
	interface Inter1{
	void m1();
	}
	class A implements Inter1{
	void m1(){
	System.out.println("A-> m1");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A-> m1
		D) None of above	

19	class Test19{
	public static void main(String args[]){
	Inter1 in=new A();
	in.m1();
	}
	}
	interface Inter1{
	void m1();
	}
	class A implements Inter1{
	public void m1(){
	System.out.println("A-> m1");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A-> m1
		D) None of above	

20	class Test20{
	public static void main(String args[]){
	A ref1=new A();
	C ref2=new C();
	System.out.println(ref1.x);
	System.out.println(ref2.x);
	} }
	interface Inter1{
	int x=90;
	}
	class A implements Inter1{ }
	class B implements Inter1{ }
		A) Compilation Error
		B) Runtime Error
		C) 90
		90
		D) 90
		80
		E) None of above	

21	class Test21{
	public static void main(String args[]){
	A ref1=new A();
	ref1.x=80;
	System.out.println(ref1.x);
	C ref2=new C();
	System.out.println(ref2.x);
	}
	}
	interface Inter1{
	int x=90;
	}
	
	class A implements Inter1{ }
	class B implements Inter1{ }
		A) Compilation Error
		B) Runtime Error
		C) 80
		80
		D) 80
		90
		E) 80
		80
		F) None of above	

22	class Test22{
	public static void main(String args[]){
	Inter1 in=new A();
	in.m1();
	in.m2();
	}
	}
	
	interface Inter1{
	void m1();
	}
	
	interface Inter2{
	void m2();
	}
	
	class A implements Inter1,Inter2{
	public void m1(){
	System.out.println("A-> m1");
	}
	public void m2(){
	System.out.println("A-> m2");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) A-> m1
		A-> m2
		D) None of above	

23	class Test23{
	public static void main(String args[]){
	Inter1 in1=new A();
	in1.m1();
	Inter2 in2=new A();
	in2.m2();
	}}
	interface Inter1{
	void m1();
	}
	interface Inter2{
	void m2();
	}
	class A implements Inter1,Inter2{
	public void m1(){
	System.out.println("A-> m1");
	}
	public void m2(){
	System.out.println("A-> m2");
	}}
		A) Compilation Error
		B) Runtime Error
		C) A-> m1
		A-> m2
		D) None of above	

24	class Test24{
	public static void main(String args[]){
	Inter1 in[]=new A[5];
	}}
	interface Inter1{ }
	class A implements Inter1{
	A(){
	System.out.println("A D.C");
	}}
		A) Compilation Error
		B) Runtime Error
		C) A D.C
		D) None of above	