Chapter 8
Java Abstract Classes & Interfaces

The division of the chapter is as follows:

References: Data and Methods:

P1

```
class A
{
int data1 = 10;
void show1()
{
System.out.println("A data = " + data1);
}
}
class B extends A
{
int data2 = 20;
void show2()
{
System.out.println("B data = " + data2);
}
}
class RefTest10a
{
public static void main(String args[])
{
A a = new A();
System.out.println("Data1 = " + a.data1);
a.show1();
}
}
```

Output:

Data1 = 10

A data = 10

P2

```
class A
{
int data1 = 10;
void show1()
{
System.out.println("A data = " + data1);
}
}
class B extends A
{
int data2 = 20;
void show2()
{
System.out.println("B data = " + data2);
}
}
class RefTest10b
{
public static void main(String args[])
{
B b = new B();
System.out.println("Data1 = " + b.data1);
System.out.println("Data2 = " + b.data2);
b.show1();
b.show2();
}}
```

Output:

Data1 = 10

Data 2 = 20

A Data = 10

B Data = 20

P3

```
class A
{
int data1 = 10;
void show1()
{
System.out.println("A data = " + data1);
}
}
class B extends A
{
int data2 = 20;
void show2()
{
System.out.println("B data = " + data2);
}
}
class RefTest10e1
{
public static void main(String args[])
{
A a = new B();
System.out.println("Data1 = " + a.data1);
System.out.println("Data2 = " + a.data2);
a.show1();
a.show2();
}}
```

Output: CF, BOZ
Class A does not have data2 and show2.

This is called a polymorphic reference: the reference of class A can refer to objects of class A and objects of its subclass B also.

P4

```
class A
{
int data = 10;
void show()
{
System.out.println("A data = " + data);
}
}
class B extends A
{
int data = 20;
void show()
{
System.out.println("B data = " + data);
}
}
class RefTest10e
{
public static void main(String args[])
{
A a = new B();
System.out.println("Data1 = " + a.data);
System.out.println("Data2 = " + a.data);
a.show();
a.show();
}}
```

Data and method should be Common.
reference object data method
Super Sub overridden overridden
(Super) (Sub)

Output:
Data1 = 10
Data2 = 10
B data = 2
B data = 20

Abstract Classes:

- A class contains **description properties or variables** and **actions or methods** of its objects.
- The rule is that **anything** that is written **in** the **class** is **applicable** to **all** of its **objects**.
- If a method is written in the class, it is available to all of its class objects.

Eg: Method calculate which calculates the square of a number is available to all the objects.

Program:

```
class MyClass
{
void calculate(double x)
{
System.out.println("Square "+ (x * x));
}
}

class Common
{
public static void main(String args[])
{
MyClass o1 = new MyClass();
MyClass o2 = new MyClass();
MyClass o3 = new MyClass();
o1.calculate(3);
o2.calculate(4);
o3.calculate(5);
}
}
```

Output:

```
Square 9.0
Square 16.0
Square 25.0
```

Requirement:

- In the preceding program, the requirement of all the objects is same i.e. to calculate the square value.
- Suppose the **requirement** is that the **first object** should **calculate square** value, **second object** should **calculate square root** value and **third object** should **calculate the cube** value.
- Since, calculate() **method** has to **perform** three **different tasks** depending on the **object**, we cannot write the code to calculate the square value in the body of calculate method.
- On the other hand, if we write **three different methods** like calSqr(), calSqrt() and calCube() in MyClass, **then all** three methods are **available to all** the three objects which is **not advisable**.

Solution:

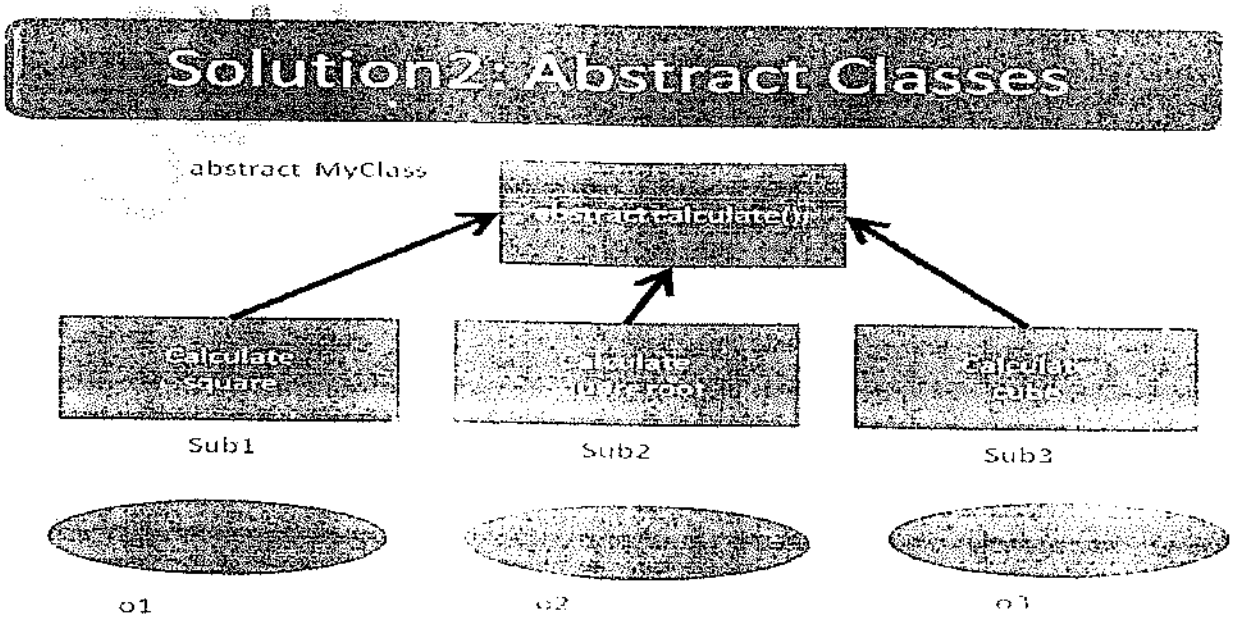
To serve each object with one and only one required method, we can perform the following steps:

1) Write calculate() method in MyClass, meaning every object wants to calculate something.

2) Do not write body for the calculate() method. Such a method is called abstract method and the class would be called abstract class.

3) Derive Sub1 from MyClass and write the body for calculate() method to find square, Sub2 for square root and Sub3 for cube.

4) Create objects for the sub classes and call the calculate method for each object.



Program for Abstract Classes:

```
abstract class MyClass{
abstract void Calculate(double x);
}

class Sub1 extends MyClass {
void calculate(double x) {
System.out.println("Square=" + (x*x) );
}
}
class Sub2 extends MyClass {
void calculate(double x) {
System.out.println("Square root=" + Math.sqrt(x) );
}
}
class Sub3 extends MyClass {
void calculate(double x) {
System.out.println( "Cube=" + (x*x*x) );
}
}

class Different {
public static void main(String args[]) {
Sub1 o1 = new Sub1();
Sub2 o2 = new Sub2();
Sub3 o3 = new Sub3();
o1.calculate(3);
o2.calculate(4);
o3.calculate(5);
}
}
```

Output:

```
>javac Different.java
>java Different
Square=9.0
Square root=2.0
Cube=125.0
```

Q1) What if we don't override the method of the abstract classes?

Ans: CF, we must override all the methods of the abstract classes in concrete SubClass.

Q2) Can we create objects of abstract class?

Ans: No, CF bcoz Abstract class cannot be instantiated

Q3) Can we create references of abstract class?

Ans: Yes, we can create ref. of AbstractClass and refer to the objects of Subclasses. (Polymorphic reference)

Rules for Abstract Classes:

Rule1: It is illegal to declare an abstract method in a class which is not declared abstract.

```
Eg:
class C2 {
abstract void show1();
}
```

X CF.

Rule2: An abstract class can be declared

```
Eg:
abstract class C2a {
void show1() {
System.out.println("Show1 method");
}}
```

~~CF [AP] Abstract class can not have body.~~ abstract class can have non abstract method. ✓

Rule3: A method cannot be declared as both abstract and final

```
Eg:
abstract class C30a {
final abstract void show1();
}
```

X CF. Can't both final or abstr

Rule4: A method cannot be declared private and abstract.

```
Eg:
abstract class C30b {
private abstract void show1();
}
```

X CF Illegal Combination.

Rule5: If the subclass is abstract, then it is not mandatory to implement all abstract methods of a superclass ✓

```
Eg:
abstract  class C30e {
abstract void show1();
}

abstract class C30f extends C30e {
abstract void show2();
}
```

Rule6: A non-abstract class must implement all the abstract methods of a Superclass.

Eg:

```
abstract  class c2
{
abstract void show1();
}

abstract class c3 extends c2
{
abstract void show2();
}

class c4 extends c3
{
void show1()
{
System.out.println(" Show1() " );
}
void show2()
{
System.out.println(" Show2() " );
}
public static void main(String args[])
{
c4 c = new c4();
c.show1();
c.show2();
}
}
```

Interfaces:

- An Abstract class is a class which contains some abstract methods as well as concrete method.
- An Interface is a class that contains methods which are all abstract.

Requirement:

Suppose a programmer is asked to write a Java program to:

- connect to a database
- retrieve the data from the database,
- process the data,
- display the result in the form of some reports and
- disconnect from database

Solution:

- The programmer writes an interface with abstract methods as shown.
- Interfaces are declared using the interface keyword.
- Interface contains abstract methods which are public and declarations are final.
- It is not possible to create an object to an interface.
- Hence we create separate classes where we can implement all the methods of the interface. These classes are called implementation classes.
- It is possible to create objects of implementation classes.

Program for Interfaces:

Interface MyInterd
Void connect();
Void disconnect();
}

Class OracleImp implements MyInterd

```
public void connect() {
// write code to connect to Oracle database
System.out.println("Connecting to Oracle DB");
}
public void disconnect() {
// disconnect from Oracle database
System.out.println("DisConnecting from Oracle DB");
}
}
```

class SybaseImp implements MyInterd
~~Public Void disconnect() {~~

```
public void connect() {
// write code to connect to Sybase database
System.out.println("Connecting to Sybase DB");
}
public void disconnect() {
// disconnect from Sybase database
System.out.println("DisConnecting from Sybase DB");
}
}
```

```
class IntTest2 {
public static void main (String args[])
{
MyInter mi;
mi = new OracleImp();
mi.connect();
mi.disconnect();
mi = new SybaseImp();
mi.connect();
mi.disconnect();
}
}
```

Output:

```
>javac IntTest2.java

>java IntTest2
```

Connecting to oracle DB
Disconnecting from oracle DB
Connecting to Sybase DB
Disconnecting from Sybase DB.

Multiple Inheritance using Interface:

- The advantage of interface is that it enables Multiple Inheritance as it is possible for a class to implement more than one interface.
- In Multiple Inheritance(MI), subclasses are derived from multiple super classes.
- If two classes have the same name for their variables or methods, then which member is inherited into sub class is the main confusion in MI.
- This confusion can be avoided by using multiple interfaces to achieve MI.

Program for Multiple Inheritance:

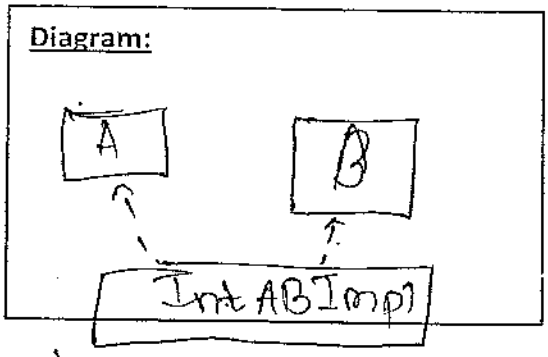
```
interface A
{
int x = 20;
void method();
}

interface B
{
int x = 30;
void method();
}

class IntABImpl implements A, B
{
        public void method()
        {
                System.out.println("A's x = " + A.x + " B's x = " + B.x);
        }

        public static void main(String args[])
        {
                IntABImpl in = new IntABImpl();
                in.method();
        }
}
```

Diagram:



Implementation class.

Output: A's x = 20 B's x = 30

Rules for Interfaces:

Rule1:
All methods in interface are implicitly public and abstract.
They cannot be private or protected.

Eg:

```
interface IntA
{
int X=20;
private void display();
}
```

```
interface IntA
{
int X=20;
protected void display();
}
```

Rule2:
Interface methods must not be static or, final, strictfp or native.

Eg:

```
interface IntA
{
int X=20;
static void display();
}
```

```
interface IntA
{
int X=20;
final void display();
}
```

Rule3:
An interface declares only constants and not instance variables. All variables declared in an interface are public, final and static.

Eg:

```
interface IntA
{
int X;
void display();
}
```

Rule4:
When implementing interface the method of interface should
have access specifier public.

```
interface IntA
{
int X=20;
void display();
}
class InterAImpl implements IntA
{
void display()
{
System.out.println("A's x = " + IntA.X);
}
public static void main(String args[])
{
InterAImpl ai = new InterAImpl();
ai.display();
}
}
```

CF.

```
interface IntA
{
int X=20;
void display();
}
class InterAImpl implements IntA
{
public void display()
{
System.out.println("A's x = " + IntA.X);
}
public static void main(String args[])
{
InterAImpl ai = new InterAImpl();
ai.display();
}
}
```

O/P A's x = 20

Rule5:
An interface cannot implement another interface.

Eg:

```
interface IntA1
{
int X=20;
void display();
}
interface IntB1 implements IntA1
{
int Y=30;
void show();
}
```

CP

Rule6:
An interface can extend one or more interface.

```
interface IntA
{
int X=20;
void display();
}
interface IntB extends IntA
{
int Y=30;
void show();
}
class InterABImpl implements IntB
{
public void display()
{
System.out.println("A's x = " + X);
}
public static void main(String args[])
{
InterABImpl ai = new InterABImpl();
ai.display();
}
}
```

```
interface IntA
{
int X=20;
void display();
}
interface IntB extends IntA
{
int Y=30;
void show();
}
class InterABImpl implements IntB
{
public void display()
{
System.out.println("A's x = " + X);
}
public void show()
{
System.out.println("A's x = " + X);
System.out.println("B's x = " + Y);
}
public static void main(String args[])
{
InterABImpl ai = new InterABImpl();
ai.display();
ai.show();
}
}
```

A's x = 20
A's y = 20
B's x = 30

	Abstract Classes	Interfaces
Variables	Abstract class can contain instance variables also.	Interface cannot contain instance variables. It contains only constants.
Methods	Abstract class can contain some abstract methods and some concrete methods.	Interface can contain only abstract methods.
Implementation	All the abstract methods of the abstract class should be implemented in its sub classes.	All the abstract methods of the interface should be implemented in its implementation classes.
Constructors	Abstract classes can have constructors, and those constructors are always called when a concrete subclass is instantiated.	Interfaces do not have constructors.
Multiple Inheritance	Abstract classes does not support multiple inheritance	Interfaces support multiple inheritance
Changes	If a new method is added to an abstract class, then there is an option of providing implementation and therefore all existing code works without any change.	If a new method is added to an interface, then it is required to track down all the implementations of the interface and define it.
Usefulness	Abstract classes are useful in situation when some general methods should be implemented and specialization behavior should be implemented by the subclasses.	Interfaces are useful in a situation when all it methods need to be implemented by subclasses.

extends and implements:

- class C1 extends C2 implements I1, I2
- interface I1 extends I2 I3.

Examples of Legal and Illegal Use of extends and implements for Classes C1, C2 and C3 and Interfaces I1,I2 and I3.

No	Scenario	Y / N	Reason
1	class C1 extends C2 { }	Yes	
2	class C1 implements I1 { }	Yes	
3	class C1 extends C2, C3 { }	No	
4	class C1 implements I1, I2 { }	Yes	
5	class C1 extends I1 { }	No	
6	class C1 extend C2{ }	No	
7	class C1 extends C2 implements I1 { }	Yes	
8	class C1 implements I1 extends C2 { }	No	extends must come before implements
9	interface I1 implements I2 { }	No	
10	interface I1 extends I2 { }	Yes	
11	interface I1 extends I2, I3 { }	Yes	
12	interface I1 extends C1 { }	No	
13	interface I1 implements C1 { }	No	

Test Paper

Q1) Given:

```
31. class Foo {
32. public int a = 3;
33. public void addFive() { a += 5; System.out.print("f "); }
34. }
35. class Bar extends Foo {
36. public int a = 8;
37. public void addFive() { this.a += 5; System.out.print("b " ); }
38. }
```

Invoked with:

```
Foo f = new Bar();
f.addFive();
System.out.println(f.a);
```

What is the result?

ref | obj | data | method.
Sup. | sub | Super | sub.

Options:

A. b 3
B. b 8
C. b 13
D. f 3
E. f 8
F. f 13
G. Compilation fails.
H. An exception is thrown at runtime.

Solution:

Q2)

```
3. class Mammal {
4. String name = "furry ";
5. String makeNoise() { return "generic noise"; }
6. }
7. class Zebra extends Mammal {
8. String name = "stripes ";
9. String makeNoise() { return "bray"; }
10. }
11. public class ZooKeeper {
12. public static void main(String[] args) { new ZooKeeper().go(); }
13. void go() {
14. Mammal m = new Zebra();
15. System.out.println(m.name + m.makeNoise());
16. }
17. }
```

What is the result?

object on the fly.

furry bray.

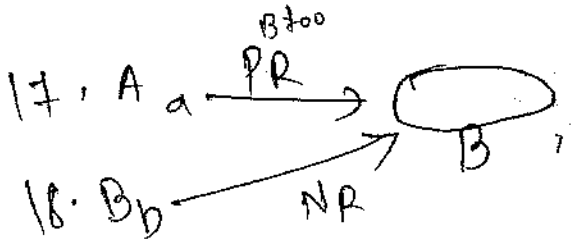
Options:

A. furry bray
B. stripes bray
C. furry generic noise
D. stripes generic noise
E. Compilation fails

Solution: A

Q3) Given:

```
11. class Alpha {
12. public void foo() { System.out.print("Afoo "); }
13. }
14. public class Beta extends Alpha {
15. public void foo() { System.out.print("Bfoo "); }
16. public static void main(String[ ] args) {
17. Alpha a = new Beta();
18. Beta b = (Beta)a;
19. a.foo();
20. b.foo();
21. }
22. }
```



What is the result?

Options:

A. Afoo Afoo
B. Afoo Bfoo
C. Bfoo Afoo
D. Bfoo Bfoo
E. Compilation fails.
F. An exception is thrown at runtime.

Solution: D

Q4) Given:

```
10. abstract public class Employee {
11. protected abstract double getSalesAmount();
12. public double getCommision() {
13. return getSalesAmount() * 0.15;
14. }
15. }
16. class Sales extends Employee {
17. // insert method here
18. }
```

Which two methods, inserted independently at line 17, correctly complete the Sales class?(Choose two.)

Options:

A. double getSalesAmount() { return 1230.45; }
B. public double getSalesAmount() { return 1230.45; }
C. private double getSalesAmount() { return 1230.45; }
D. protected double getSalesAmount() { return 1230.45; }

Solution: B D

Q5) What is the result of the following program?

```
1. public abstract class Book {
2. public final void read() {
3. System.out.println("Reading a Book");
4. }
5.
6. public static void main(String [] args) {
7. Book book = new NonFictionBook();
8. book.read();
9. }  10. }
11.
12. class NonFictionBook extends Book {
13. public void read() {
14. System.out.println("Reading a NonFictionBook");
15. }  16. }
```

Options:

A. Reading a Book
B. Reading a NonFictionBook
C. Compiler error on line 7
D. Compiler error on line 8
E. Compiler error on line 13

Solution:

Q6) What is the result of the following code?

```
1. public abstract class Book {
2. public abstract void read();
3.
4. public static void main(String [] args) {
5. Book book = new NonFictionBook();
6. book.read();
7. }  8. }
9.
10. class NonFictionBook extends Book {
11. public void read(int time) {
12. System.out.println("Reading a NonFictionBook");
13. }  14. }
```

Options:

A. Reading a NonFictionBook
B. Compiler error on line 5
C. Compiler error on line 6
D. Compiler error on line 10
E. An exception occurs at runtime on line 6.

Solution:

Q7) Given the following class definitions:

```
1. public class Parent {
2. protected void sayHi() {
3. System.out.print("Hi");
4. }   5. }
6.
7. class Child extends Parent {
8. public void sayHi() {
9. System.out.print("Hello");
10. }  11. }
```

what is output of the result of the following statements?

```
15. Parent p = new Child();
16. p.sayHi();
```

Options:

A. Hi
B. Hello
C. Compiler error on line 8
D. Compiler error on line 15
E. Line 16 causes an exception to be thrown.

Solution: B

Q8) Given:

```
11. abstract class Vehicle { public int speed() { return 0; }}
12. class Car extends Vehicle { public int speed() { return 60; }}
13. class RaceCar extends Car { public int speed() { return 150; }} ...
...
21. RaceCar racer = new RaceCar();
22. Car car = new RaceCar();
23. Vehicle vehicle = new RaceCar();
24. System.out.println(racer.speed() + ", " + car.speed()
25. + ", " + vehicle.speed());
```

What is the result?

Options:

A. 0, 0, 0
B. 150, 60, 0
C. Compilation fails.
D. 150, 150, 150
E. An exception is thrown at runtime.

Solution: D

Q9) Given:

```
21. abstract class C1 {
22. public C1() { System.out.print(1); }
23. }
24. class C2 extends C1 {
25. public C2() { System.out.print(2); }
26. }
27. class C3 extends C2 {
28. public C3() { System.out.println(3); }
29. }
30. public class Ctest {
31. public static void main(String[ ] a) { new C3(); }
32. }
```

What is the result?

every subclass constructor calls DC of superclass.

Options:

A. 3
B. 23
C. 32
D. 123
E. 321
F. Compilation fails.
G. An exception is thrown at runtime.

Solution: D

Q10) Given:

```
11. public abstract class Shape {
12. private int x;
13. private int y;
14. public abstract void draw();
15. public void setAnchor(int x, int y) {
16. this.x = x;
17. this.y = y;
18. } 19. }
```

Which two classes use the Shape class correctly? (Choose two.)

Options:

A.
public class Circle implements Shape {
private int radius;
}
	B.
	public abstract class Circle extends Shape {
	private int radius;
	}

C.
public class Circle extends Shape {
private int radius;
public void draw();
}
	D.
	public abstract class Circle implements Shape {
	private int radius;
	public void draw();
	}

E.
public class Circle extends Shape {
private int radius;
public void draw() {/* code here */}
}
	F.
	public abstract class Circle implements Shape {
	private int radius;
	public void draw() { /* code here */ }
	}

Solution: B & E

Hint!- in Interface ① all methods are public and abstract.
②all variables are ~~final~~, public, static and final.

Q11) Given

```
11. public interface Status {
12. /* insert code here */ int MY_VALUE = 10;
13. }
```

Which three are valid on line 12? (Choose three.)

Options:

A. final ~~B. static~~ C. native
D. public E. private F. abstract
G. protected

Solution: A, B, D

Q12) Which two classes correctly implement both the java.lang.Runnable and the java.lang.Cloneable interfaces? (Choose two.)

Options:

A.
```
public class Session
implements Runnable, Cloneable {
public void run();
public Object clone();
```
nobody.

B.
```
public class Session
extends Runnable, Cloneable {
public void run() { /* do something */ }
public Object clone() { /* make a copy */ }
```

C.
```
public class Session
implements Runnable, Cloneable {
public void run() { /* do something */ }
public Object clone() { /* make a copy */ }
```

D.
```
public abstract class Session
implements Runnable, Cloneable {
public void run() { /* do something */ }
public Object clone() { /*make a copy */ }
```

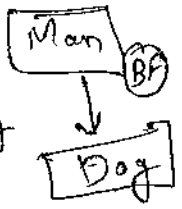
E.
```
public class Session
implements Runnable, implements Cloneable {
public void run() { /* do something */ }
public Object clone() { /* make a copy */ }
```

Solution:

Q13) Which Man class properly represents the relationship
"Man has a best friend who is a Dog"?

Options:

A. class Man extends Dog { }
B. class Man implements Dog { }
C. class Man { private BestFriend dog; } Class Man may or may not be a BF
D. class Man { private Dog bestFriend; }
E. class Man { private Dog<bestFriend>; }
F. class Man { private BestFriend<dog>; }



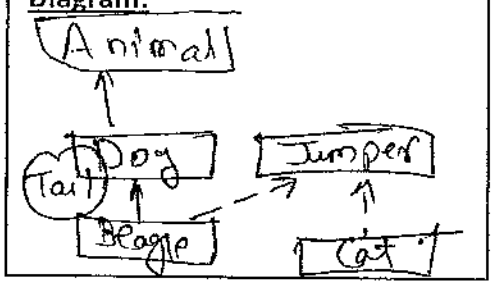
Solution:

Q14)Given:

```
10. interface Jumper { public void jump(); }
...
20. class Animal {}
...
30. class Dog extends Animal {
31. Tail tail;
32. }
...
40. class Beagle extends Dog implements Jumper{
41. public void jump() {}
42. }
...
50. class Cat implements Jumper{
51. public void jump() {}
52. }
```

Which three are true? (Choose three.)

Diagram:



Options:

A. Cat is-a Animal
B. Cat is-a Jumper
C. Dog is-a Animal
D. Dog is-a Jumper
E. Cat has-a Animal
F. Beagle has-a Tail
G. Beagle has-a Jumper

Solution:

Q15) Given:

```
10. interface Data { public void load(); }
11. abstract class Info { public abstract void load(); }
```

Which class correctly uses the Data interface and Info class?

Options:

A. public class Employee extends Info implements Data { public void load() { /*do something*/ }}	B. public class Employee implements Info extends Data { public void load() { /*do something*/ } }

C.
public class Employee extends Info
implements Data {
public void load(){ /*do something*/ }
public void Info.load(){ /*do something*/ }
}
	D. public class Employee implements
	Info extends Data {
	public void Data.load(){ /*do something*/
	}
	public void load(){ /*do something*/ } }

E.
public class Employee implements Info
extends Data {
public void load(){ /*do something*/ }
public void Info.load(){ /*do something*/ }
}
	F. public class Employee extends Info
	implements Data{
	public void Data.load() { /*do something*/
	}
	public void Info.load() { /*do something*/
	}}

Solution:

Q16) Given:

```
11. public interface A111 {
12. String s = "yo";
13. public void method1();
14. }
17. interface B { }
20. interface C extends A111, B {
21. public void method1();
22. public void method1(int x);
23. }
```

What is the result?

Options:

A. Compilation succeeds.
B. Compilation fails due to multiple errors.
C. Compilation fails due to an error only on line 20.
D. Compilation fails due to an error only on line 21.
E. Compilation fails due to an error only on line 22.
F. Compilation fails due to an error only on line 12.

Solution:

Q17) Given:

```
1. interface DoStuff2 {
2. float getRange(int low, int high); }
3.
4. interface DoMore {
5. float getAvg(int a, int b, int c); }
6.
7. abstract class DoAbstract implements DoStuff2, DoMore { }
8.
9. class DoStuff implements DoStuff2 {
10. public float getRange(int x, int y) { return 3.14f; } }
11.
12. interface DoAll extends DoMore {
13. float getAvg(int a, int b, int c, int d); }
```

What is the result?

Options:

A. The file will compile without error.
B. Compilation fails. Only line 7 contains an error.
C. Compilation fails. Only line 12 contains an error.
D. Compilation fails. Only line 13 contains an error.
E. Compilation fails. Only lines 7 and 12 contain errors.
F. Compilation fails. Only lines 7 and 13 contain errors.

Solution:

Q18) Given:

```
11. public interface A { public void m1(); }
12.
13. class B implements A { }
14. class C implements A { public void m1() { } }
15. class D implements A { public void m1(int x) { } }
16. abstract class E implements A { }
17. abstract class F implements A { public void m1() { } }
18. abstract class G implements A { public void m1(int x) { } }
```

What is the result?

Options:

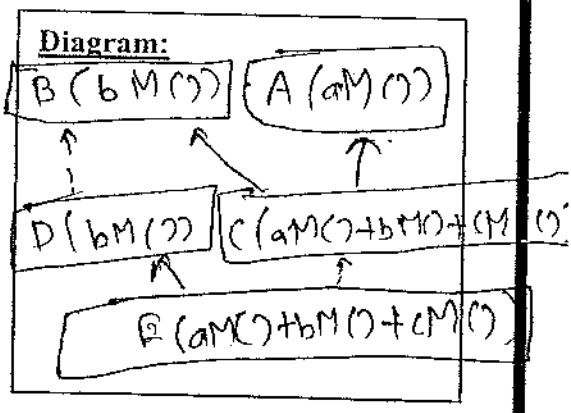
A. Compilation succeeds.
B. Exactly one class does NOT compile.
C. Exactly two classes do NOT compile.
D. Exactly four classes do NOT compile.
E. Exactly three classes do NOT compile.

Solution:

Q19) Given:

```
1. interface A { public void aMethod(); }
2. interface B { public void bMethod(); }
3. interface C extends A,B { public void cMethod(); }
4. class D implements B {
5. public void bMethod() { }
6. }
7. class E extends D implements C {
8. public void aMethod() { }
9. public void bMethod() { }
10. public void cMethod() { }
11. }
```

What is the result?



Options:

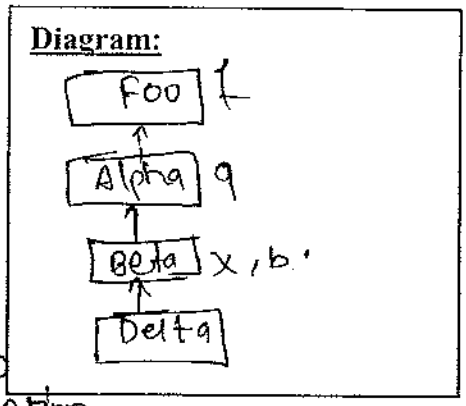
A. Compilation fails because of an error in line 3.
B. Compilation fails because of an error in line 7.
C. Compilation fails because of an error in line 9.
D. If you define D e = new E(), then e.bMethod() invokes the version of bMethod() defined in Line 5.
E. If you define D e = (D)(new E()), then e.bMethod() invokes the version of bMethod() defined in Line 5.
F. If you define D e = (D)(new E()), then e.bMethod() invokes the version of bMethod() defined in Line 9.

Solution:

Q20) Given:

```
10. interface Foo { }
11. class Alpha implements Foo { }
12. class Beta extends Alpha { }
13. class Delta extends Beta {
14. public static void main( String[ ] args ) {
15. Beta x = new Beta();
16. // insert code here
17. }
18. }
```

Which code, inserted at line 16, will cause a java.lang.C



antException.

Options:

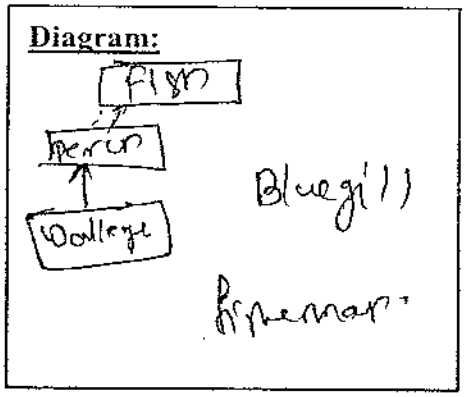
A. Alpha a = x;
B. Foo f = (Delta)x;
C. Foo f = (Alpha)x;
D. Beta b = (Beta)(Alpha)x;

Solution:

Q21) Given:

Hint:- every Object of Sub class is also an object of Super class, and not vice versa.

```
3. interface Fish { }
4. class Perch implements Fish { }
5. class Walleye extends Perch { }
6. class Bluegill { }
7. public class Fisherman {
8. public static void main(String[] args) {
9. Fish f = new Walleye();
10. Walleye w = new Walleye();
11. Bluegill b = new Bluegill();
12. if(f instanceof Perch) System.out.print("f-p ");
13. if(w instanceof Fish) System.out.print("w-f ");
14. if(b instanceof Fish) System.out.print("b-f ");
15. }
16. }
```



What is the result?

instanceof → object

Options:

A. w-f
B. f-p w-f
C. w-f b-f
D. f-p w-f b-f
E. Compilation fails.
F. An exception is thrown at runtime.

Solution: