Dt : 20/9/2023

faq:

define sealed classes?(Java17 - new feature)

=>The classes which are declared with "sealed" keyword are known as sealed classes introduced by Java17 version.

=>These sealed classes specify which classes must be extened,which means sealed classes will give permission for the classes for extention.

=>The classes which are extended from sealed classes must be final classes or non-sealed classes or sealed classes.

syntax:

sealed class Class_name permits EClass1,EClass2,...

{

//Class_body

}

Ex:

p1 : A.java

```
package p1;
public sealed class A permits B,C,D{
  public void mA() {
       System.out.println("---mA()---");
   }
}
```

p1 : B.java

```
package p1;
public final class B extends A{
 public void mB() {
      System.out.println("---mB()---");
 }
}
```

p1 : C.java

```
package p1;
public non-sealed class C extends A{
   public void mC() {
        System.out.println("---mC()----");
   }
}
```

p1 : D.java

```
package p1;
public sealed class D extends A permits E{
   public void mD() {
        System.out.println("---mD()----");
   }
}
```

p1 : E.java

```
package p1;
public non-sealed class E extends D{
  public void mE() {
       System.out.println("---mE()----");
  }
}
```

p2 : DemoPoly8.java(MainClass)

```
package p2;
import p1.*;
public class DemoPoly8 {
     public static void main(String[] args) {
        System.out.println("****Class-B****");
        B ob1 = new B();
        ob1.mA();
        ob1.mB();
        System.out.println("****Class-C****");
        C ob2 = new C();
        ob2.mA();
        ob2.mC();
        System.out.println("****Class-E****");
        E ob3 = new E();
        ob3.mA();
        ob3.mD();
        ob3.mE();
     }
}
```

o/p:

*****Class-B*****

---mA()---

---mB()---

*****Class-C*****

---mA()---

---mC()----

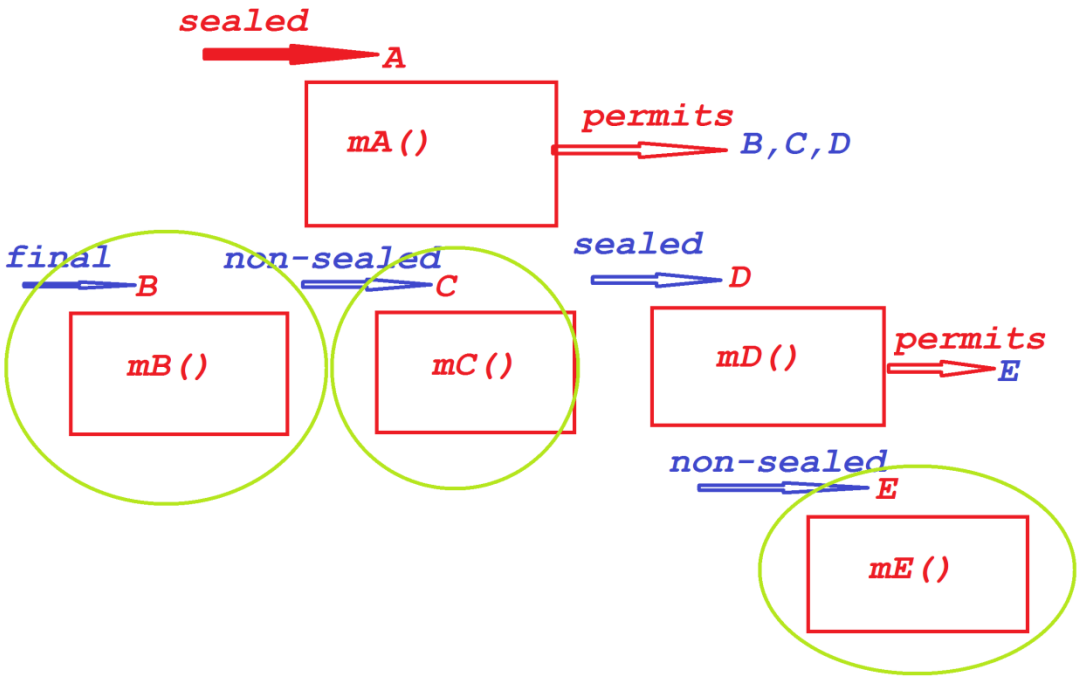
*****Class-E*****

---mA()---

---mD()----

---mE()----

Diagram:



==========

faq:

define sealed Interfaces?(Java17 - new feature)

=>The Interfaces which are declared with "sealed" keyword are known as sealed Interfaces introduced by Java17 version.

=>These sealed interfaces specify which classes must be implemented,which means sealed Interfaces will give permission for the classes for Implementation.

=>The classes which are implemented from sealed Interfaces must be final classes or non-sealed classes or sealed classes.

Ex:

p1 : ITest.java

```
package p1;
public sealed interface ITest permits X,Y,Z{
  public abstract void m(int a);
}
```

p1 : X.java

```
package p1;
public final class X implements ITest{
    public void m(int a) {
     System.out.println("===Class-X m(a)====");
     System.out.println("The value a:"+a);
     }
}
```

p1 : Y.java

```
package p1;
public non-sealed class Y implements ITest{
  public void m(int a) {
       System.out.println("====Class-Y m(a)====");
       System.out.println("The value a:"+a);
  }
}
```

p1 : Z.java

```
package p1;
public sealed class Z implements ITest permits P  {
```

```
    public void m(int a) {
     System.out.println("====Class-Z m(a)=====");
     System.out.println("The value a:"+a);
     }
}
```

p1 : P.java

```
package p1;
public final class P extends Z{
   public void mP(int b) {
        System.out.println("====Class-P mP(b)=====");
        System.out.println("The value b:"+b);
   }
}
```

p2 : DemoPoly9.java(MainClass)

```
package p2;
import p1.*;
public class DemoPoly9 {
    public static void main(String[] args) {
       X ob1 = new X();
       ob1.m(11);
       Y ob2 = new Y();
       ob2.m(12);
       P ob3 = new P();
       ob3.m(13);
       ob3.mP(13);
    }
}
```

o/p:

===Class-X m(a)====

The value a:11

====Class-Y m(a)====

The value a:12

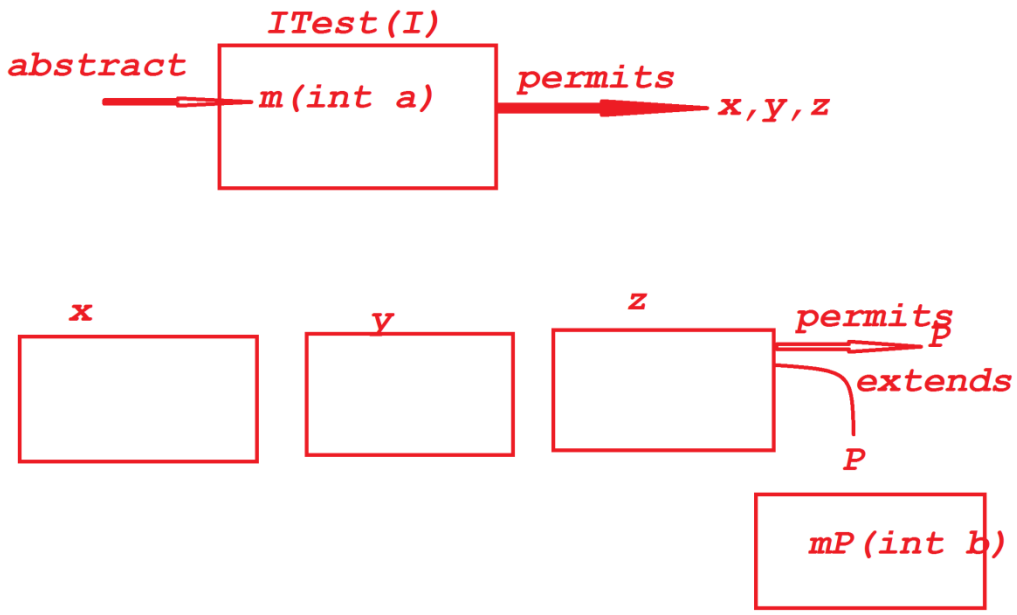
====Class-Z m(a)=====

The value a:13

====Class-P mP(b)=====

The value b:13

Diagram:



==========

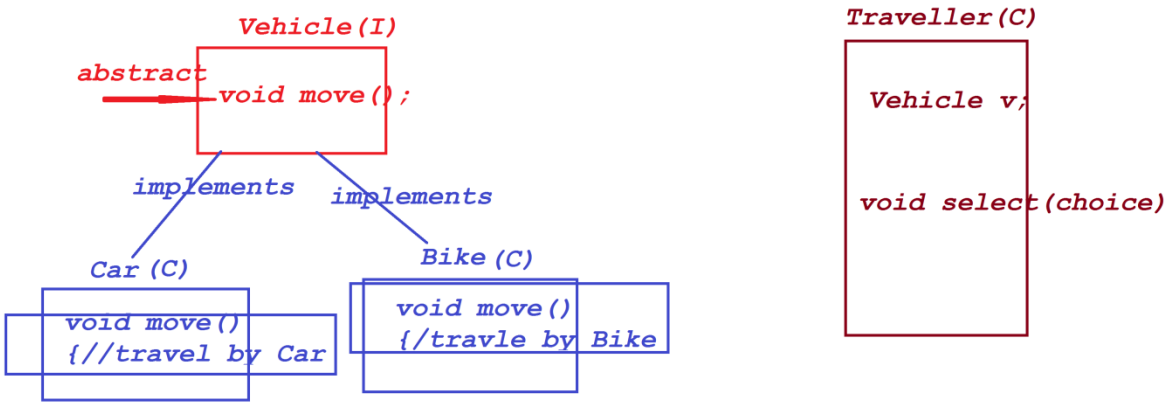
=

Note:

=>Method Overriding process comes under Dynamic PolyMorphism,because

same method signature can have many forms at execution stage.

Layout:



Ex:

p1 : Vehicle.java

```
package p1;
public interface Vehicle {
    public abstract void move();
}
```

p1 : Car.java

```
package p1;
public class Car implements Vehicle{
     @Override
    public void move()  {
         System.out.println("****Car-move()****");
         System.out.println("Travel by Car....");
    }
}
```

p1 : Bike.java

```
package p1;
public class Bike implements Vehicle{
     @Override
        public void move() {
             System.out.println("****Bike-move()****");
             System.out.println("Travel by Bike.....");
        }
}
```

p1 : Traveller

```
package p1;
public class Traveller {
     public Vehicle v;//Reference variable
     public void select(int choice) {
     switch(choice ) {
     case 1:
          v = new Car();
          v.move();
          break;
     case 2:
          v = new Bike();
          v.move();
          break;
     default:
          System.out.println("Invalid Choice...");
     }
     }
}
```

p2 : DemoPoly9.java(MainClass)

```
package p2;

import java.util.*;

import p1.*;
```

```
public class DemoPoly9 {
	public static void main(String[] args) {
	Scanner s = new Scanner(System.in);
	try(s;){
	  try {
		Traveller t = new Traveller();
		System.out.println("****Choice****");
		System.out.println("\t1.Car"
				+ "\n\t2.Bike");
		System.out.println("Enter the Choice:");
		int choice = s.nextInt();
		t.select(choice);
	  }catch(Exception e) {
		e.printStackTrace();
	  }
	}//end of try with resource
	  }
}
```

==========

Note:

=>Through Method Overloading process we can provide same method with

different forms at compilation stage,because of this reason Method Overloading process comes under static PolyMorphism.

Ex:

```
package p1;
public class Addition
{
   public void add(int x,int y){}
   public void add(int x,int y,int z){}
   public void add(int x,float y){}
}
```

add() method is having three forms:

- *add(int,int)*
- *add(int,int,int)*
- *add(int,float)*

==========