Interface

⇒ Interfaces are not classes, Hence objects of interface cannot be created.

Eg:-

```
interface A {

    void show();

}

public class Main {

    A obj = new A();

}
```

This is wrong X.

→ The methods inside an interface are public abstract by default, Hence they cannot be defined inside interface.

Eg:- interface A {

public abstract void show();

}

default inserted

→ The variables / data items inside Interface are final and static. Hence they must be initialized inside interface.

Eg:- interface Ad

```
int data;  → Wrong X
int data=0; → correct ✓
}
```

→ Since data items are final and static. They cannot be changed and they can be accessed directly from the name of interface, without creating an object.

Eg:- public class Main{

psvm(){

```
A.data=1;  → wrong as data is final. and data already initialized to 0
sout(A.data);
```

output:- 0. }}

→ We can implement multiple interfaces in single class. (Multiple Inheritance)

Example

```
interface A{
    void showA();
}

interface B{
    void showB();
}

public class Main implements A, B
{
    void showA()
    {
        sout("HiA");
    }
    void showB()
    {
        sout("HiB");
    }
}
```

Note:- Compulsorily implement all the methods declared in interfaces implemented.

→ For class to class → extends
interface to interface → extends.
Implementing interface in class → implements

Enums

→ Special type of class, where we define named constants.

Syntax:-

```
enum Status {
Running, Failed, Pending, Success;
}

public class Main{

psvm(){

Status s = Status.Success;
sout(s); output:- Success

sout(s.ordinal()); → output:- 3
                  ↳ gives index
Status[] s1 = Status.values();
                  ↓
              returns array of constants
```

```
for( Status s : s1 )
{
        sout(s); -> Output:- Running
                            Failed
                            Pending
                            Success.
}
}
```

Enum with If and Switch

```
public class Main {

    psvm(){
        Status s = Status.Pending;

    if (s == Status.Running)
            sout("Running");

    else if( s == Status.failed)
            sout(" ");

    " "

}
}
```

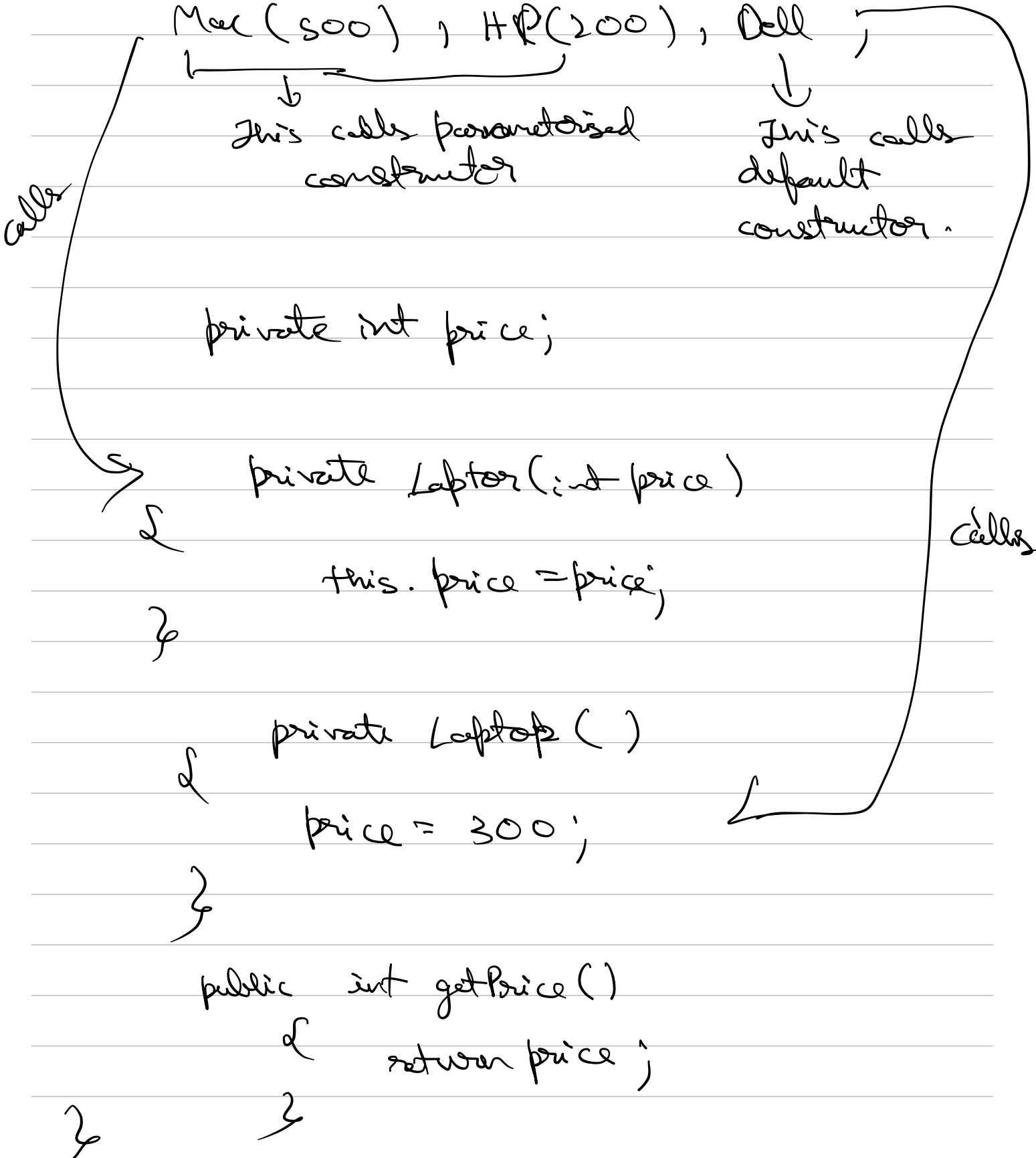
output:- Pending.

Enum with switch :-

```
public class Main {
    psvm()
    {
        Status s = Status.Success;

        switch (s) {

        case Running:
                sout (" Upand Running");
                break ;

        case Faibd:
                sout (" sorry faibd");
                break;

        "       "

        case Success:
                sout (" Yep success");
                break;
    }
}
```

Output :- Yep Success.

Using Constructors inside Enum Class :-



```
public class Main{
    psvm(){

        Laptop obj = Laptop.Mac;

        sout( obj + ":" + obj.getPrice());
```

Output:-

Mac : 500

```
}
```

Types of Interface

- Normal
- Functional / SAM
- Marker.

Normal:-

Interfaces with two or more methods.

Ex:-

```
interface A{
    void showA();
    void showB();
}
```

SAM:- (Single Abstract Method)

Eg:- interface Ad

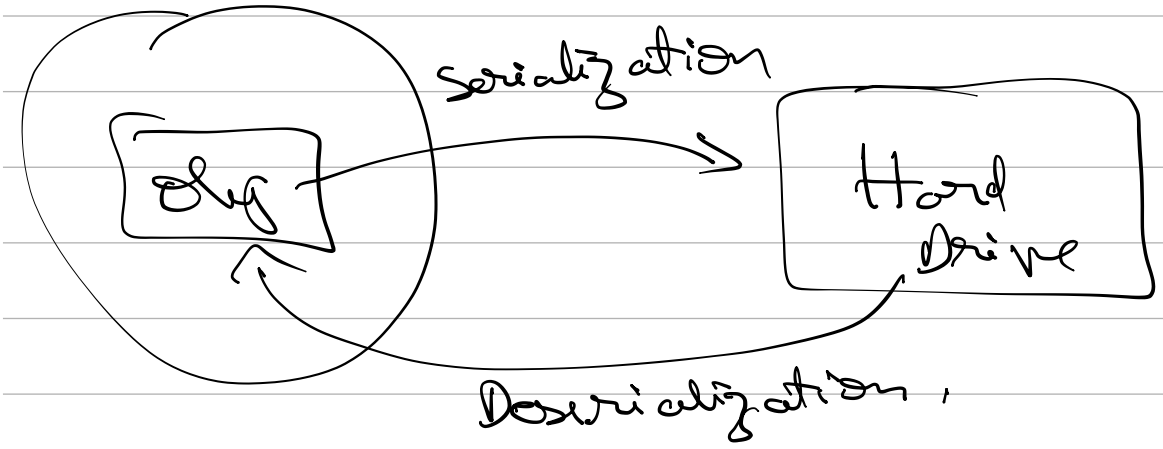
{

void showA();

}

→ Only single method.

Marker:- These interfaces are used in serialization and deserialization.



Serialization:- Storing Object in harddrive before destroying object.

Deserialization:- Restoring object from harddrive.

Functional Interface | SAM :-

→ To make sure we are using Functional Interface we use annotation

@FunctionalInterface.

Eg:-

```
@FunctionalInterface
interface A
{
    void show();
}

public class Main{

    psvm(){

        A obj = new A(){

            public void show()
            {
                sout("In A show");
            }
        };

        obj.show();
    }
}
```

object of interface can be done with creation of anonymous class.

Inner anonymous class helps to define show() function.

Output:-

In A show.

Note:- Implementation of show() function can be done by another class by implementing 'A' interface also.

Here we used anonymous class..

Lambda Expression

-> In the above code.

```
interface A {
        void show();
}
public class Main {
    psvm()
    {
        A obj = () ->
        {
            sout("In A");
        };
        sout(obj.show());
    }
```

Output:-

In A.

→ Here we use '→' symbol instead of creating an object.

This symbol (→) is called Lambda Expression.

⚠ LAMBDA EXPRESSION CAN BE USED ONLY FOR FUNCTIONAL INTERFACES.
BECAUSE we define only one function. in a block { } after →.

→ If we are having parameters in that one function.

Suppose.

```
psvm(){
    A obj = (int i) -> {
        Sout("Number:"+i);
    };
    obj.show(5);
}
```

(A ← interface)

Output :-

Number : 5

NOTES :- We can also pass more parameters to method in SAM method

Eg :-

```
interface Ad
{
    void show (int i, int j);
}

psvm()
{
    A obj = (i, j) ->
    {
        sout ( i + " " + j );
    };

    obj. show (5, 10);
}
```

Output :-

5 10.

LAMBDA EXPRESSION WITH RETURN :-

```
@Functional Interface
interface A {

    int add(int i, int j);

}

public class Main {

    psvm()
{
    A obj = (i,j) ->
    {
        return i+j;
    };

int ans = obj.add(5,10);
          sout(ans);
}
}
```

Another method.

```
psvm(){
A obj = (i,j) ->
          i+j;
int ans = obj.add
          (5,10);
sout(ans);
}
```

Output:-

15.