D2D ⇒ HW. Consistency

(1%)

Peformance eq.

Performance = Potential − Interference

↑ ↓

support @ scaler.com
slack

Dr APJ Abdul Kalam

Don't ask questions untill you fail to find the answer.

"what if... ?" ⟹
- → Write
- → Try (30 min)
- → Google (SOF) ⇒ 15-20
- Slack → DM

DSA → CS Fund → LLD
↳ DBMS

1 - 1.5 Months

- OOP Basic ✓
- SOLID ✓
- Design Pattern → Creational, Structural, Behavioural → 3-4
- UML
- How to approach a LLD problem.
- Case Studies → Pen
  → Game
  → Parking lot
  → Splitwise
  → Book My Show
- Database schema design
- Hands on Classes (Machine Coding)

10-12%

Easily,

- ✓ Maintainable
- ✓ Understandable
- ✓ Extensible.

LLD

88-90%

Maintenance ←
- Testing
- Find Bug
- Fix Bug

Understandability ←
- Understand other code
- PR Review
- KT Session.

Extensibility ←
- Regression Issue
- Merge Conflicts.

```
int add (a, b) {
    ret a+b;
}
```

Functional

Procedural

```
main ()
{
    print (add (5,3));
}
```

① ② ③

```
Start (Car c) {
}
```

```
Car.start ()
```

OOP

- Inheritance
- Abstraction
- Polymorphism
- Encapsulation

- Interface & abstract class

0 → 1 → 2

0 — Scratch — Kid

1 — Knows but not sure — Adult

2 — Master — Legend.

* Software that represents a Bird

Bird
- Properties
- Actions / Behaviour

```
Bird hen = new Bird();
```
↳ Constructor

```
hen.wt;      hen.type = "hen";
hen.fly();
Bird eagle = new Bird();
eagle.fly();
```

```
void fly() {
    if (type == "hen") {
        // fly like a hen;
    }
    else if (type == "eagle")
        // fly like an eagle;
    }
}
```

+48 else-if
+1 else-if

→ Template / Blue Print

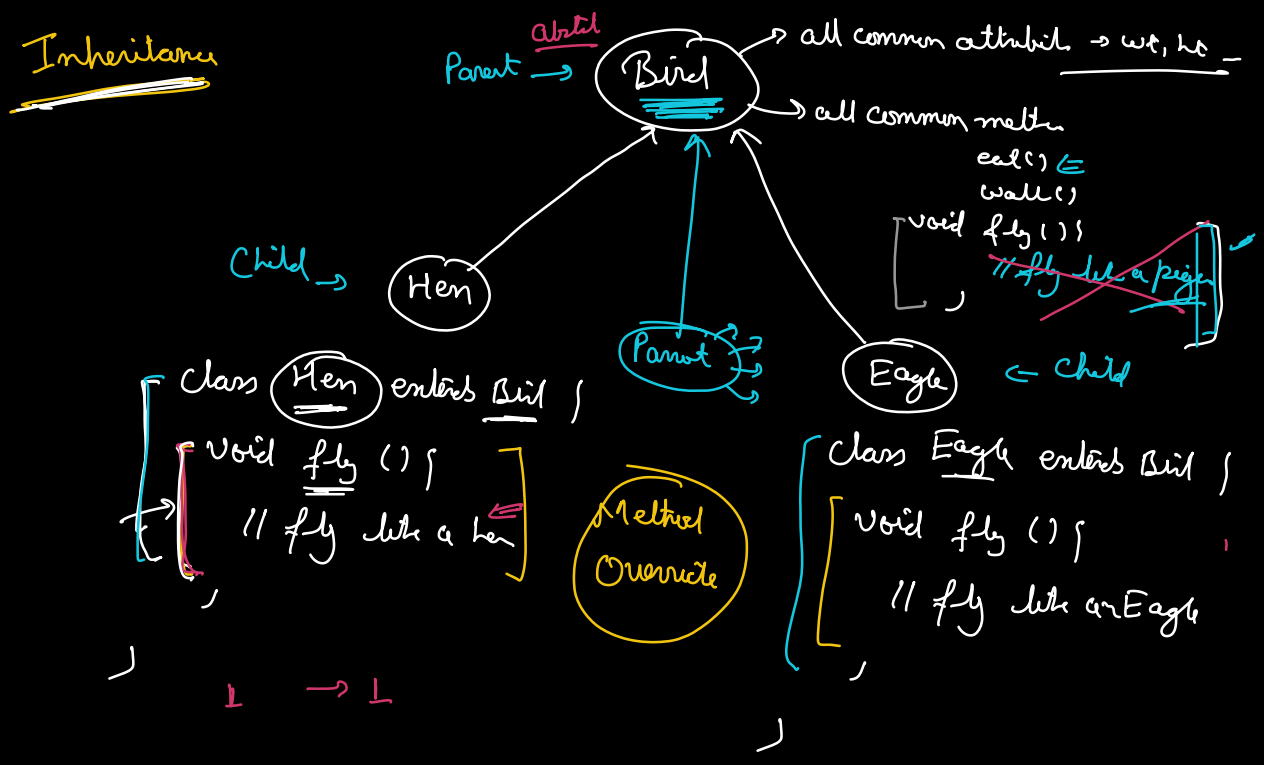
```
Class Bird {
// Attributes
double wt, ht;
String color, type;
:
//Method
void eat() {
  // ----
}
void walk() {
  // ----
}
void fly() {
  // ---
}}
```

Maintainable x

Understandable.

Extensible ⟹ life

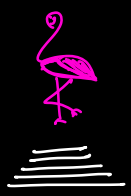
eat() two



Angry Bird

Tech lead

Add Flamingo



```
class Flamingo extends Bird {
    fly() {
    }
}
```

```
Flamingo f = new Flamingo();
f.fly();
```

→ fly like peiger

1. No implementation of fly() in Bird is required.
2. Creating object of Bird should not be allowed.
3. All child classes of Bird must be forced to implement fly().

Bird → All common attributes
→ All methods with common implementation (die())
→ All common methods with diff implementation

Should be either
Wether default implementation

```
abstract class Bird {
    abstract fly();
}

class Hen extends Bird {
    void fly() {
        // - - -
    }
}
```

```
int add(int a, int b) {
    ret a+b;
}

double add(double a, double b) {
    ret a+b;
}

String add(String a, String b) {
    ret a+b;
}
```

```
main() {
    double a=1.5, b=2.5;
    add(a, b);
    add("abc", "xyz");
}
```

Method Overloading
[Compile time]

Client

Class Seller {

Void Sell (Bird b) {

}

}

Bird Package
- Hen
- Eagle
- Flamingo
- Parrot ..
- Vulture

Sell (Hen h)
Sell (Eagle e)
Sell (Flamingo f)
;
Sell (Vulture v)

⇒ Every bird object should be able to tell its price.

⇒ Price of diff types of birds should depend on diff features/attributes

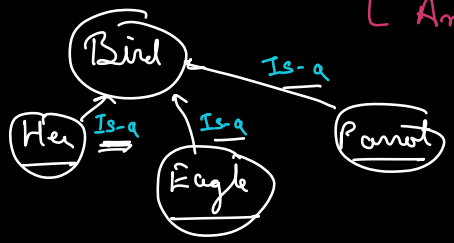
Hen
- Color
  - Black → 2500/-
  - Any other → 5000/-

Eagle
- Wing Span
  - > 1m → 10K
  - ≤ 1m → 7500/-

Flamingo
- Age
  - < 3y → 17000/-
  - ≥ 3y → 12000/-

⇒ Complete detail of ⇑

[How many birds are there ?
  Any new addition ?
  Any deletion ?]




```
Hen h = new Hen();
Sell(h);
```

Class Seller {

void sell (Bird b) {

b.getSP();

}

}

Abstraction

Hen h = new Hen();

Eagle e = new Eagle();

sell(h);

sell(e)

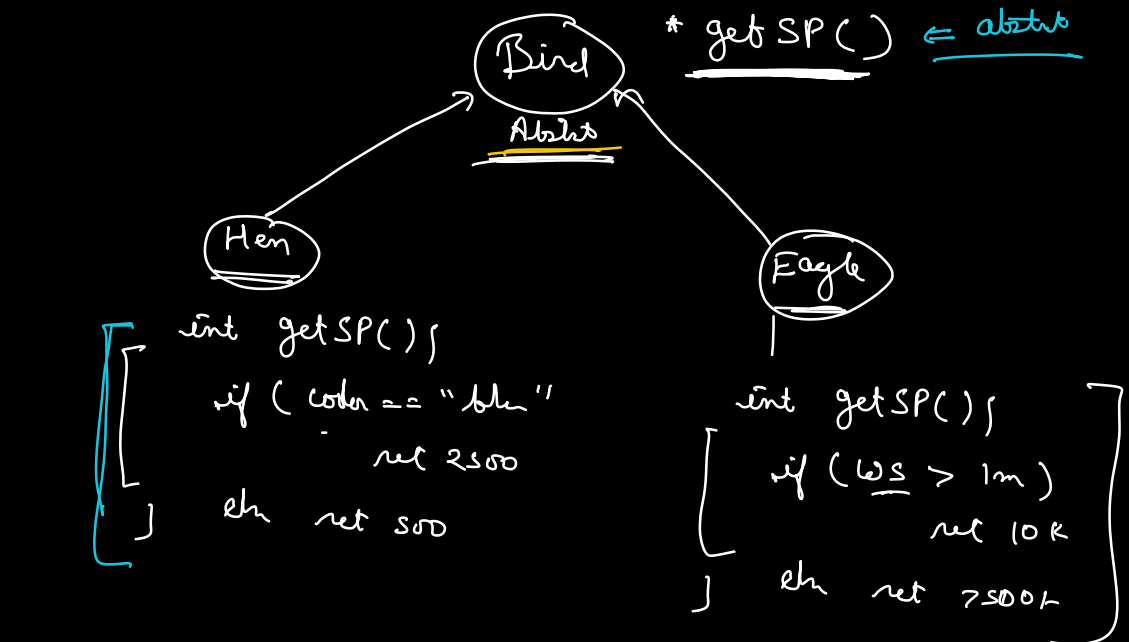
sell (h) {

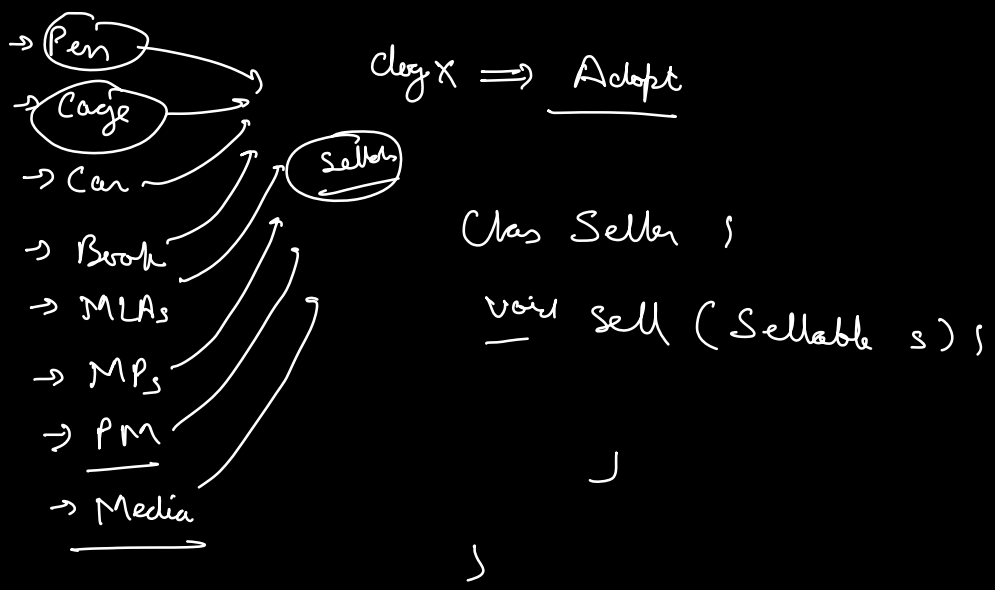
Hen h = new Hen();

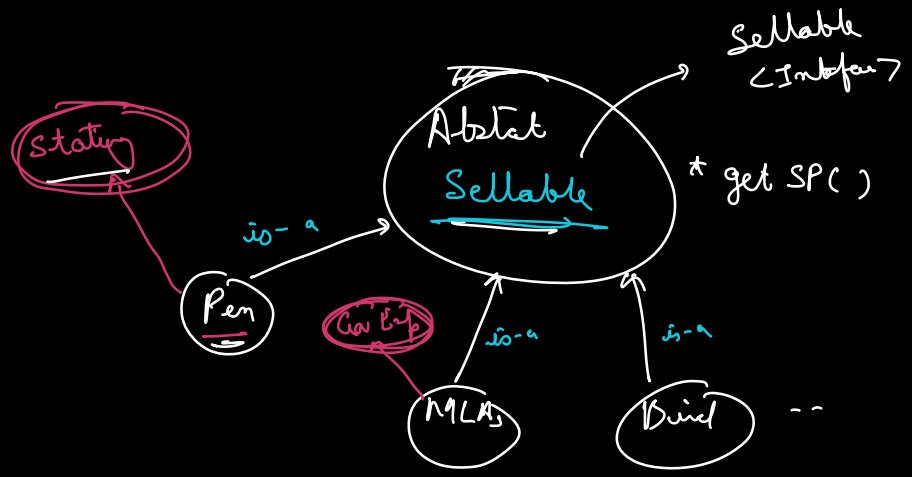
Factory

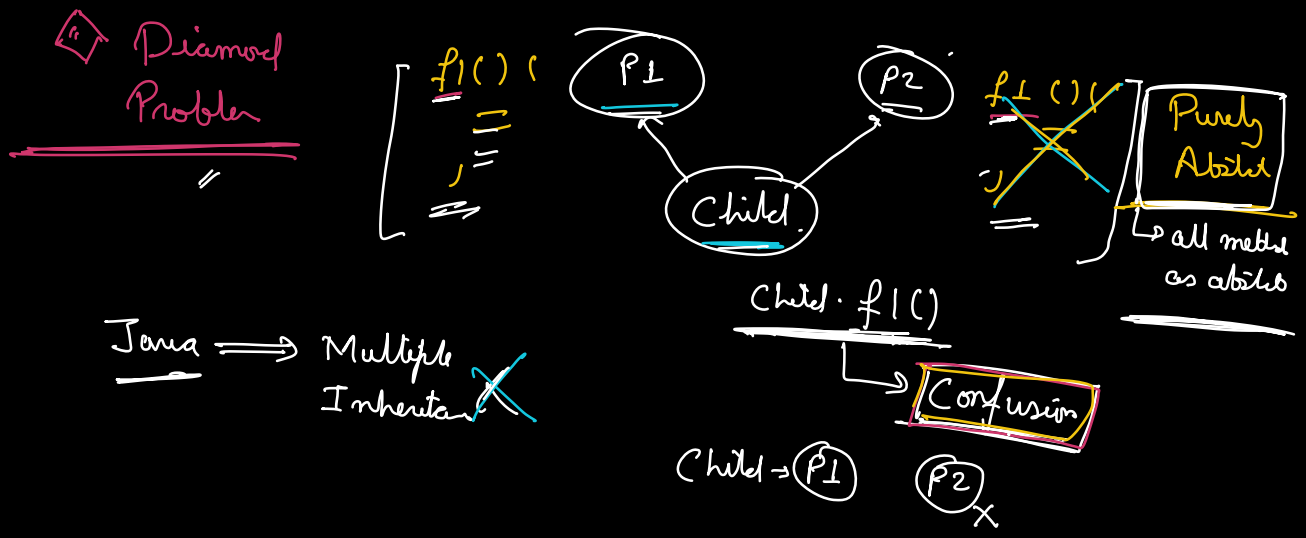
DI

}




Pen
Cage
Car
Books
MLAs
MPs
PM
Media
Sellable
Clas Seller ;
void sell (Sellable s);
}


Sellable
<Interface>
Abstract
Sellable
* getSP()
Static
is-a
Pen
Car
is-a
is-a
MLAs
Bird
--


Diamond
Problem
f1() {
}
P1
P2
Child
f1() {
}
Purely
Abstract
→ all methods as abstract
Child.f1()
Confusion
Child → P1
P2
Java ⟹ Multiple Inheritance

Interface

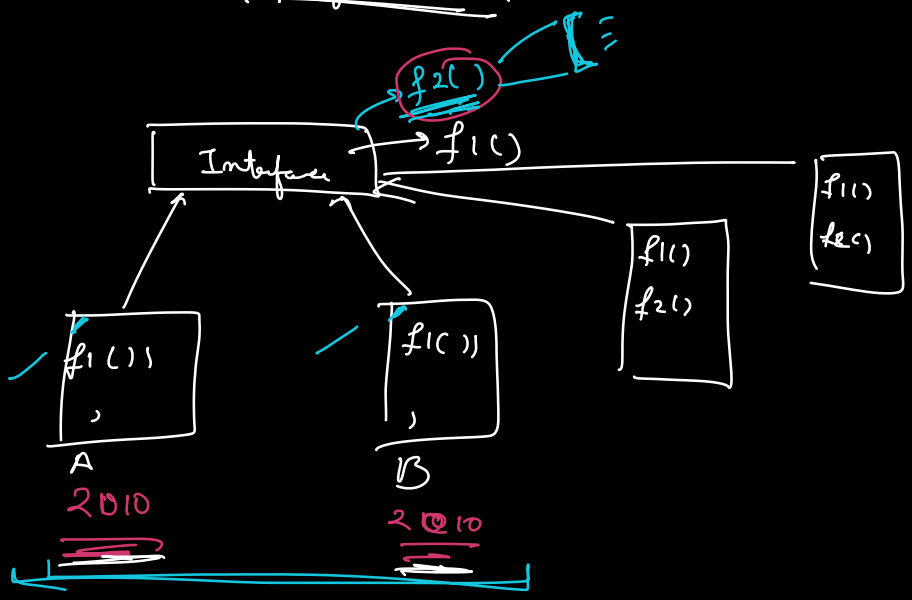
```
interface Sellable {
    int getSP();
}

Class Pen Impl Sellable {

}

Sell(Sellable s) {

}
```

ref

Java 8

↳ Default impls ⇒ Backward compatibility

↳ Specific use case



If it works → don't touch it.