Low level design Basics

Thursday, 12 August 2021 9:00 PM

#BBM

"Don't ask till you fail to find the answer"

"What if ..... ?"

- → Write down.
- → Write code & observe the results.
- → Find the reason.

LLD

1) Basics OOP
2) SOLID
3) Creational
4) Structural
5) Behavioural

} DP

6) UML
7) Pen
8) Parking lot
9) Tic-tac-toe / Snake & Ladder
10) Splitwise
11) BookMyShow
12) Database Schema design 1
13) Database schema design 2

} Machine Coding

13 x 2.5 = 33h

2.5h

10-12%

88-90% →

1. Maintain
2. Transparent
3. Extensible

Today
Bug fixes
Refactor
Understand
Review PRs
Explain (KT) → TOT → Knowledge Transfer
Jun 7 → June 8
Merge
SVN [Conflicts]
Adding new feature

Meetings
Lunch Break (AFK)
TT
IT returns
Chai Sutta
Scaler Ass / HW

Procedural

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

main(){
    print(add(5, 6));
}
```

1. Inheritance
2. Abstraction
3. Polymorphism
4. Encapsulation

SOLID
- SRP
- O/C P

Design a Software representing a Bird

99%

Bird
- Properties: color, wt, ht
- Behavior: fly, eat, chirp

```
Bird hen = new Bird();
hen.wt;
hen.fly();

Bird eagle = new Bird();
eagle.fly();
```

```
class Bird {
    String color;
    double wt, ht;
    String type;
    void eat(){
        // ____
    }
    void chirp(){
        // ____
    }
    void fly(){
        if(type == "hen"){
            // fly like a hen;
        }
        else if(type == "eagle"){
            // fly like a eagle
        }
    }
}
```

→ Attributes

→ Methods

100+

+100

Single Responsibility Principle

Tech
- Ops
- CR
- Run
- UI
- Proch
- Pay
- IPrac
- Search

Abstract

Bird
- → Att: wt, ht, color
- → Methods: fly(), eat()
  - fly() → abstract

Extends

Concrete

- Hen (is-a)
- Parrot (is-a)
- Eagle (is-a)

Open / Close principle
- Open → extension
- Close → Modification

```
class Hen extend Bird {
    // override
    void fly(){
        // fly like hen;
    }
}

Hen h = new Hen();
h.fly();
```

```
class Eagle extend Bird {
    void fly(){
        // fly like eagle
    }
}
```

c.fly()

Angry Bird

Add flamingo

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

```
Flamingo f = new Flamingo();
f.fly();
    → fly like flamingo
```

Silent Killers

Method Overloading

Compile time Poly

```
int add(int a, int b){
}

double add(double a, double b){
}

Str add(Str a, Str b){
}
```

```
double a = 1.75, b = 2.38;
add(a, b);

Str a = "abc", b = "def"
add(a, b)
```

```
abstract class Bird {
    void abstract fly();
    int abstract getSellingPrice();
}

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

    int getSellingPrice(){
        if(this.color == "black")
            ret 1000;
        else ret 250;
    }
}
```

Kadaknath

```
class Seller {
    void sell(Hen h){
    }
    void sell(Flam f){
    }
    void sell(Eagle e){
    }
}
```

Hen → Color
Flam → Age
Eagle → Sub Sp

```
void sell(Bird b){
    b.getSellingPrize();
}
```

Run time Polymorphism

Interface

Bird
- TV
- Car
- House

```
class Seller {
    void sell(Object b){
        b.getSellingPrize();
    }
}
```

Problem

Sellable ← Eagle, Hen → Bird, Car, TV

```
void sell(Sellable s){
    s.getSellingPrize();
}
```

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

class TV implements Sellable {
    int getSellingPrize(){
    }
}

class Bird implements Sellable {
}
```

Bird → attr

- Hen: fly()
- Eagle: fly()

Interface
- Hen
- TV
- House

Pass by ref

```
fn(&Arr[]){
    Arr[0] = 1;
}

main(){
    fn(Arr);
}

Arr[0];
```

Pass by val

```
fn(Arr[])
    Arr[0] = 1;
```

```
abstract class Bird implements Sellable {
    int getSelly(){
    }
}

class Hen extend Bird {
}
```