9__30-07-2022

Saturday, 30 July 2022 8:06 PM

Dependency Inversion Principle (DIP)

DIP states that high-level modules | classes should not depend upon low-level modules | classes.

Both should depend upon level of abstractions.

A high-level module | class that has a dependency on low-level modules | classes or some other class and knows a lot about other classes it interacts with, is said to be tightly coupled.

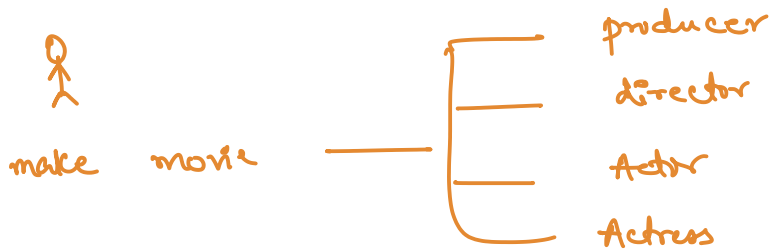
When a class knows explicitly about the design and implementation of another class,

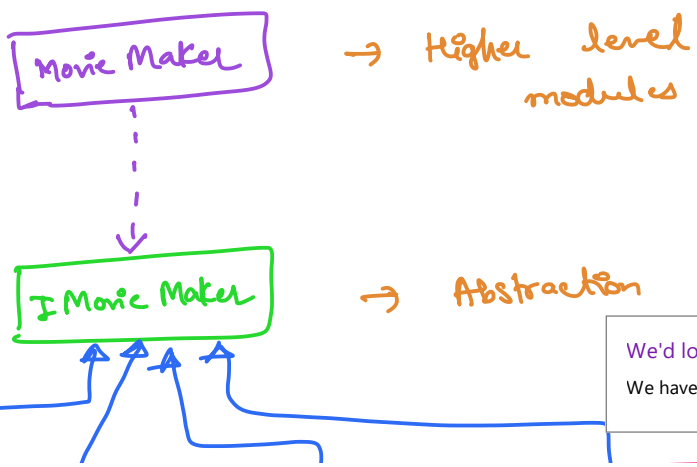
it raises the risk that changes

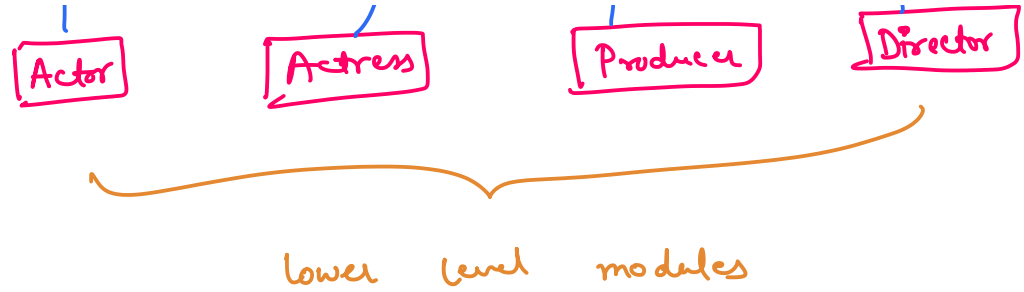
class will break the other class.

So we must keep these high-level and low level modules | classes loosely coupled as much as we can.

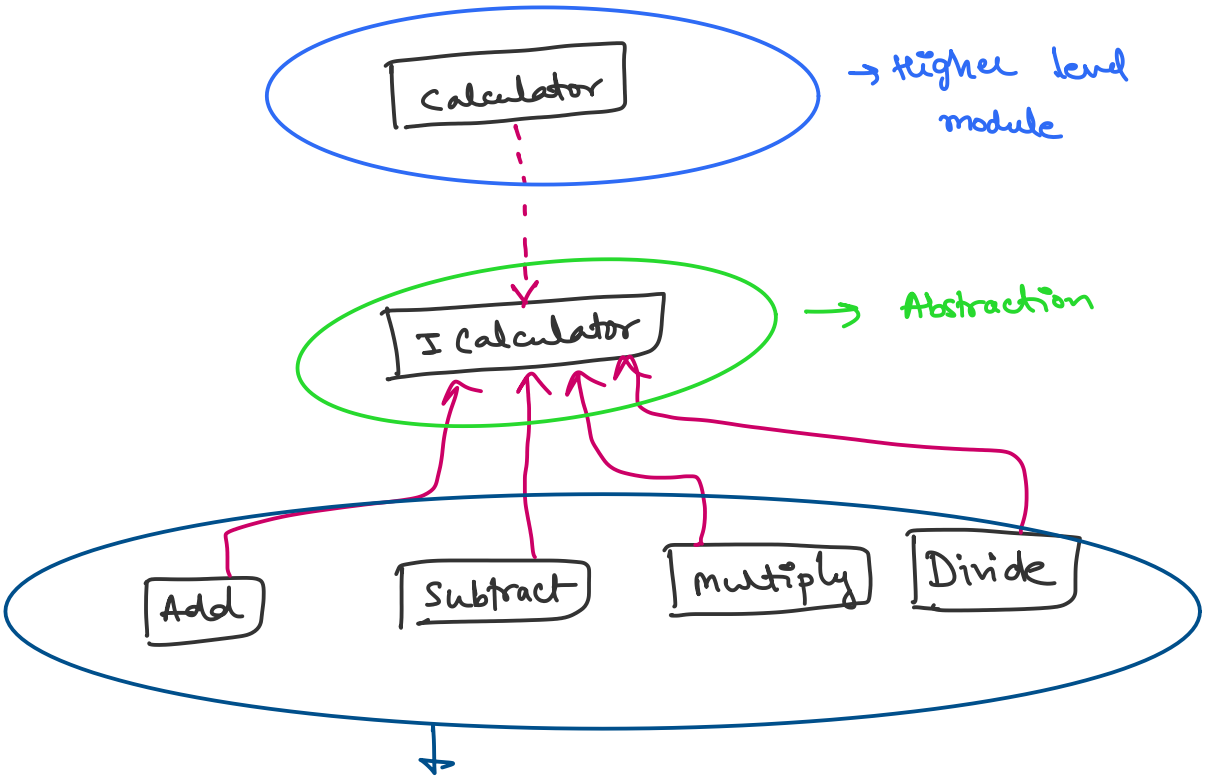
To do that, we need to make both of them dependent upon abstractions instead of knowing each other.







Another eg —



Dependency Inversion

↳ focused on how to structure your code.

Dependency Injection

↳ focused about how the code functionally work

```
public interface IMovieMaker {
    public void hirePeople();
}


public class Actor implements IMovieMaker {
    @Override
    public void hirePeople() {
        System.out.println("Hire Actor");
    }
}


public class Producer implements IMovieMaker {
    @Override
    public void hirePeople() {
        System.out.println("Hire Producer");
    }
}


public class Actress implements IMovieMaker {
    @Override
    public void hirePeople() {
        System.out.println("Hire Actress");
    }
}


public class Director implements IMovieMaker {
    @Override
    public void hirePeople() {
        System.out.println("Hire Director");
    }
```

```
}


public class Choreographer implements IMovieMaker {
   @Override
   public void hirePeople() {
      System.out.println("Hire Choreographer");
   }
}




public class MovieMaker {
   public void startMovieHiring(IMovieMaker movieMaker)
   {
      movieMaker.hirePeople();
   }
}




public class Program {
   public static void main(String[] args) {
      MovieMaker movieMaker = new MovieMaker();
      movieMaker.startMovieHiring(new Actor());
      movieMaker.startMovieHiring(new Actress());
      movieMaker.startMovieHiring(new Director());
      movieMaker.startMovieHiring(new Producer());
      movieMaker.startMovieHiring(new Choreographer());
   }
}
```

Output:

```
Hire Actor
Hire Actress
Hire Director
Hire Producer
Hire Choreographer
```

Design

Design Principles

↓

Majorly theory, no concrete example

↓

What should we do?

Design patterns

↓

Majorly about implementation

↓

How we should do?

Design patterns

- Reliable
- scalable
- maintainable

↳ Design level solutions for recurring problems that we software engineers come

often.

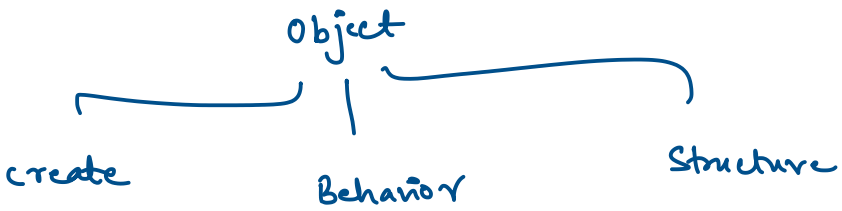
- It is like a description on how to tackle these problems and design a solution.
- Design patterns are reusable sol^n's to the problems that we encounter in the day to day programming.
- Design patterns act as a template which can be applied to the real-world programming problems.

History and evolution of design patterns

Book - Elements of reusable object - oriented software.

Gang of four → 4 authors

- Gamma
- Helm
- Johnson
- Vlissides



Types of design patterns

- Creational design pattern
- Structural design pattern
- Behavioral design pattern.

Creational design pattern

Is These patterns are designed for class instantiation.

↳ They can be either class - creation patterns or object - creational patterns.

↳ It gives the program more flexibility in deciding which object needs to be created for a given case.

Types of creational design patterns —

1. Singleton pattern
2. Factory pattern
3. Factory method pattern
4. Abstract factory pattern
5. Builder pattern
6. Prototype pattern
7. Object pool pattern.

Structural design pattern

- These design patterns are designed with regards to a class's structure and composition.

- Major focus area
  - Decoupling interfaces
  - Implementation of classes and its objects

- The main goal of most of these patterns is to increase the functionality of the classes involved without changing much of its composition.

Types of structural design patterns—

1. Adaptor pattern
2. Bridge pattern
3. Composite pattern
4. Decorator pattern
5. Facade pattern
6. Flyweight pattern
7. Proxy pattern.

Behavioral design pattern

↳ These patterns are designed depending upon on how one class communicates with others and between classes and objects.

Types of behavioral design patterns —

1. Command pattern
2. Interpreter pattern
3. Iterator pattern
4. Mediator pattern
5. Memento pattern
6. Observer pattern
7. State pattern
8. Strategy pattern
9. Template pattern
10. Visitor pattern
11. Chain of responsibility pattern
12. Null object pattern