Dependency Inversion Principle

- Program to an interface and not to an implementation
    - Any problem?

Button	Lamp
+ Poll()	+ TurnOn() + TurnOff()

"HIGH LEVEL MODULES SHOULD NOT DEPEND UPON LOW LEVEL MODULES, BOTH SHOULD DEPEND UPON ABSTRACTIONS.

ABSTRACTIONS SHOULD NOT DEPEND UPON DETAILS. DETAILS SHOULD DEPEND UPON ABSTRACTIONS."

No variable should hold a reference to a concrete class.

No class should derive from a concrete class.

No method should override an implemented method of any of its base classes.

It is OK to depend on stable classes like String, Integer, JPanel, etc.

Avoids designs that are rigid, Fragile and Immobile.

Example: We have a classes: Copy, Keyboard, Printer. The Copy reads from keyboard and prints to the printer. If we use DIP, we have an abstraction for Keyboard and Printer. So we can add input and output devices later on.

Example: Collections in Java.

Example: Customer is a class. Employee is a class. Employees are now allowed to purchase on credit. We need an interface Buyer.

For every variable use the maximum abstract type possible.

1

DIP implemented
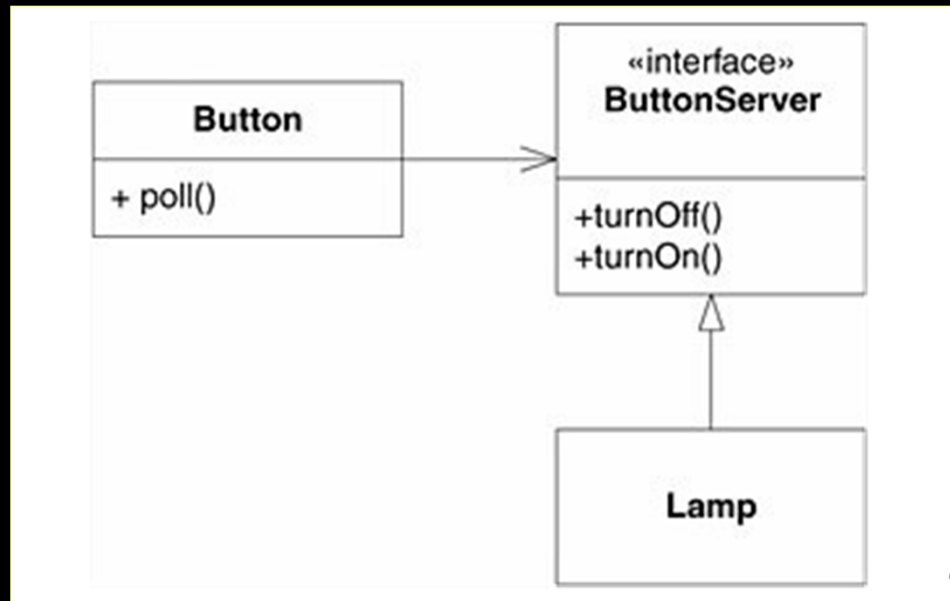
16-Feb-23 5:33 PM   https://vijaynathani.github.io/   2

Advantages

   Clients are unaware of the specific class of the object they are using

   One object can be easily replaced by another

   Object connections need not be hardwired to an object of a specific class, thereby increasing flexibility

   Loosens coupling

   Increases likelihood of reuse

   Improves opportunities for composition since contained objects can be of any class that implements a specific interface
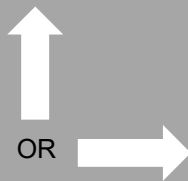
Disadvantages

   Modest increase in design complexity

 Q96, Q94, Q93, Q91 - DIP

Which is Better?

```
Class Service {
    //…
}
class Client {
    //…
    Service s =
        new Service();
    //…
}
```

OR →

```
Class Service {
    private Service(){}
    static Service
        getInstance() {
        return
            new Service();
    }
    //…
}
class Client {
    //…
    Service s = Service.
        getInstance();
    //…
}
```

The right side is better. Service class can be made abstract tomorrow.

Which is Better (C++)?

```cpp
class Service {
    //...
};
class Client {
public:
  void someFunction() {
    Service* s = new
Service();
      //...
    delete s;
  }
};
```

OR

```cpp
class Service {
  Service() {}
  public: static Service*
   getInstance();
  Service(const Service&
  s) = delete; //...
};
Service*
  Service::getInstance(){
    return new Service();}
class Client {  public:
  void someFunction() {
    Service* s =
    Service::getInstance();
    //.. delete s at end}};
```
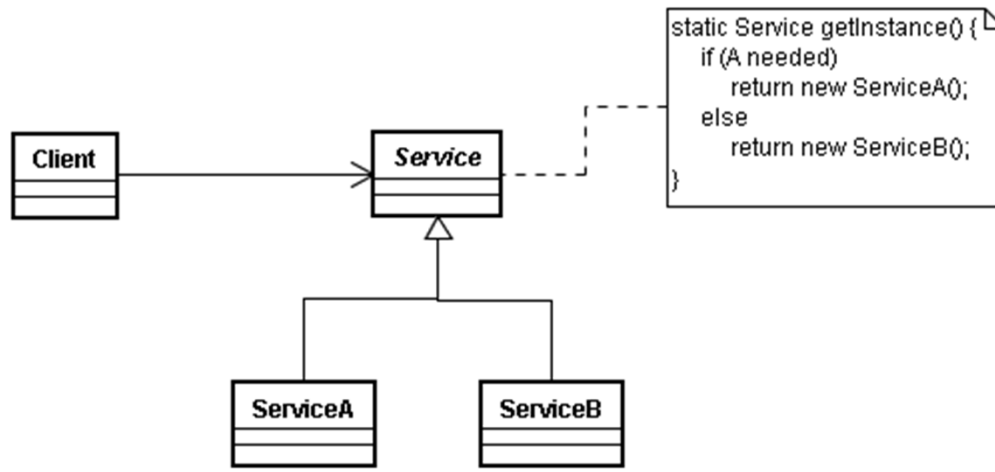
The right side is better. Service class can be made abstract tomorrow.

Service can be Abstract now!

```
static Service getInstance() {
    if (A needed)
        return new ServiceA();
    else
        return new ServiceB();
}
```

Client → Service

ServiceA   ServiceB

16-Feb-23 5:33 PM   https://vijaynathani.github.io/   5

The main advantages of using Static function for construction instead of constructor directly is:

1. Static methods have a name. This serves as documentation e.g. BigInteger(int, int, Random) is not clear but BigInteger.getProbablePrime(int, int, Random) is more clear.

2. The static method can return the same object for multiple calls e.g. if the object is immutable e.g. Boolean class has two objects TRUE and FALSE. For repeated calls, the same object can be returned.

3. Last, the static function can return a subtype object.

4. Instead of writing "Map<String,Integer>m=new Hashmap<String,Integer>();" we can write

    "Map<String,Integer>m=Hashmap.newInstance();"

Functional Programming

- Why are functional languages like Clojure, Scala and F# getting popular?
  - Languages like Java can get some functional features by proper design, using libraries like "Functional Java" and "Akka".

Functional language advantages:

functions are first class citizens

functions are side-effect free

declarative. Uses recursion to reduce the size of the problem.

Pure Functions

- A function that computes output based on parameters passed only.
  - No access to instance variables or global variables
- The function does not alter any input parameters.

Pure Function

```
public static int Add(int a, int b)
{
    return a+b;
}
```

No hidden dependencies, No side effects, Thread-safe.

?

Also known as side-effect free functions.

Q60.

Q61