DESIGN PATTERNS

STRUCTURAL PATTERNS

HOW OBJECTS COMBINE TO FORM A STRUCTURE

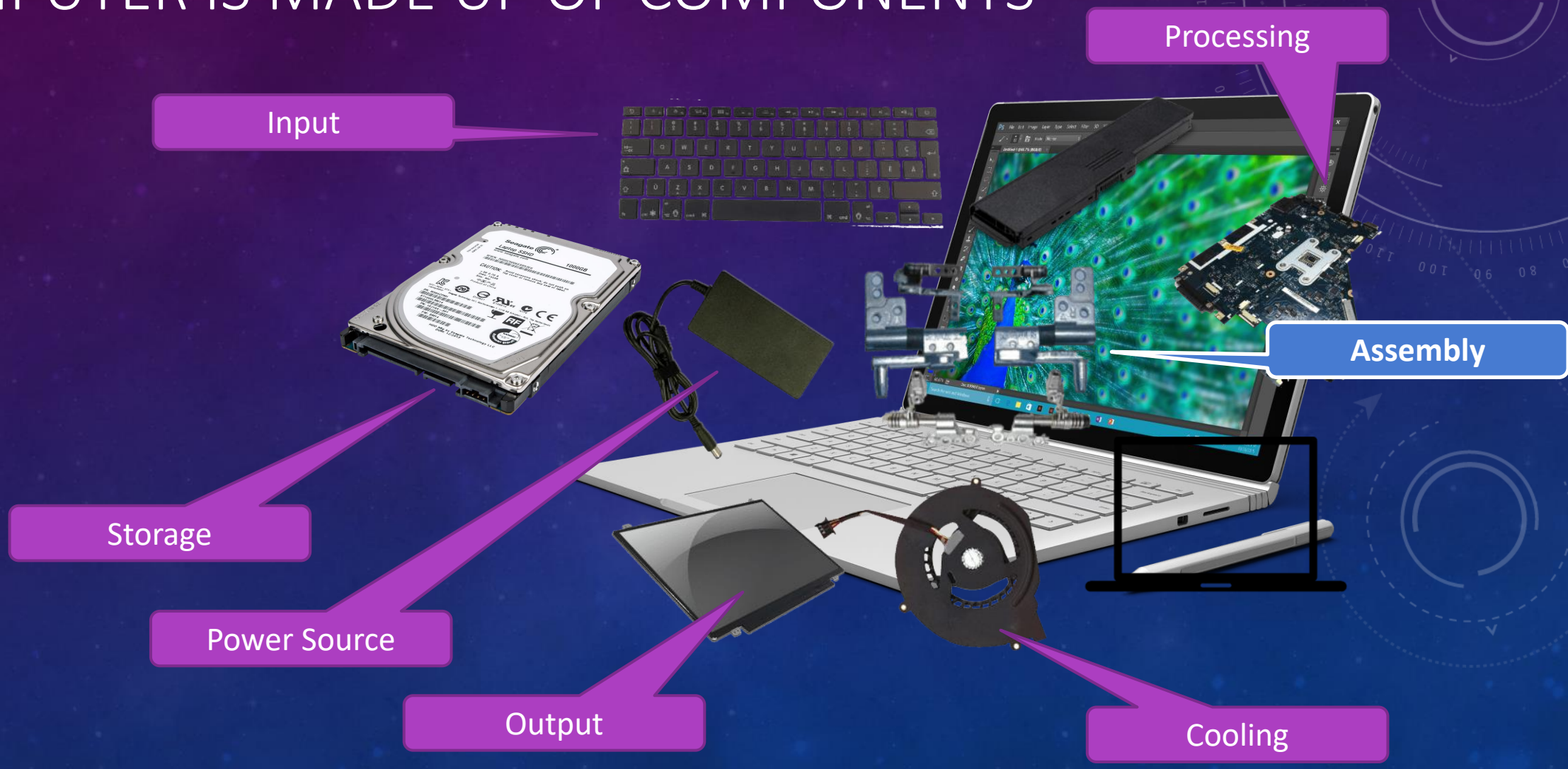
SYSTEM MODELLING

- A GOOD SYSTEM IS OFTEN A ASSEMBLY OF MANY SMALL COMPONENTS
- EACH COMPONENT PERFORMS A WELL DEFINED SINGLE RESPONSIBILITY
- EACH COMPONENT BECOMES A PART OF A LARGER SYSTEM PERFORMING A LARGER SINGLE RESPONSIBILITY

A COMPUTER SYSTEM

- Specific Responsibility - Computation
- Computation Requires
  - Input
  - Processing
  - Storage
  - Output
- Each Responsibility Carried out by sub components it is assembled of

COMPUTER IS MADE UP OF COMPONENTS



LETS DIVE DEEPER IN ONE OF THE COMPONENTS

HardDisk Responsible for storing the data

LETS DIVE DEEPER IN HARDDISK

Magentic Plates
Responsible for data storage

Magentic Head
Responsible for Creating Reading and Writing Magenetic plate

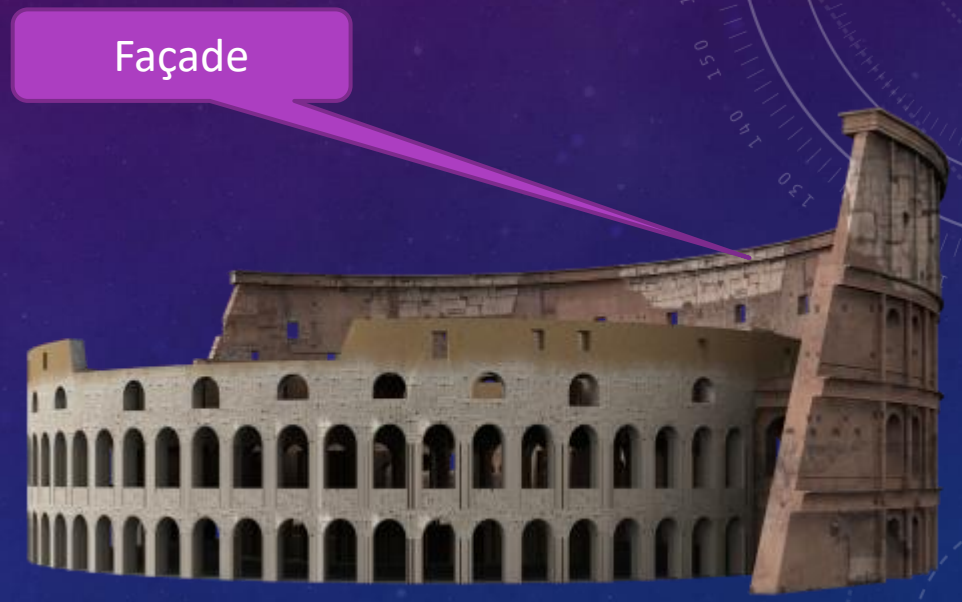
Control Circuit to direct Magentic Head

STRUCTURAL PATTERNS

- STRUCTURAL PATTERNS STUDY HOW A SYSTEM IS COMPOSED OF MULTIPLE SMALLER COMPONENTS
- GUIDELINES FOR COMBING COMPONENTS TO FORM A LARGER SYSTEM
- ONE OR MORE COMPONENT WORK TOGETHER
- STUDY OF LAW #1 OF OO DESIGN
- ENCAPSULATE TO REUSE

FAÇADE

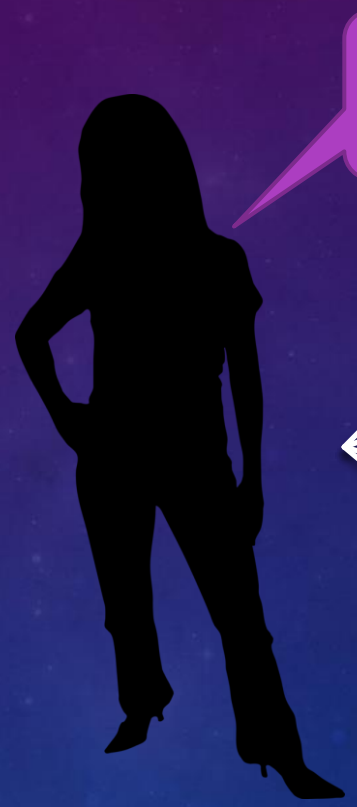
- MOTIVATED FROM BUILDING ARCHITECTURE
- A FALSE EXTERIOR TO CONCEAL SOMETHING UGLY



- IN SOFTWARES Façade MEANS AN OPTIONAL INTERFACE TO CONCEAL SOMETHING COMPLEX
- A SIMPLIFIED INTERFACE TO HIDE COMPLEX INTERACTION/DEPENDENCY

REAL WORLD USE CASE - TELESHOPPING

- Customer is connected to different department
- Multiple dependency
- You don't want to be in this situation
- What's the solution?



Customer

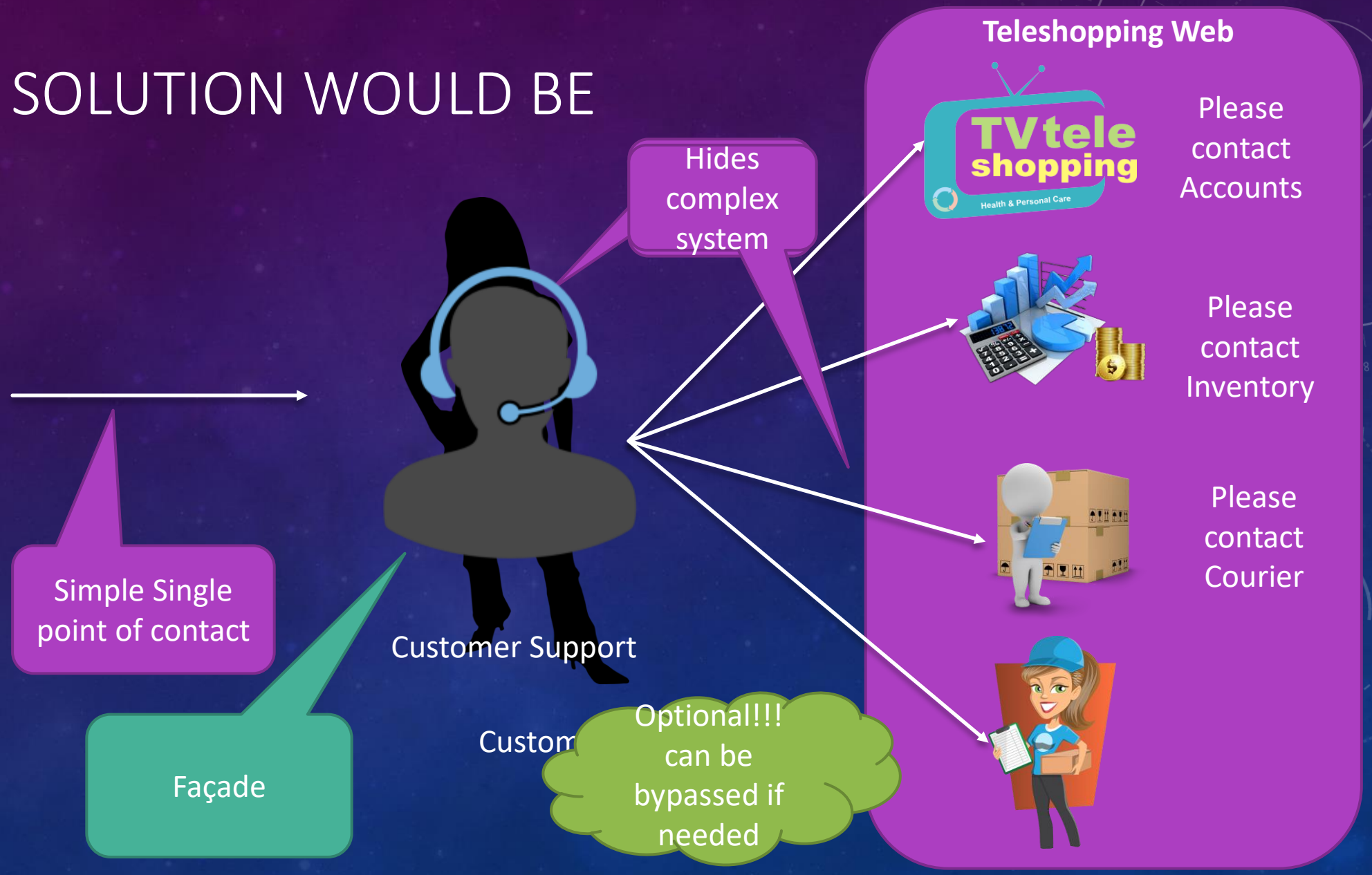
Teleshopping Web

Please contact Accounts

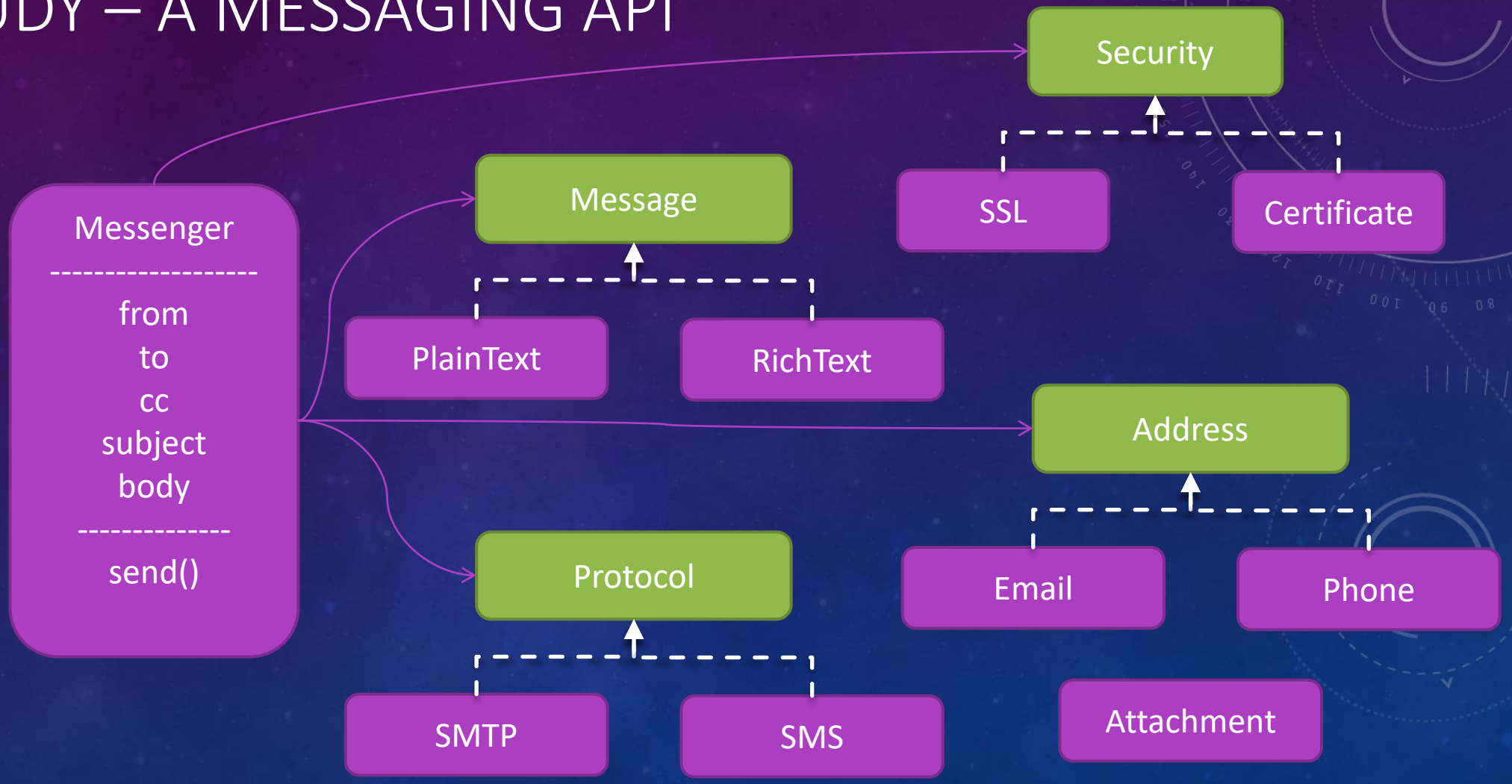
Please contact Inventory

Please contact Courier

A BETTER SOLUTION WOULD BE



CASE STUDY – A MESSAGING API

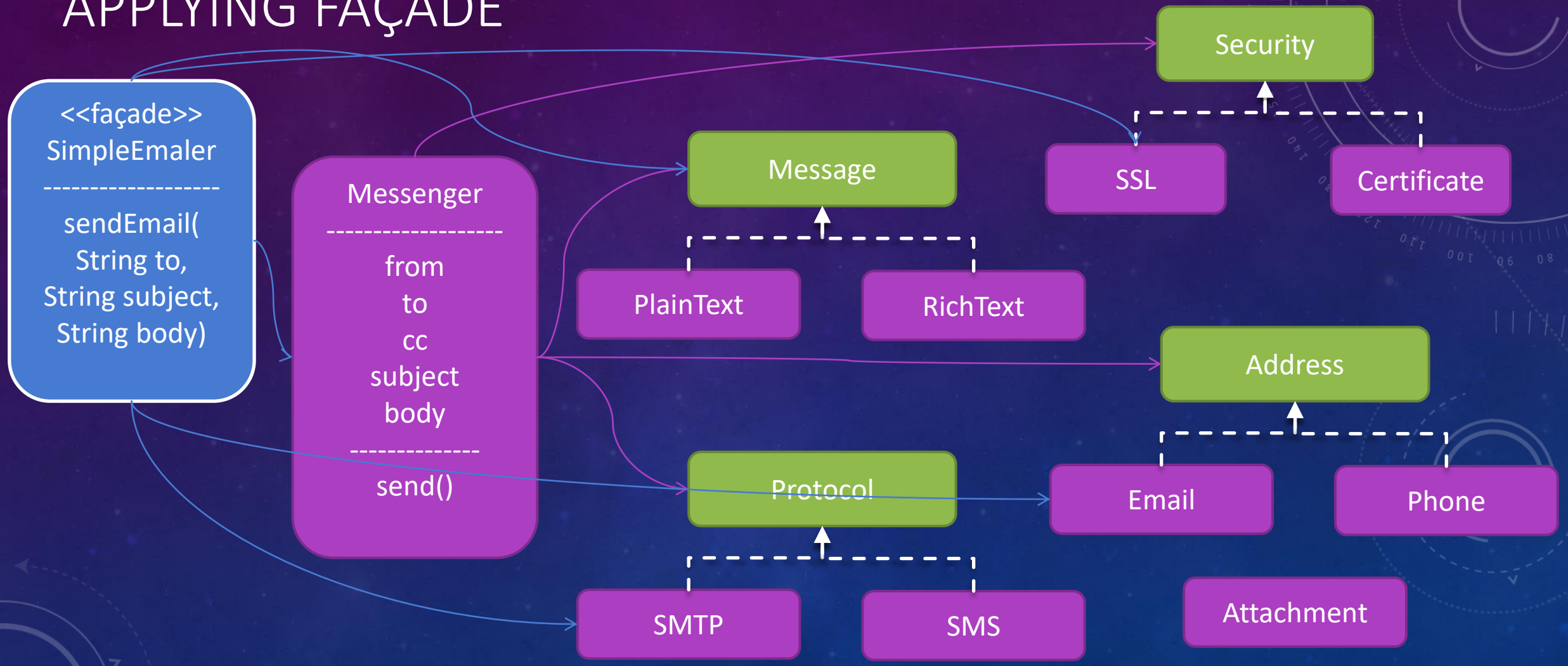


IF WE NEED TO SEND A ACCOUNT ACTIVATION MAIL

```
void sendActivationEmail(){

    Messenger messenger=new Messenger();
    messenger.setFrom( new Email("admin@web.com"));
    messenger.addTo( new Email(newUser.getEmail()));
    messenger.setProtocol( SMTP.instance);
    messenger.setSubject( "Activation Text");
    messenger.setBody( new HtmlBody(activationMail));
    messenger.setSecurity( new UserNamePassword("admin", "p@ss");
    messenger.send();
}
```

All this just to send a simple Email which is a very common requirement!!!

APPLYING FAÇADE



SIMPLE MAILER FACADE

```
class SimpleMailer{
    void send (String to, String subject, String body){

        Messenger messenger=new Messenger();
        messenger.setFrom( new Email(config.from));
        messenger.addTo( new Email(to));
        messenger.setProtocol( SMTP.instance);
        messenger.setSubject( subject);
        messenger.setBody( new HtmlBody(body));
        messenger.setSecurity( new UserNamePassword(
                                    config.username,
                                    config.password));

        messenger.send();
    }
}
```

Reusable Façade

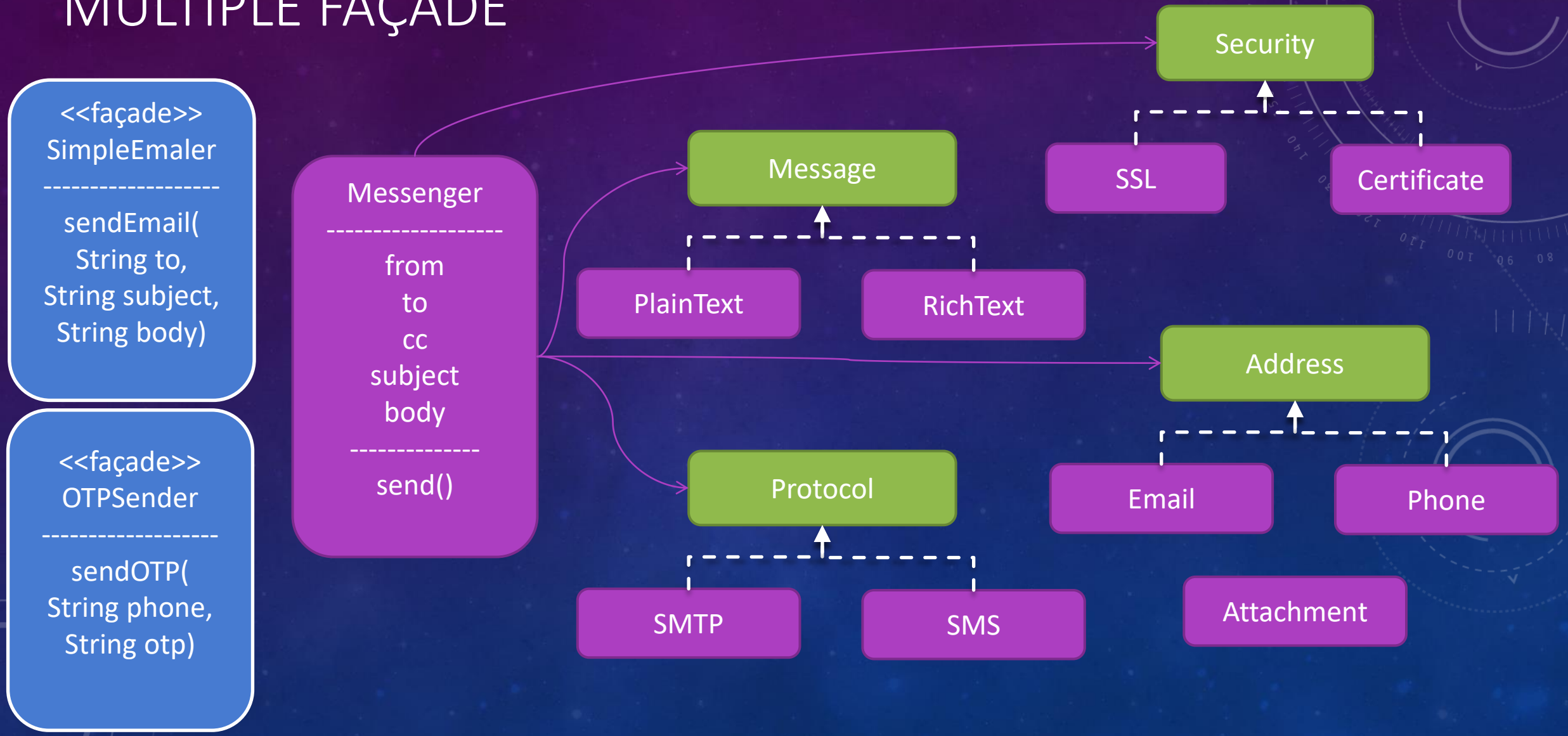
config parameter

user parameter

USING FACADE

```
void sendActivationMail(){

    SimpleEmailer
        .getInstance()
        .send( newUser.getEmail(), “activation link”, activationText);


}
```

MULTIPLE FAÇADE

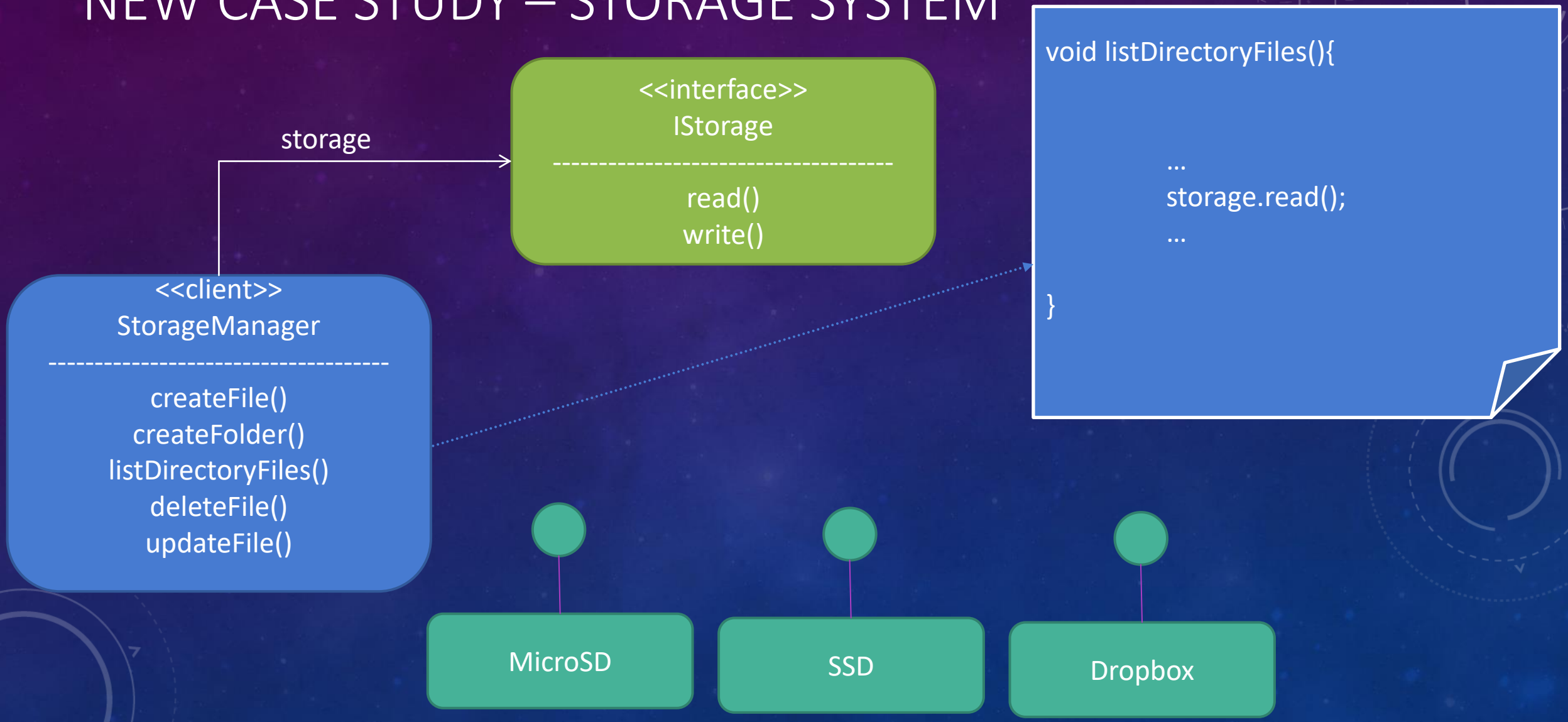


CAUTION

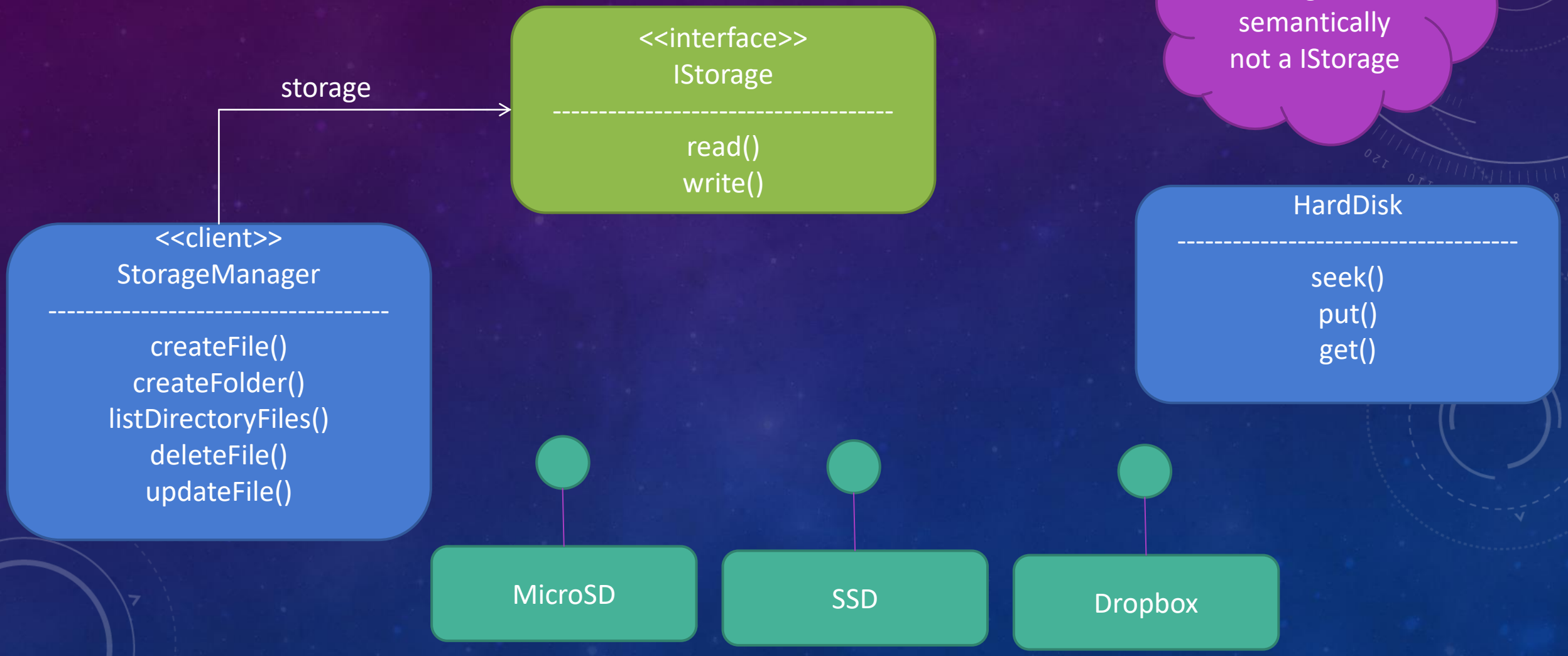
- Façade WAS INTENDED TO BE A SIMPLIFIED INTERFACE TO HIDE SOMETHING COMPLEX
- OF LATE Façade HAS BECOME AN EXCUSE FOR WRITING UGLY CODE
- ANY ARBIRATRY FUNCTION IS GENERALLY PASSED AS façade
- IT HAS BECOME AN ANTI-PATTERN
- Façade GENERALLY DOESN'T IMPLEMENT AN INTERFACE
  - BECOMES IRRISPONSIBLE DESIGN
- Façade SHOULD BE USE TO HIDE COMPLEX DEPENDNECY AND INTERACTION
- AVOID CREATING TOO MANY FACADE

AS A BEGINNER ITS BEST TO AVOID Façade

NEW CASE STUDY – STORAGE SYSTEM



INTRODUCING OLD STORAGE - HARDDISK

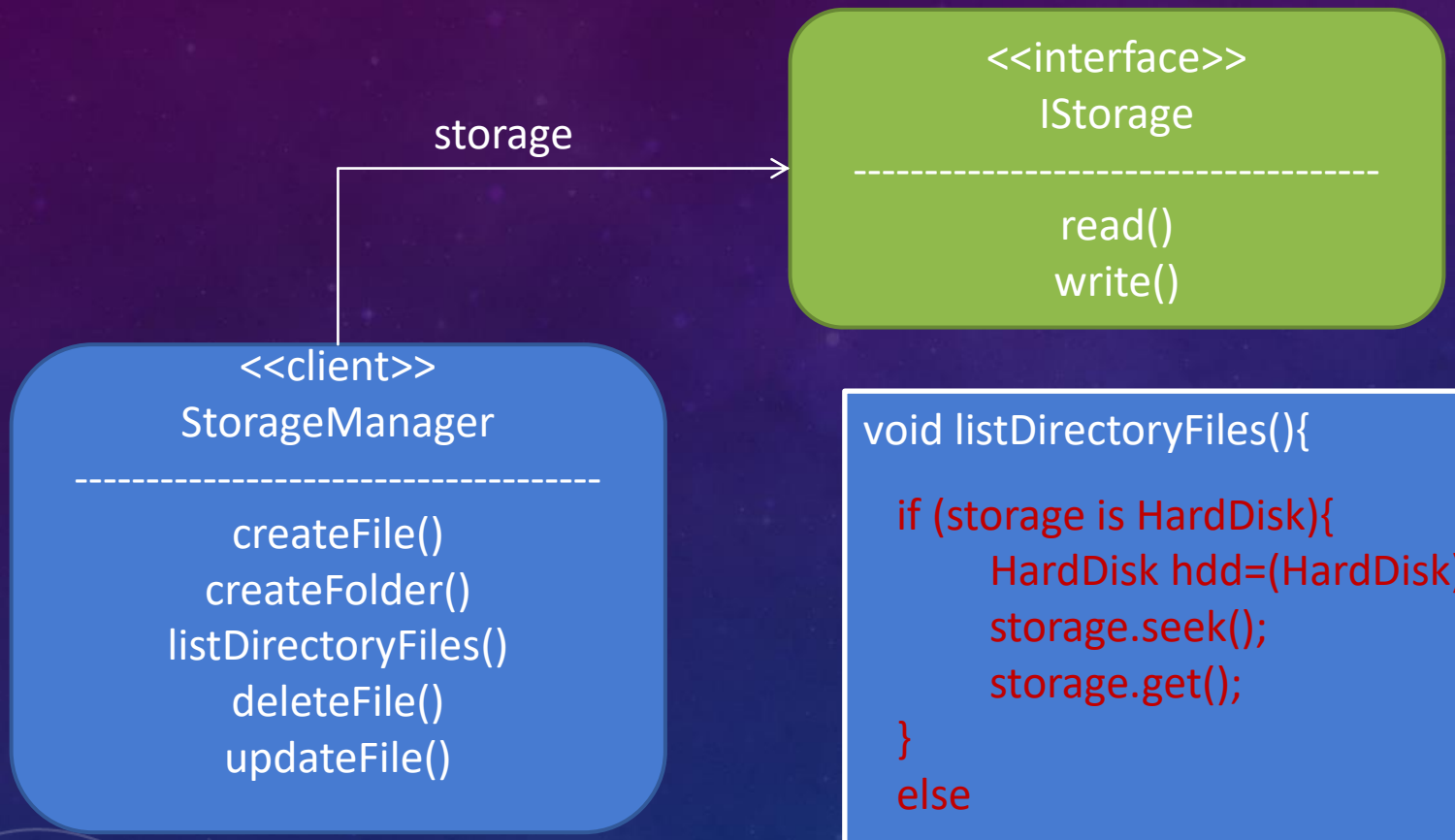


IS HARDDISK A STORAGE?

- FUNCTIONALLY HARDDISK IS A STORAGE
- BUT THE INTERFACE IS DIFFERENT
- STORAGE MANAGER CANNOT DIRECTLY USE HARDDISK
- WHAT ARE THE OPTIONS???

APPROACH A. MODIFY STORAGE MANAGER

- violates OCP/LSP
- What if tomorrow I need Magentic Tape



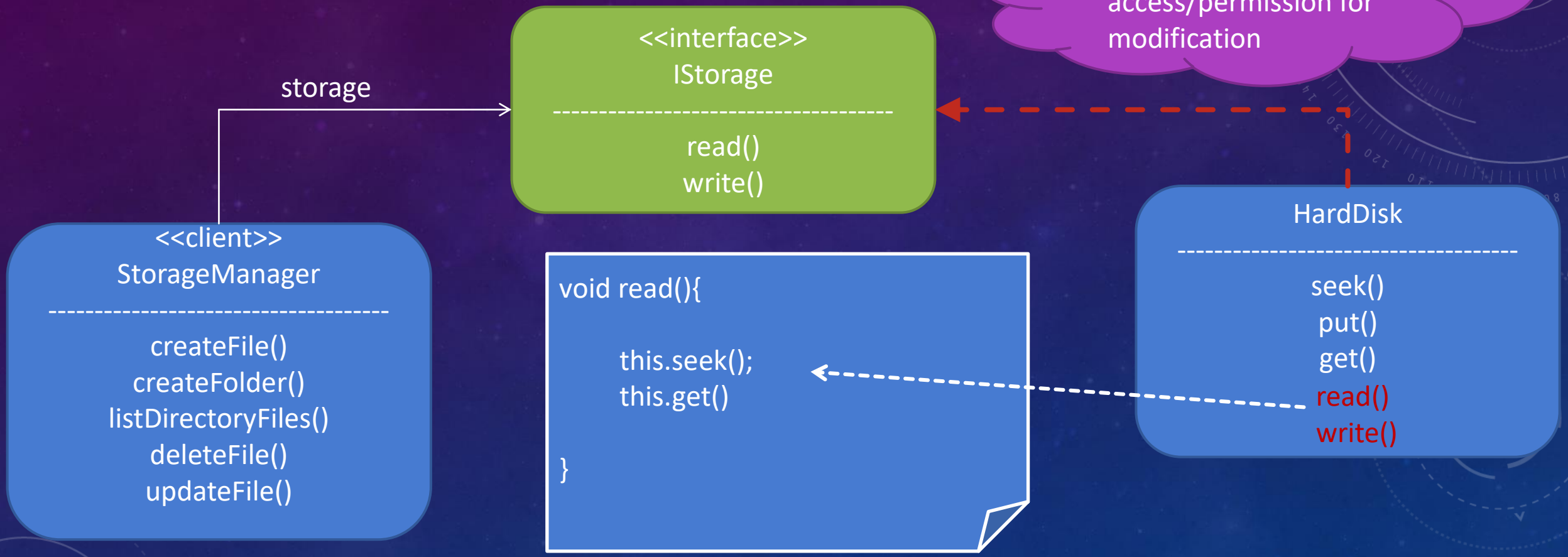

```
void listDirectoryFiles(){

  if (storage is HardDisk){
        HardDisk hdd=(HardDisk)storage;
        storage.seek();
        storage.get();
  }
  else

      storage.read();
}
```

HardDisk

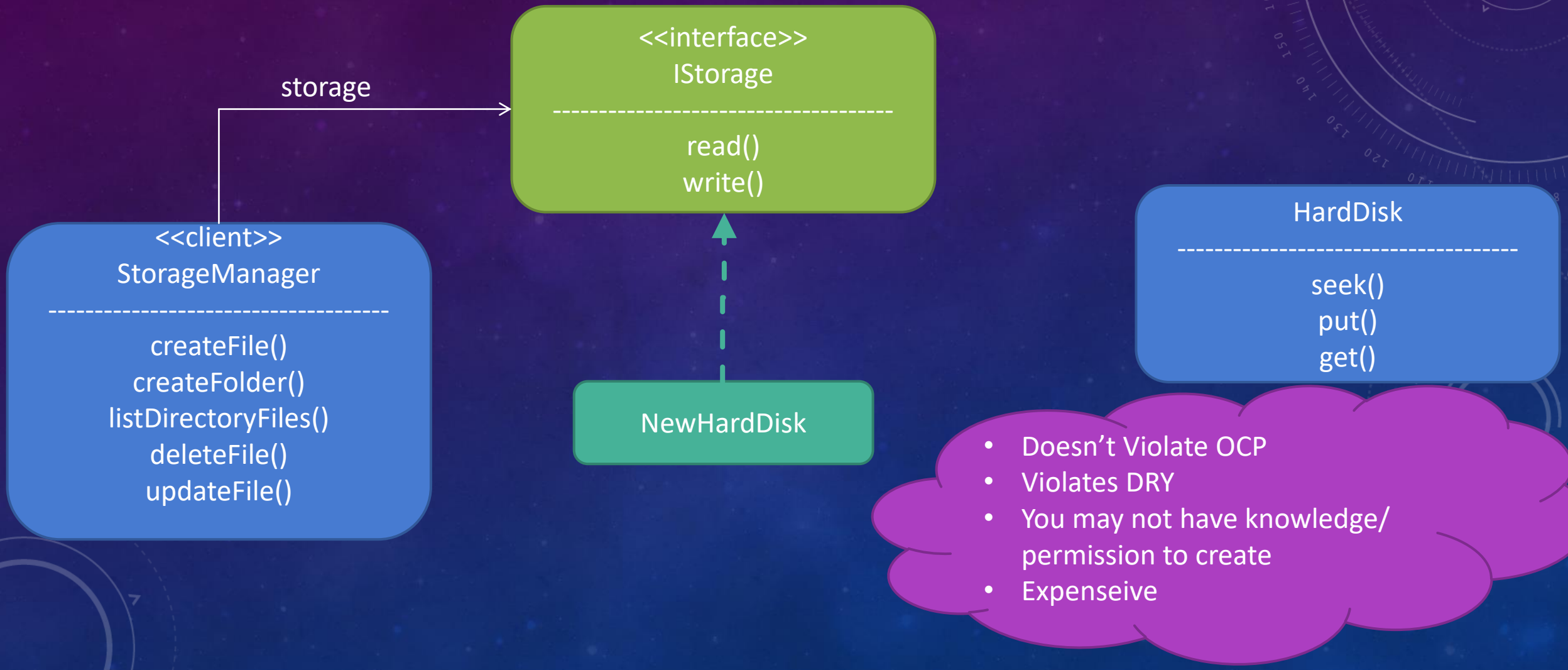
seek()
put()
get()

APPROACH B. MODIFY HARDDISK

- violates OCP/LSP
- You may not have access/permission for modification



APPROACH C. CREATE NEW HARDDISK



PROBLEMS WITH THE EXISTING APPROACH

- Approach A – MODIFY CLIENT
  - VIOLATES OCP
  - BREAKS OCP
    - TOMORROW THERE MAY BE A MAGNETIC DISC
- APPROACH B – MODIFY THE COMPONENT
  - VIOLATES OCP
  - YOU MAY HAVE PERMISSION/ACCESS TO MODIFY THE COMPONENT
- APPROACH C – CREATE NEW HARDDISK
  - NO VIOLATEION OF OCP
  - VIOLATES DRY
  - YOU MAY NOT HAVE KNOWLEDGE OR PERMISSION TO REPRODUCE LOGIC
  - EXPENSIVE

INTRODUCING ADAPTER DESIGN PATTERN

- ADAPTER IS WRAPPER DESIGN PATTERN
  - IT WRAPS A TARGET
- IT DOESN’T ADD NEW FUNCTIONALITY
  - IT REUSES FUNCTIONALITY FROM THE TARGET
- ADAPTERS **JOB** IS TO TRANSLATE THE INTERFACE
- ADAPTER IS USED TO MAKE TO INCOMPATIBLE SYSTEM WORK WITH EACH OTHER
- LIGHTWEIGHT AND INEXPENSIVE

HARDWARE ADAPTER

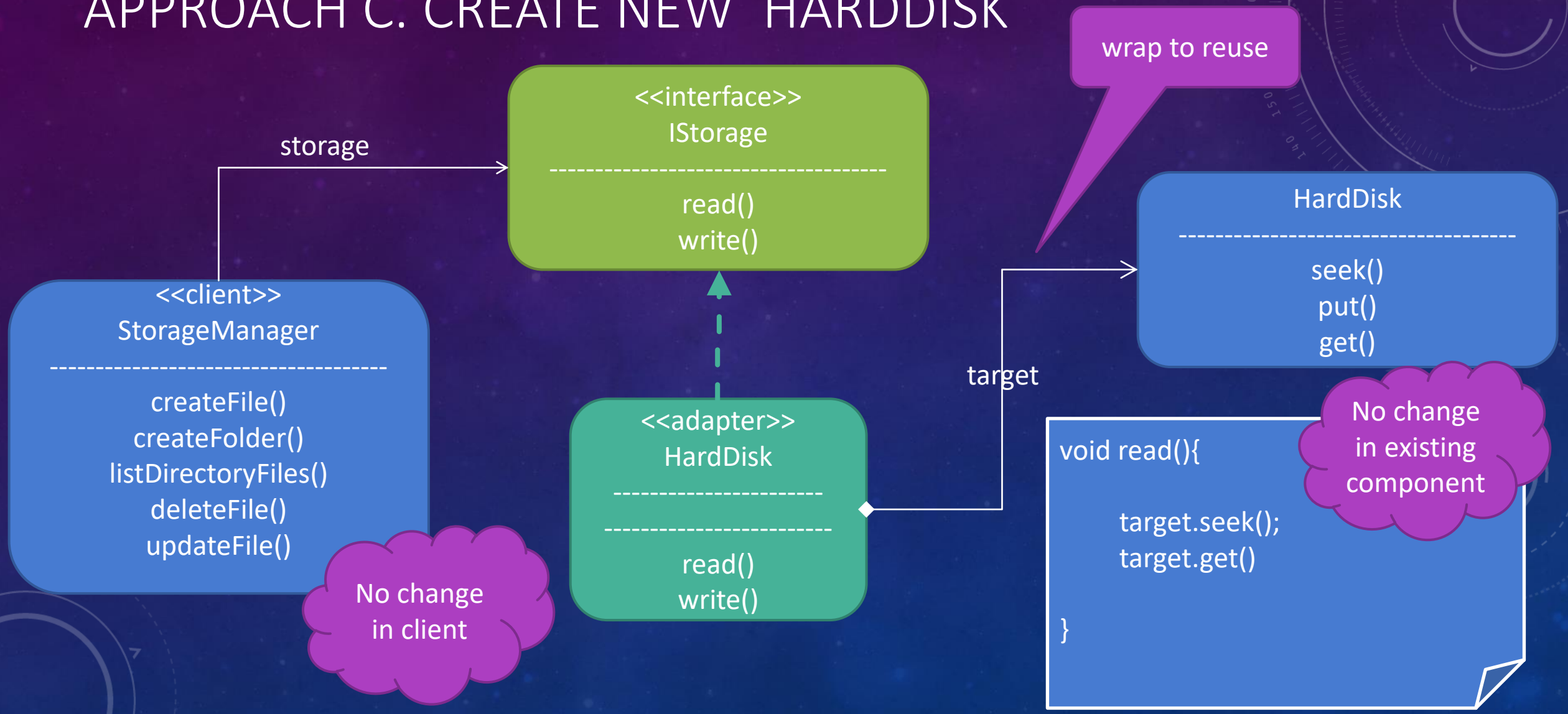
Adapter allows incompatible system work with each other

A Mobile and Mouse are not originally desiged to work with Each other. Mobile connects to USB port, Mobile have USB Type C Support

ADAPTER PATTERN BENEFITS

- LATE DESIGN SOLUTION
- CAN ADD NEW INTERFACE TO AN EXISTING OBJECT AT RUNTIME
- MOBILE PHONE DOESN’T HAVE USB INTERFACE
- MOBILE PHONE CAN SUPPORT USB INTERFACE WHEN COMBINED WITH OTG ADAPTER
- EXCELLENT FOR REFACTORING OLD CODE

APPROACH C. CREATE NEW HARDDISK



ADAPTER ADDS DYNAMIC INTERFACE TO AN OBJECT

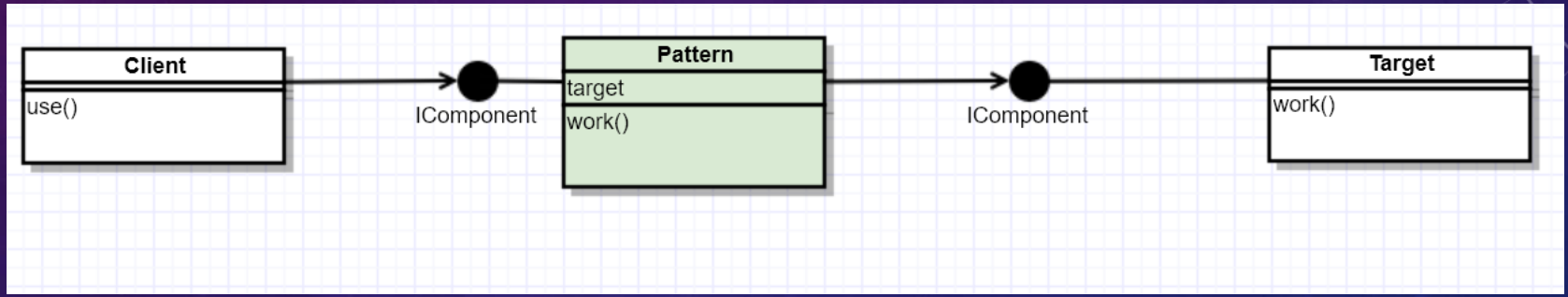
```
HardDisk hdd1=new HardDisk(512);

HardDisk hdd2= new HardDisk(1024);

IStorage adapter=new HardDiskAdapter(hdd2);

IStorage storage=hdd1;

IStorage storage=adapter;
```

Effectively hdd1 is not a IStorage but hdd2 (with adapter) is a IStorage

Fails. HardDisk doesn't implements IStorage

Works. adapter is essentially the wrapped hdd2

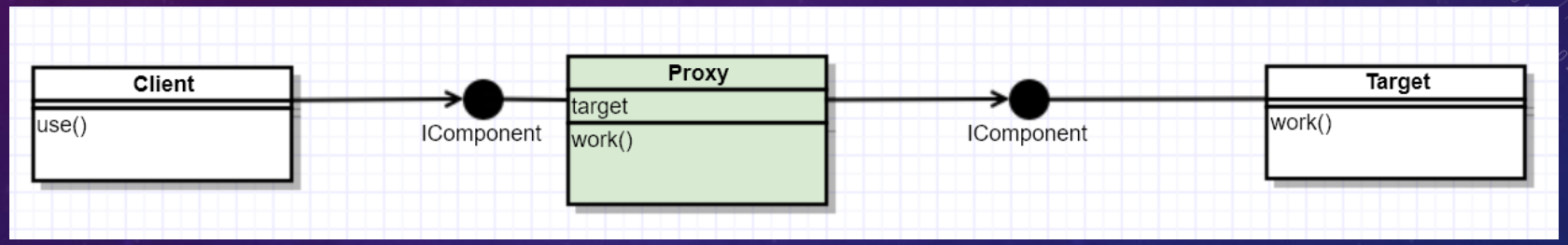
WHAT DOES THIS DIAGRAM REPRESENT?



- IT IS A WRAPPER DESIGN PATTERN
- CLIENT → PATTERN → TARGET
- PATTERN EXPOSES TARGETS FUNCTIONALITY
- NO NEW BEHAVIOR ADDED

- NO INTERFACE TRANSLATION
- TRANSPARENT TO CLIENT
- CLIENT MAY NOT BE AWARE OF THE EXISTANCE OF THIS PATTERN
- WHAT IS IT AND WHY DOES IT EXIST?

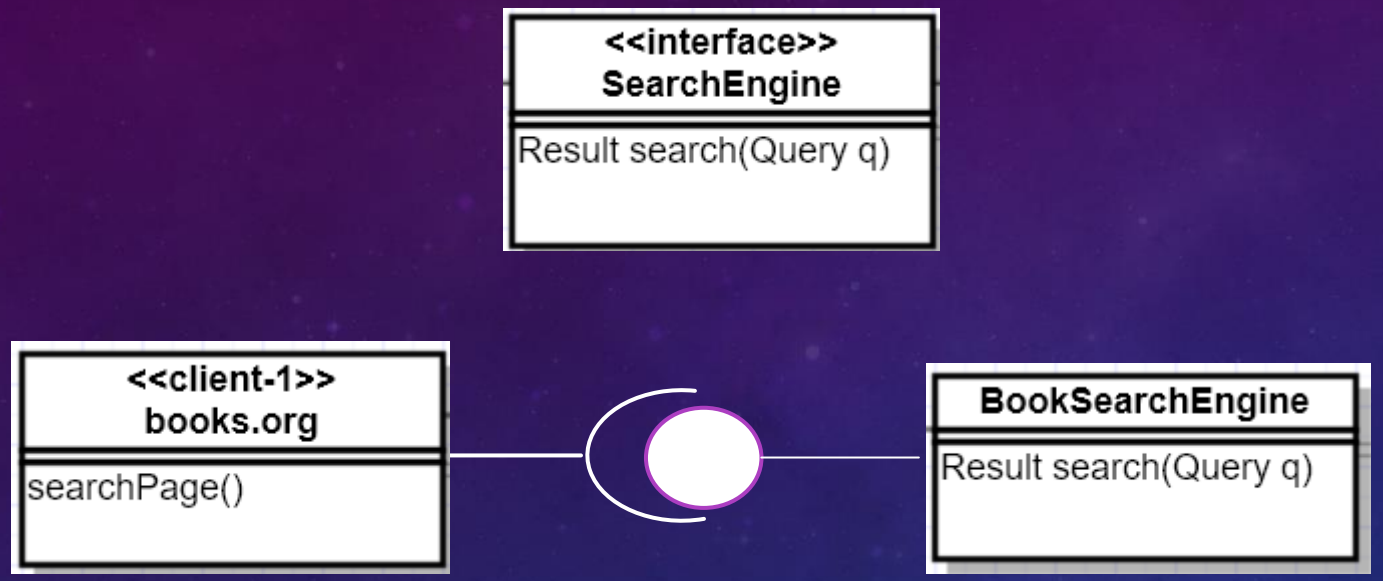
PROXY



IT DOESN'T ADD NEW BEHAVIOR; IT CONTROLS THE EXISTING BEHAVIORS

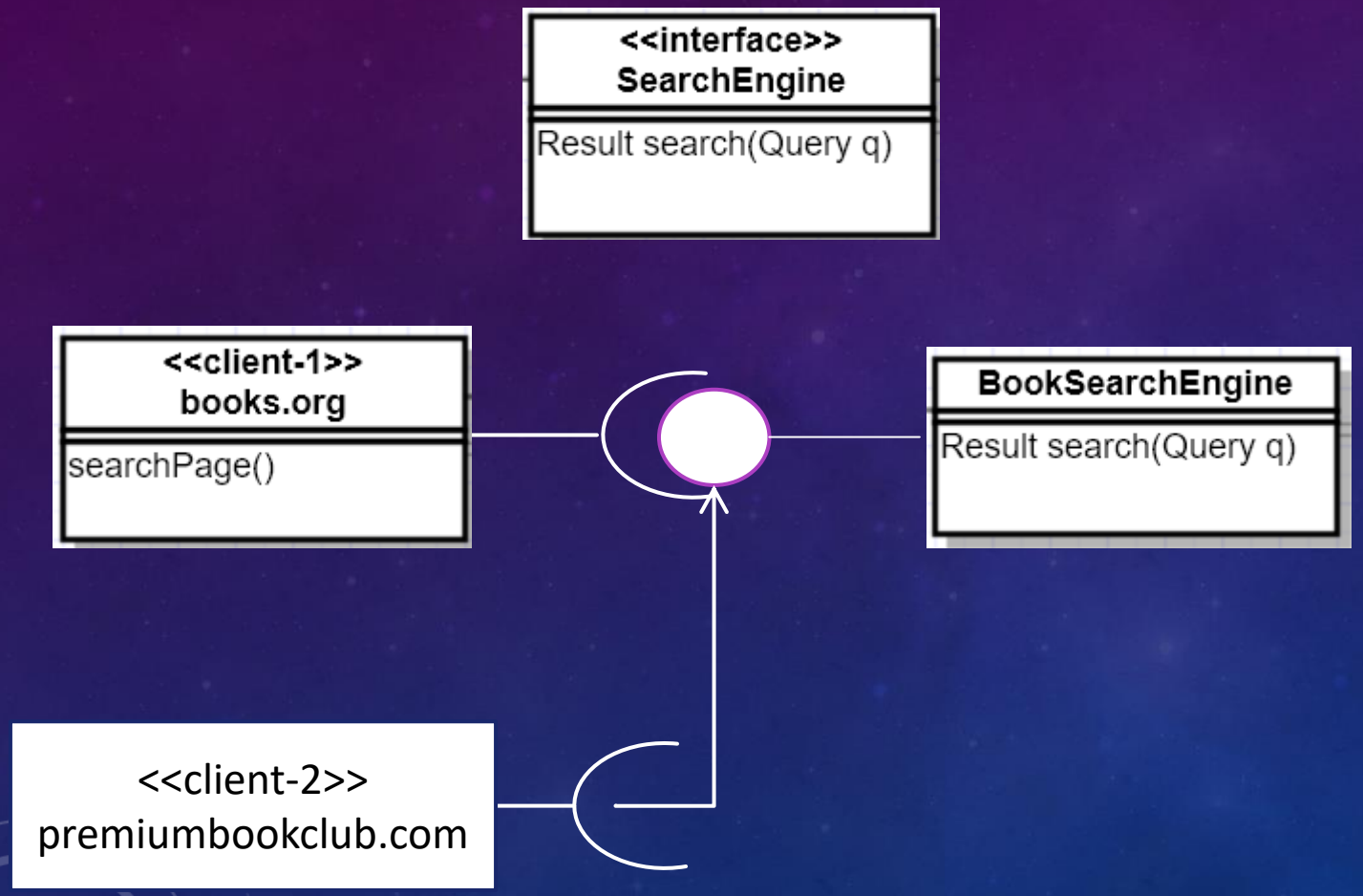
BOOK SEARCH MODEL

SEARCH ENGINE USE CASE




```
public class BookSearchEngine implements SearchEngine{


 public void Result search(Query q){




  BookQuery bq=(BookQuery) q;
  BookSearchResult result=new BookSearchResult();
        //do the search

                return result;


        }

}
```

NEXT CLIENT



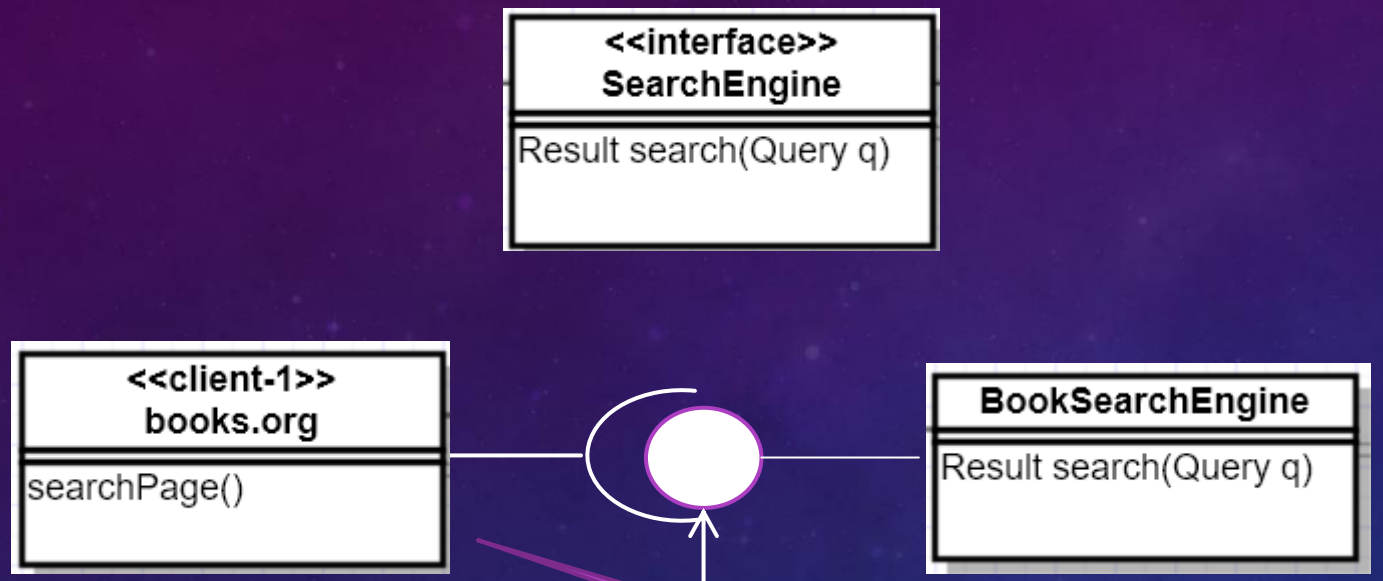
Time for New Client.

But New Client Need a Missing Feature - Authentication

How do I add this feature???

Good News!!!
Its Just One line Change

NEXT CLIENT



<<client-2>>
premiumbookclub.com

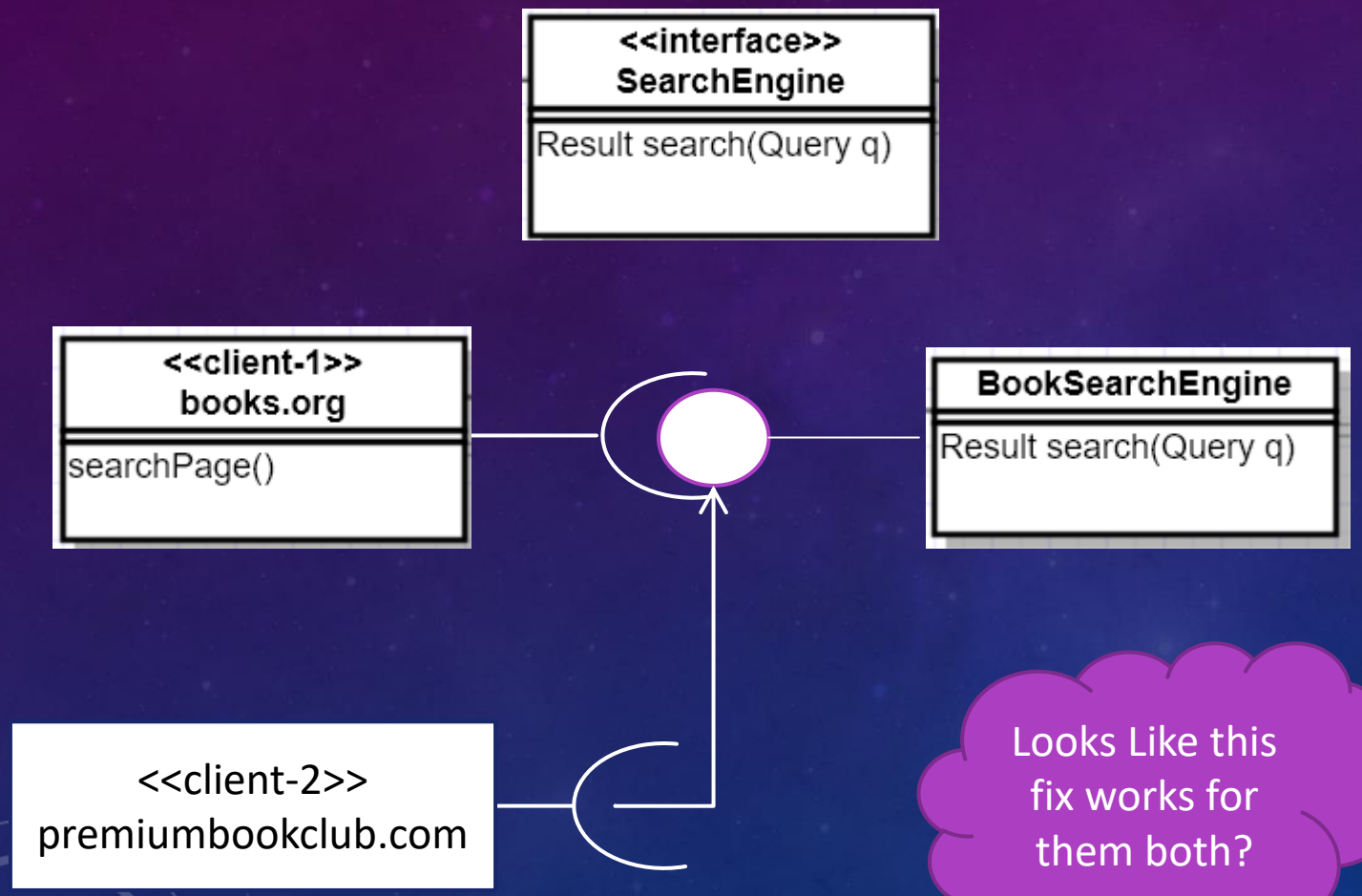
```
public class BookSearchEngine implements SearchEngine{


 public void Result search(Query q){

   if ( ! HttpContext.IsAuthenticated())
         throw new SearchFailedException();

 BookQuery bq=(BookQuery) q;
 BookSearchResult result=new BookSearchResult();
        //do the search

                return result;


        }

}
```

Wow Its Perfect Now!!!

Hell !!! Its Now broken.

A Change may not be acceptable to all

LETS FIX IT FOR THEM BOTH

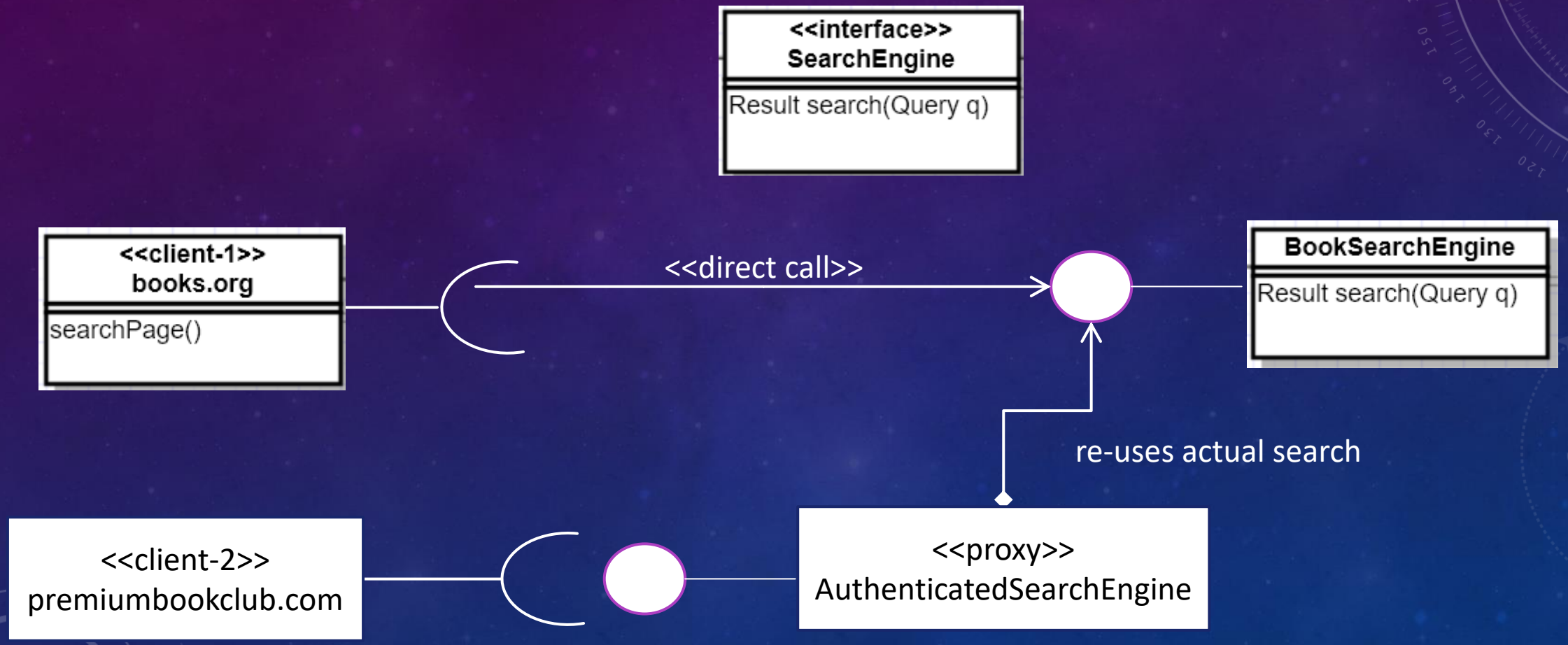



```
public class BookSearchEngine implements SearchEngine{

 boolean authenticationRequired;

 public void Result search(Query q){
 if( authenticationRequired)
    if ( ! HttpContext.IsAuthenticated())
          throw new SearchFailedException();

  BookQuery bq=(BookQuery) q;
  BookSearchResult result=new BookSearchResult();
        //do the search

                return result;


        }

}
```

Looks Like this fix works for them both?

BUT THERE ARE MORE CLIENTS AND MORE NEEDS?

LETS FIX IT FOR THEM BOTH



AUTHENTICATION PROXY

```
public class AuthenticatedSearchEngine implements SearchEngine{

 ISearchEngine target; //DI

 public Result search(Query q){

   if ( ! HttpContext.IsAuthenticated())
         throw new SearchFailedException();

    else
          return target.search(q);



        }

}
```

reuse

```
public class BookSearchEngine implements SearchEngine{


 public Result search(Query q){

   if ( ! HttpContext.IsAuthenticated())
         throw new SearchFailedException();

 BookQuery bq=(BookQuery) q;
 BookSearchResult result=new BookSearchResult();
       //do the search

              return result;


        }

}
```

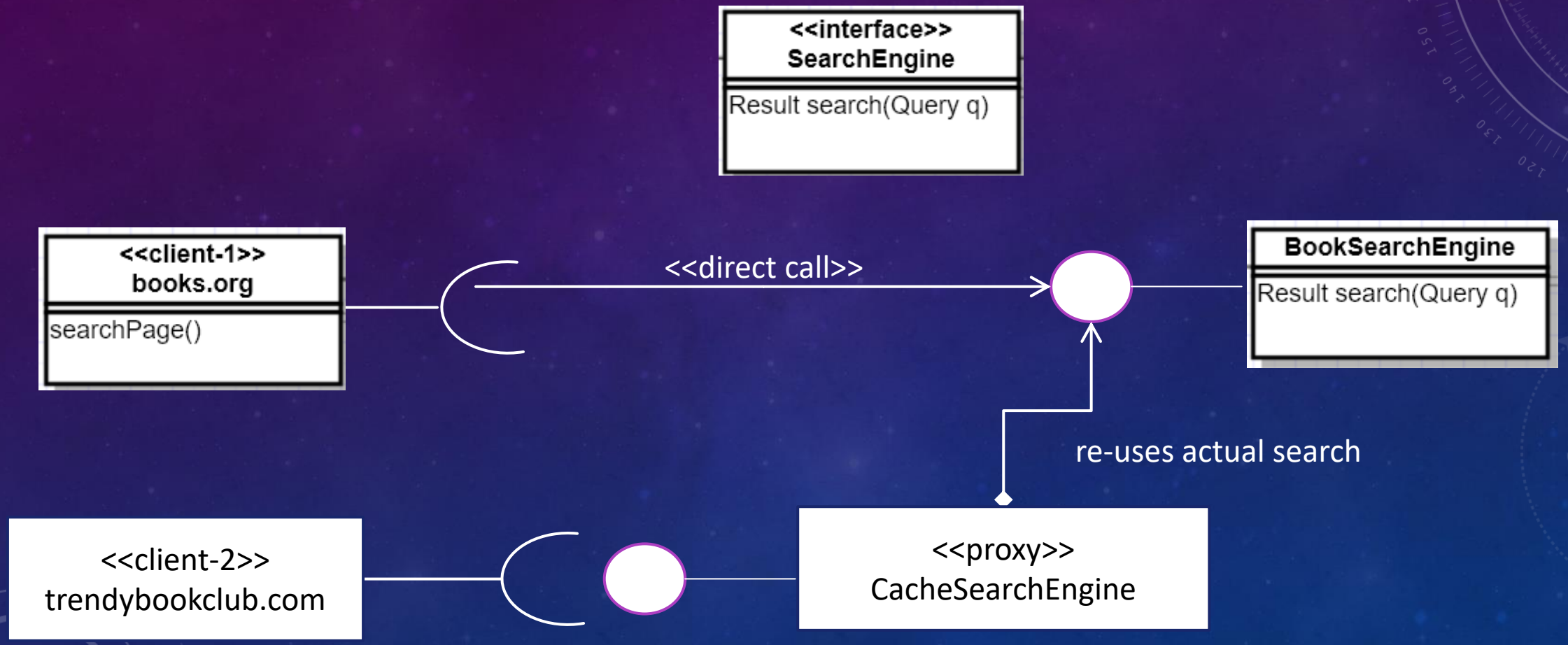
SO WHAT DOES PROXY DO?

1. WRAPS TARGET
2. CLIENT CALLS PROXY
    1. CLIENT MAY NOT KNOW ITS NOT CALLING THE REAL TARGET
3. PROXY MAY PERFORM SOME TASK
    1. IT MAY RETURN WITHOUT CALLING ACTUAL TARGET OR
    2. IT MAY CALL ACTUAL TARGET
4. TARGET RETURNS CONTROL BACK TO PROXY
    1. PROXY MAY DO ADDITIONAL WORK
5. RESULT IS RETURNED TO THE CLIENT.

NEW CLIENT – NEW NEED

- CLIENT: TRENDYBOOKCLUB.COM
- PROBLEM STATEMENT
  - WE HAVE HUGE DATABASE OF BOOKS BUT 90% QUERY RELATED TO TRENDIGN 2% BOOK. SAME QUERY HITS THE DATABASE AGAIN AND AGAIN. THIS LOOKS LIKE A PERFORMANCE OVERHEAD.
- WHATS THE SOLUTION?

SIMILAR SOLUTION - CACHING PROXY



AUTHENTICATION PROXY

```
public class CachedSearchEngine implements SearchEngine{
 ISearchEngine target; //DI
 Dictionary<Query,Result> cache;
 public Result search(Query q){

   if ( ! cache.ContainsKey(q))
        return cache[q]; //returns cached result
    else{

          var r= target.search(q);
          cache[q]=r;
          return r;
    }

      }

}
```

reuse

```
public class BookSearchEngine implements SearchEngine{


 public Result search(Query q){

 BookQuery bq=(BookQuery) q;
  BookSearchResult result=new BookSearchResult();
        //do the search

                return result;


        }

}
```

BUT THERE ARE MORE CLIENTS AND MORE NEEDS?

I NEED TO KNOW WHAT CLIENT IS MOSTLY SEARCHING

I NEED TO BLACKLIST CERTAIN IP

DIFFERENT NEEDS SAME SOLUTION

I NEED TO MEASURE THE PERFORMANCE

I NEED TO RECORD FAILED SEARCHES

MULTIPLE NEEDS

I NEED TO KNOW WHAT CLIENT IS MOSTLY SEARCHING

I NEED TO BLACKLIST CERTAIN IP

SAME CLIENT MULTIPLE NEEDS

I NEED TO MEASURE THE PERFORMANCE

I NEED TO RECORD FAILED SEARCHES

USE-CASE 1 : PLAIN SEARCH


```
ISearchEngine engine= new BookSearchEngine();
```

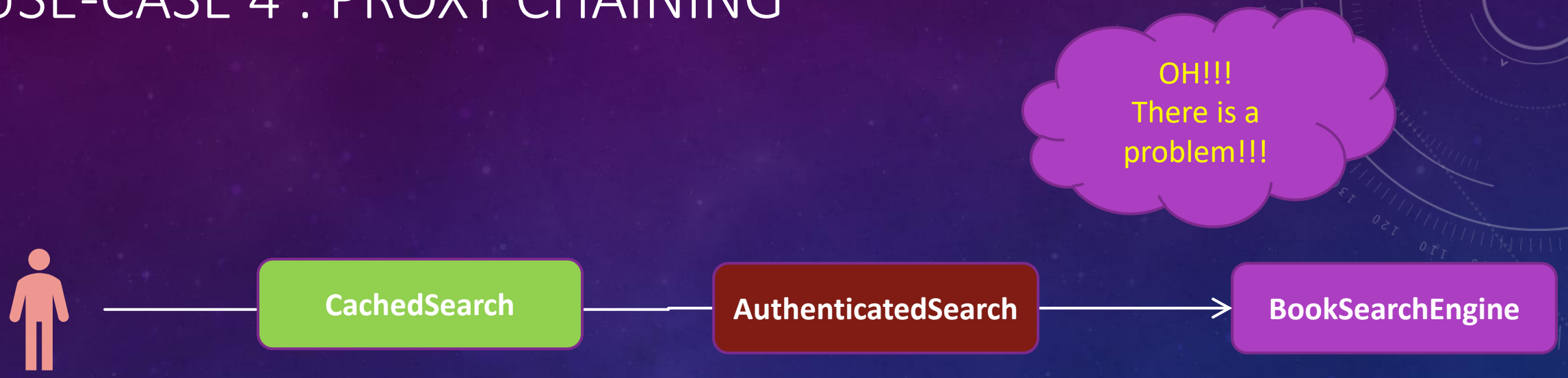
USE-CASE 2 : AUTHENTICATED SEARCH


```
ISearchEngine engine= new AuthenticatedSearch
                                (
                                        new BookSearchEngine()
                                );
```

USE-CASE 3 : CACHED SEARCH


```
ISearchEngine engine= new CachedSearch
                                (
                                        new BookSearchEngine()
                                );
```

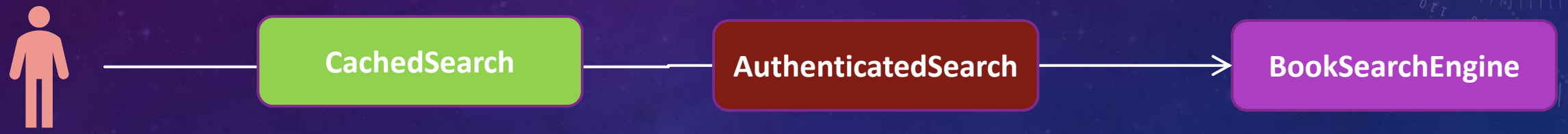
USE-CASE 4 : PROXY CHAINING




```
ISearchEngine engine= new CachedSearch
                        (
                            new AuthenticatedSearch(
                                new BookSearchEngine()
                            );
                        );
```

USE-CASE 4 : PROXY CHAINING

Can be corrected by re-ordering object creation.




```
ISearchEngine engine= new AuthenticatedSearch(

                    new CachedSearch(
                        new BookSearchEngine()
                    )
                );
```

PROXY CONFIGURATION

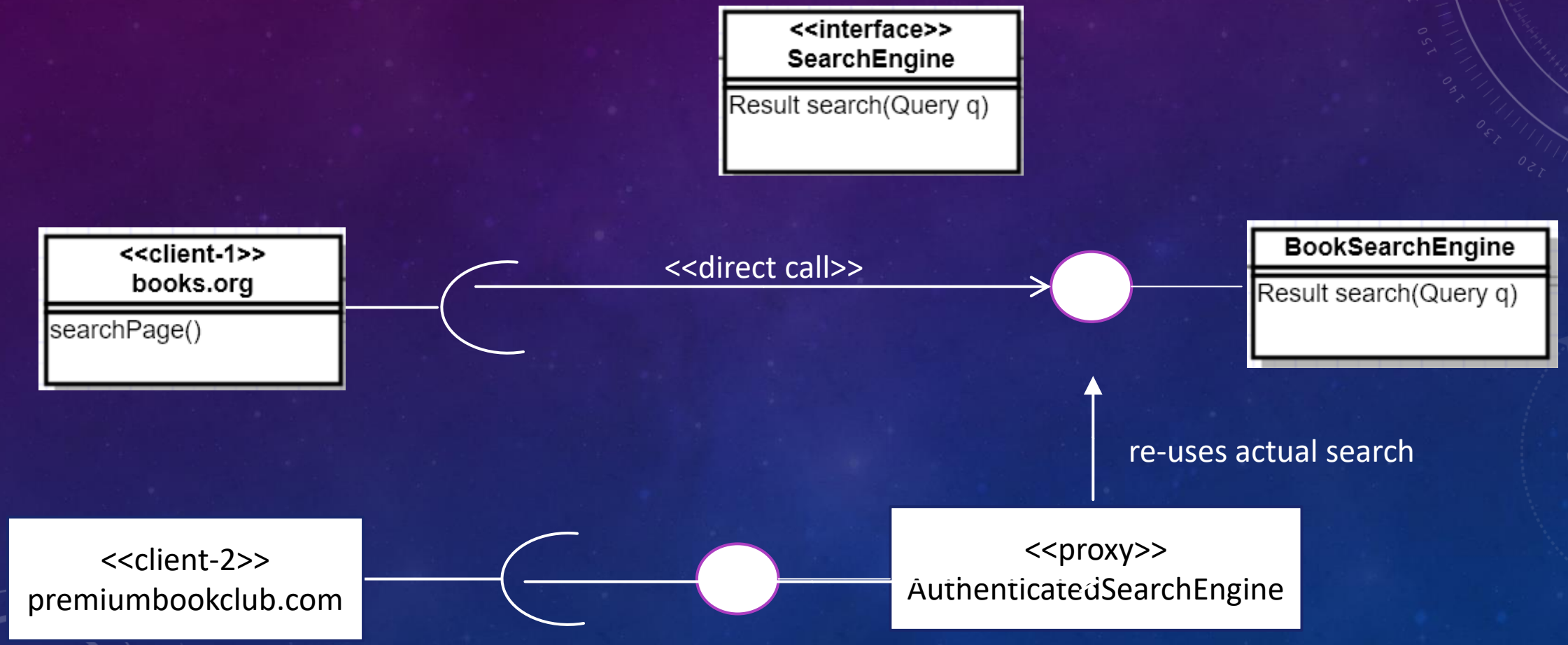
```
ISearchEngine engine= SearchEngineBuilder.Configure("proxy-config.json");
```

```
{
    "target" : "BookSearchEgine",
    "proxies" : [ "AuthenticationSearch" , "CachedSearch" ]
}
```

INHERITANCE & PROXY

- PROXY ENCAPSULATES COMPONENT TO REUSE
- POPULAR PERCEPTION IS – INHERITANCE HELPS IN REUSE
- CAN WE IMPLMENT INHERITANCE TO IMPLEMENT PROXY?

INHERITANCE PROXY



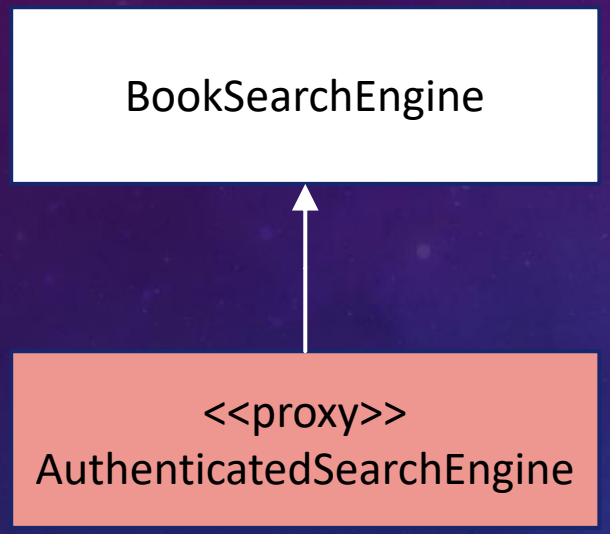
INHERITANCE PROXY CODE

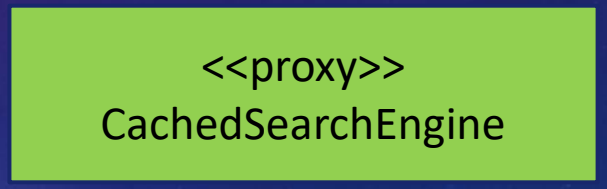
```
public class AuthenticatedSearchEngine  extends BookSearchEngine
{
 ISearchEngine target; //DI


 public Result search(Query q){
    if ( ! HttpContext.IsAuthenticated())
          throw new SearchFailedException();
     else
          return target.search(q);
          return base.search(q);



        }

}
```

reuse

```
public class BookSearchEngine implements SearchEngine{


 public Result search(Query q){
  BookQuery bq=(BookQuery) q;
  BookSearchResult result=new BookSearchResult();
         //do the search

                return result;


        }

}
```

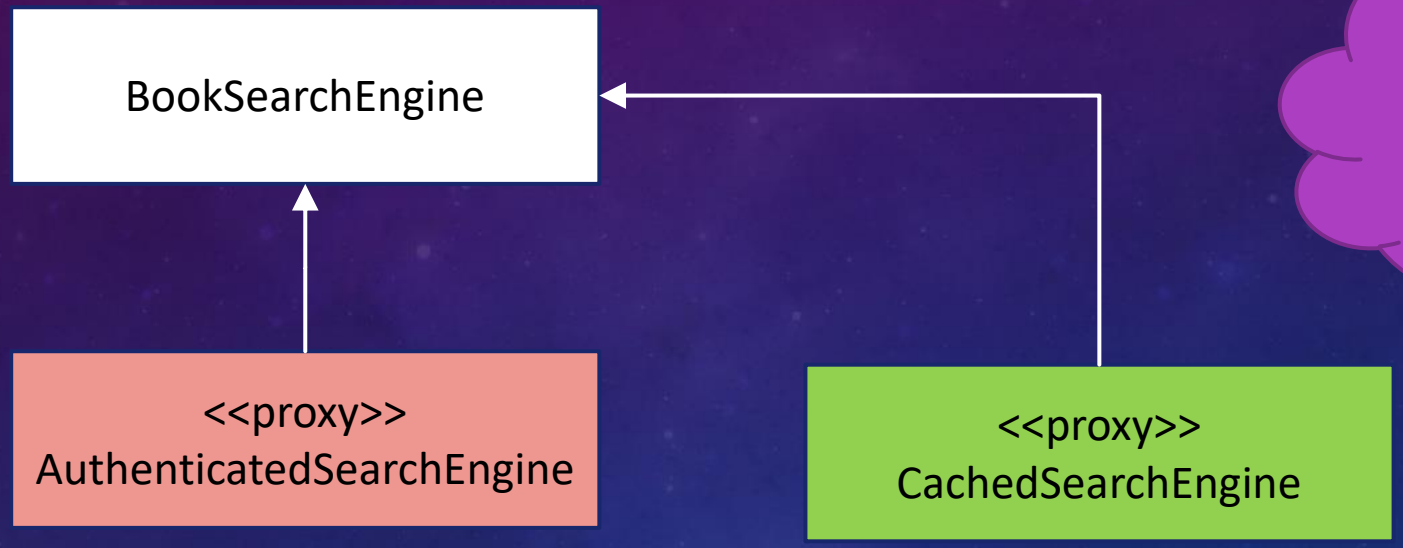
HOW ABOUT SECOND PROXY





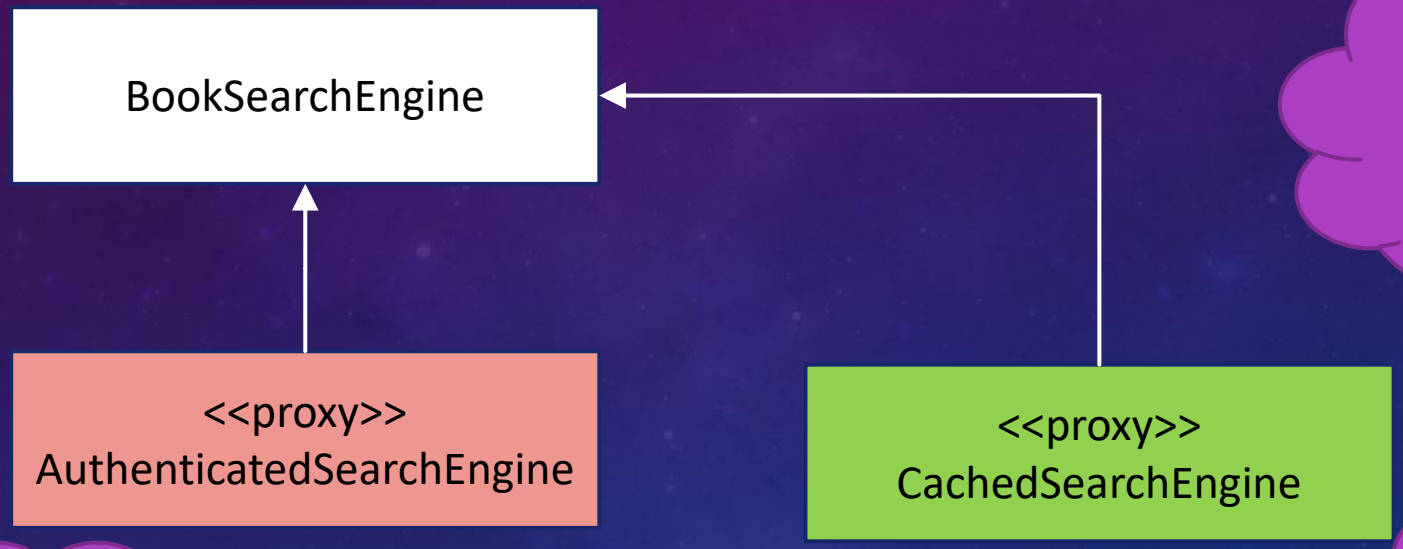
Where should CachedSearchEngine Inherit From

HOW ABOUT SECOND PROXY



How will Proxy chain work???

HOW ABOUT PROXY CHAINING?

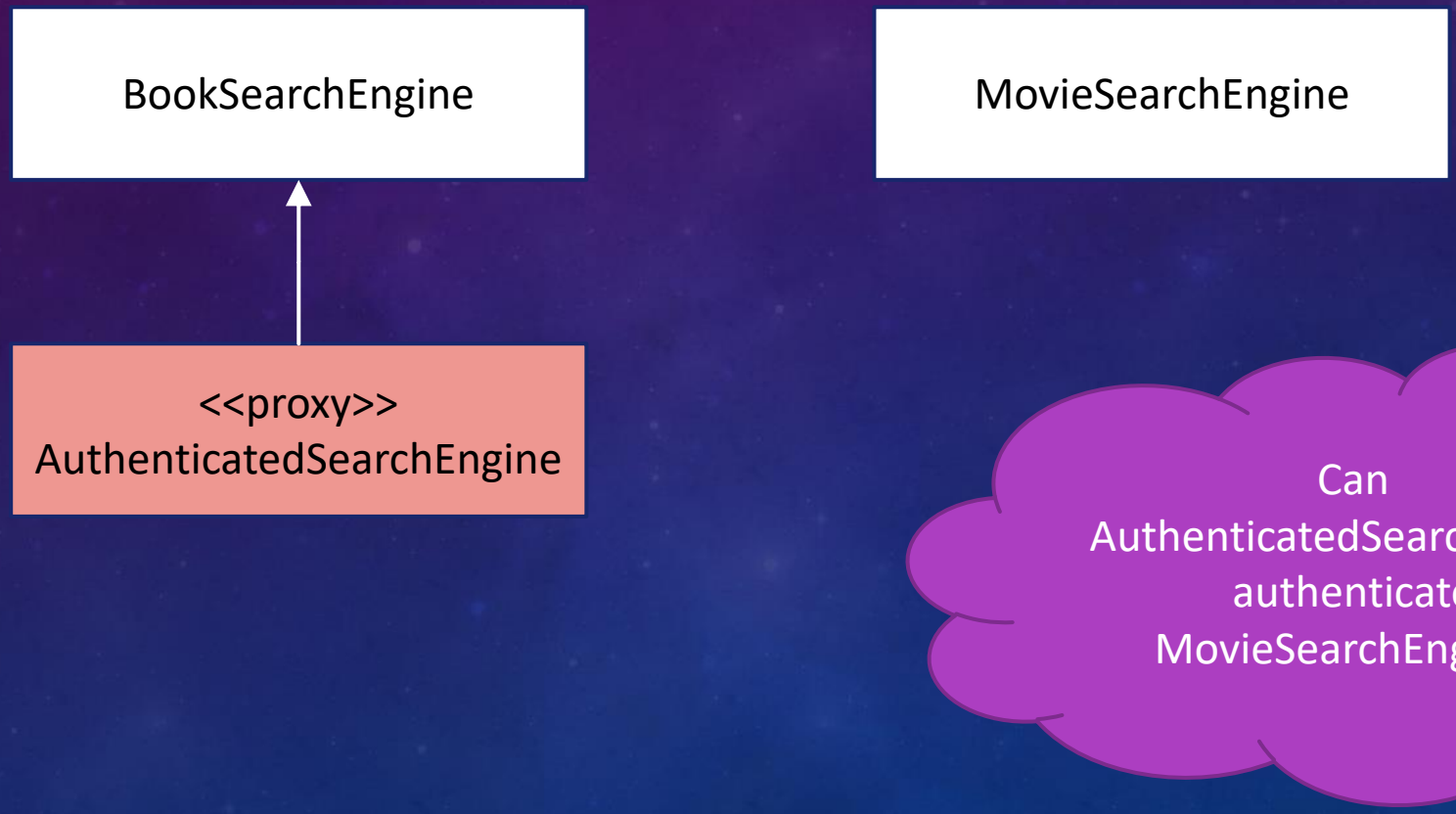


How will Proxy chain work???

what if I want no Authentication caching?

we had same problem earlier? Is the solution equally simple?

PROXYING ANOTHER TARGET



PROXYING ANOTHER TARGET

what should be the name of this class

What should be name of this class?

what is the code difference between these 2 class?

INHERITANCE PROXY CODE

```
public class AuthenticatedBookSearchEngine extends BookSearchEngine
{
        public Result search(Query q){

                if( ! HttpContext.IsAuthenticated)
                        throw new SearchException();
                else
                        return base.search(q)



        }

}
```

Why can't I Reuse?

```
public class AuthenticatedMovieSearchEngine extends MovieSearchEngine
{
        public Result search(Query q){

                if( ! HttpContext.IsAuthenticated)
                        throw new SearchException();
                else
                        return base.search(q)



        }

}
```

INHERITANCE BLOCKS PATH OF REUSABILITY

LETS CREATE A BOOK CLASS

Book

Title
Author
Price
Borrower
DateBorrow
History

Borrow()
Return()

How about these extra properties?

Which class should they be in?

LETS CREATE A BOOK CLASS

What is the Relation?

Book

Title
Author
Price

Borrower
DateBorrow
History

Borrow()
Return()

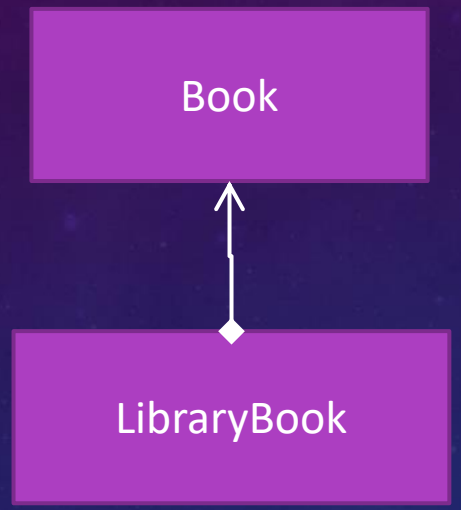
LibraryBook

HOW TO CREATE BOOK AND LIBRARY BOOK

```
Book [] books=new Book[10] ; //Ordinary Book

LibraryBook [] libs=new LibraryBoo[5]; //Library Book
```

- What is the real world difference in the two type of books?
- Can I move my book between Library Book and Book?

HAS A MODEL



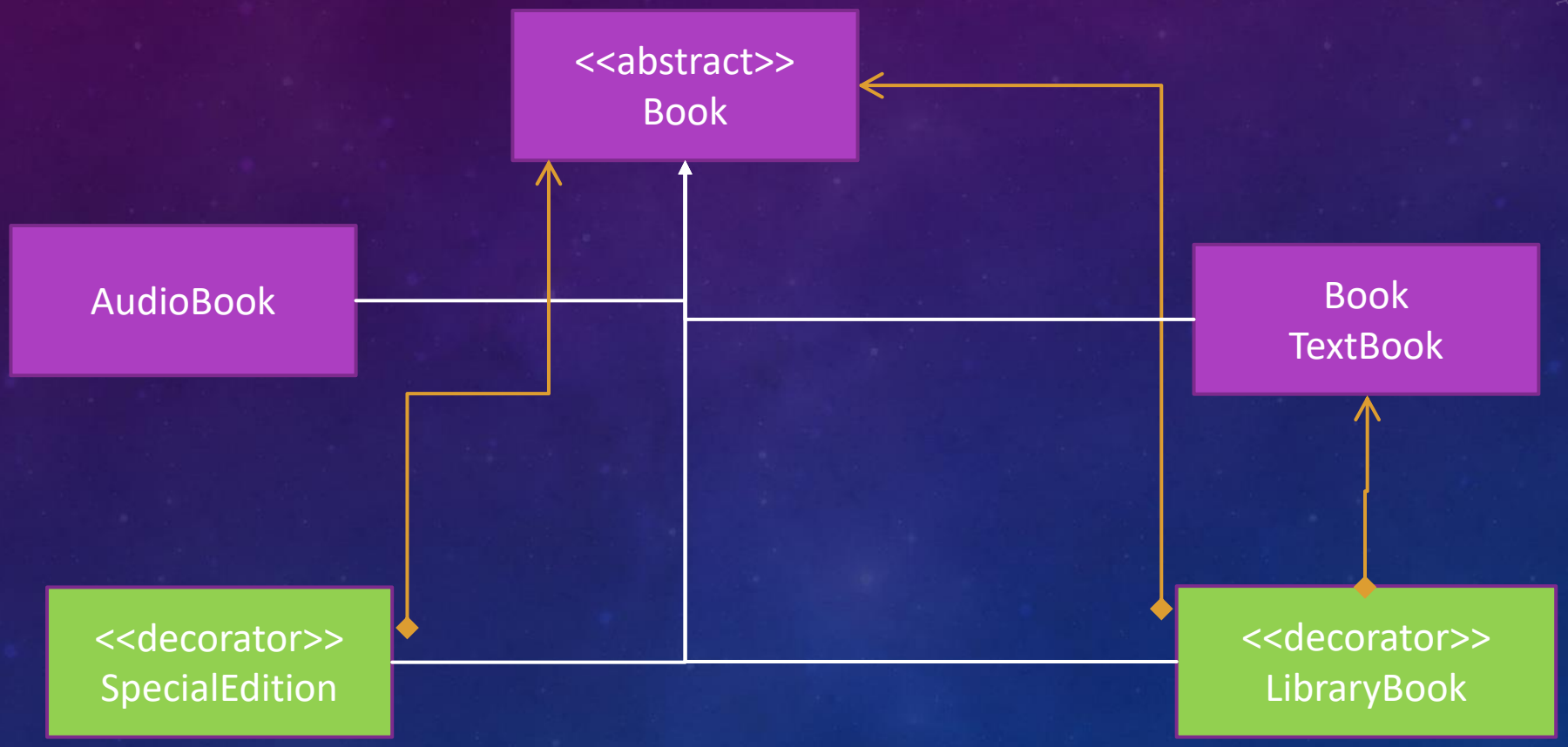

```
Book b1=new Book();   //Normal Book

Book b2=new Book();  //Normal Book

LibraryBook lb=new LibraryBook( b1);  //wrapped as Library Book

lb.GetTitle();  //can access target property

lb.Borrow();  // can add new behavior
```

But I Think Library Book is a Book not Has a Book

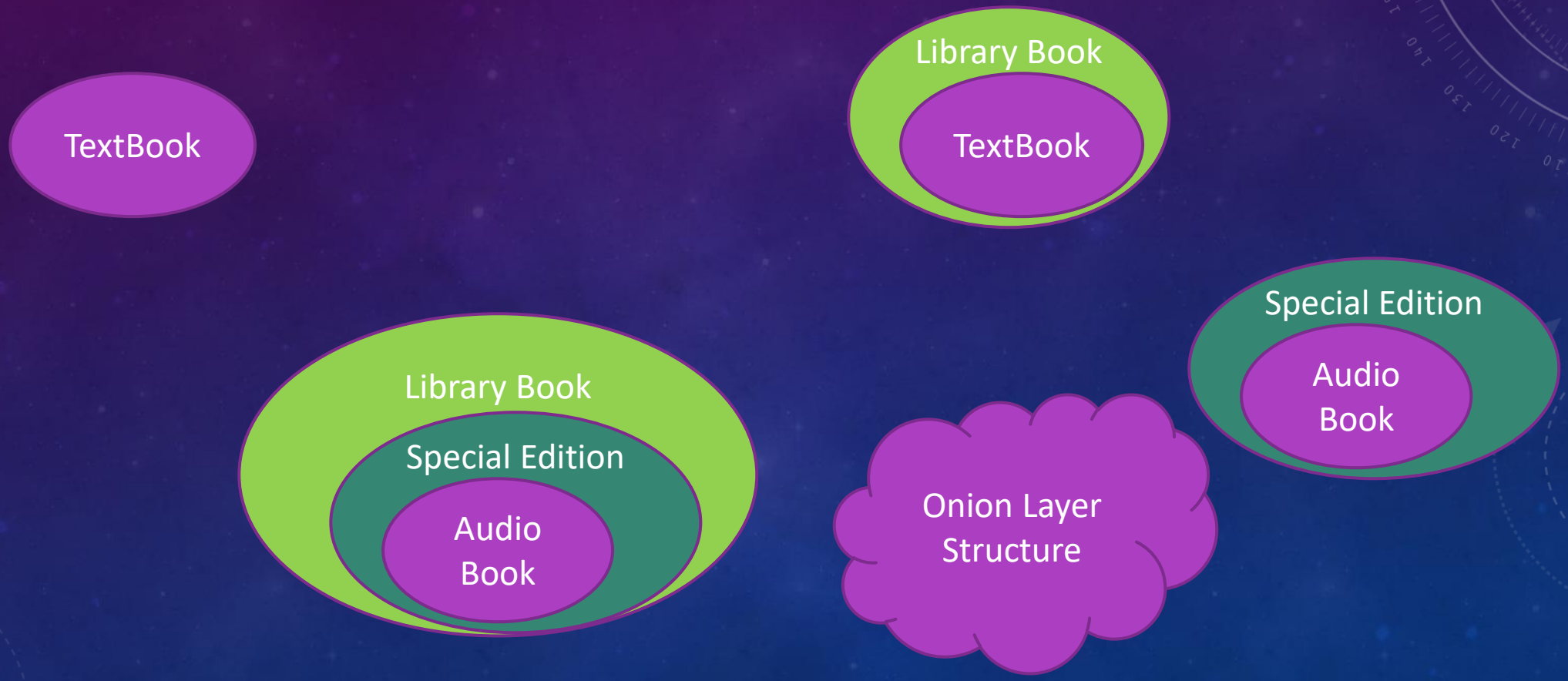
DECORATOR PATTERN

- DECORATOR IS A WRAPPER PATTERN TO ADD NEW BEHAVIOR
- DECORATOR DECORATES A TARGET TO GETHER ITS EXISTING FUNCTIONALITY
- DECORATOR CAN ADD NEW CAPABILITIES AT RUNTIME
- DECORATOR AND TARGET IMPLEMENTS SAME INTERFACE (LIKE PROXY)
- WHEREAS PROXY DOESN’T ADD NEW BEHAVIOR, DECORATOR ADDS NEW BEHAVIOR
- DECORATOR IS NOT TRANSPARENT TO THE CLIENT.
- DECORATOR CAN BE TREATED AS PROXY PLUS.

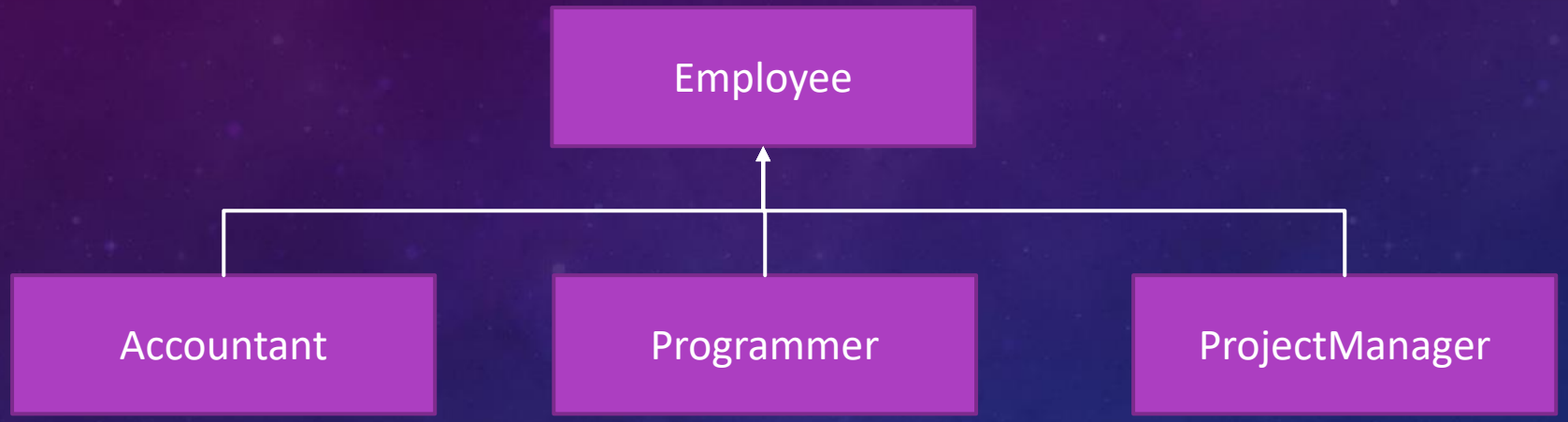
DECORATORS


<<abstract>>
Book
AudioBook
Book
TextBook
<<decorator>>
SpecialEdition
<<decorator>>
LibraryBook

APPLYING DECORATORS



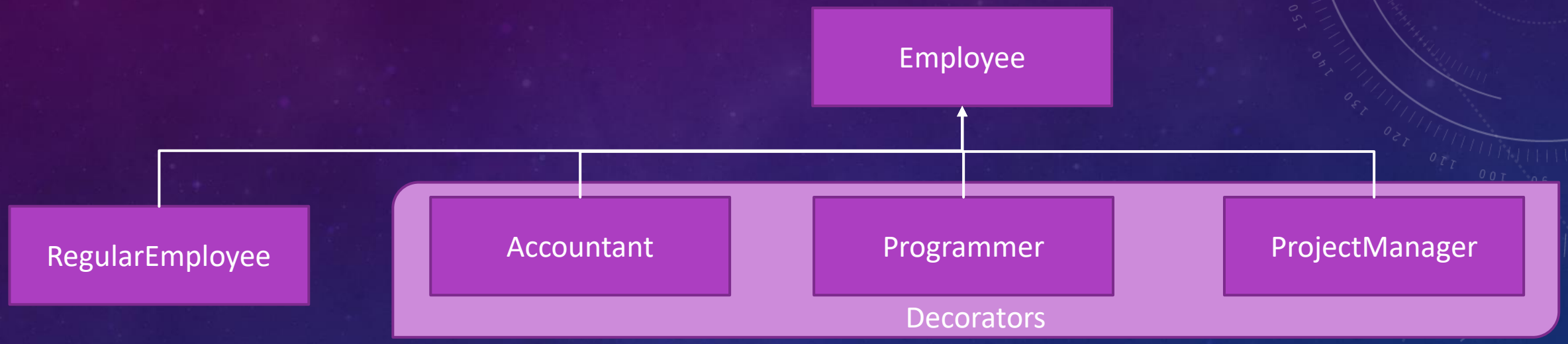
EMPLOYEE HIERARCHY



Programmer p =new Programmer("Rajiv Bagga");

How to Promote Rajiv Bagga to Project Manager?

EMPLOYEE DECORATOR



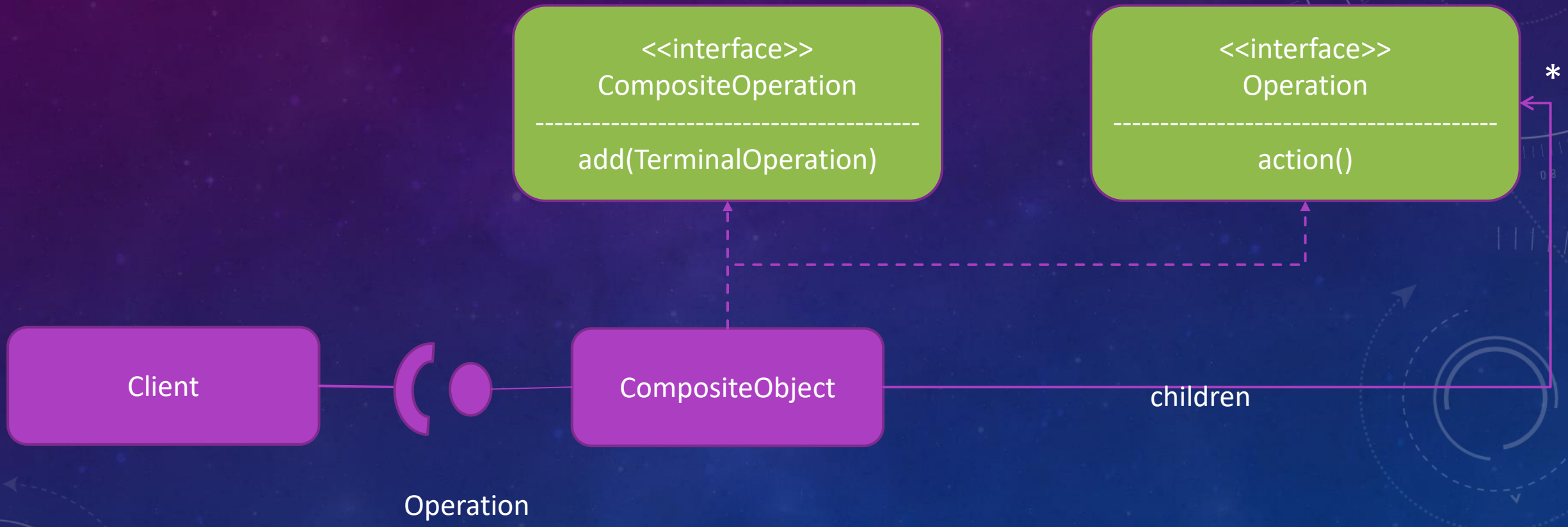
Lets Create an Employee → **Employee emp=new RegularEmployee("Rajiv Bagga");**

Make Him Programmer → **Programmer p=new Programmer(emp);**

Promote Him as Project Manager→ **ProjectManager pm=new ProjectManager(emp);**

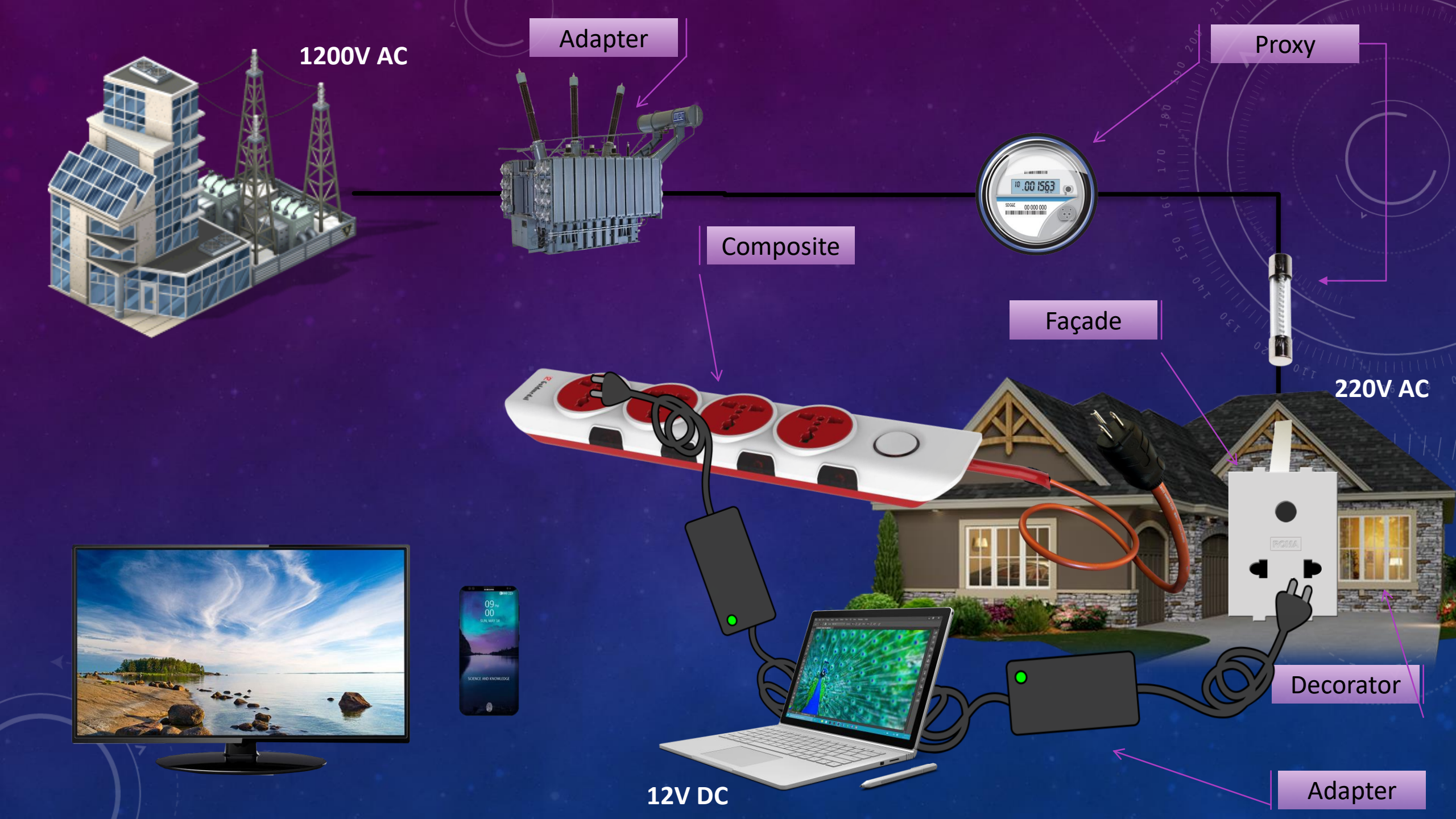
He is No more a Programmer→ **p.Dispose();**

COMPOSITE


<<interface>>
CompositeOperation
add(TerminalOperation)
<<interface>>
Operation
action()
*
Client
CompositeObject
children
Operation

COMPOSITE

- DEFINES AN INTERFACE FOR AN ACTION (TERMINAL COMPONENT)
- DEFINES ADDITIONAL INTERACE TO ADD CHILDREN ELEMENTS (COMPOSITE COMPONENTS)
- A COMPOSITE COMPONENT IS A TERMINAL COMPONENT
- A COMPOSITE COMPONENT MAY HAVE ONE OR MORE CHILDREN
  - EACH CHILDREN MAY BE COMPOSITE OR TERMINAL
- **FUNCTIONALITIES ARE DELEGATED TO THE CHILDREN ELEMENT**
- **A PATTERN TO SCALE A REQUIREMENT TO MULTIPLE ELEMENT**
- **YOU CAN USE ITEM, BUT YOU WANT TO USE MULTIPLE ITEM**


1200V AC
Adapter
Proxy
Composite
Façade
220V AC
Decorator
12V DC
Adapter

DECORATOR OR PROXY?

- Why is Fuse a Proxy and Switch a Decorator?
  - Fuse is Transparent to the Client
    - An Internal Detail Client Doesn’t Know about
  - Switch is an additional Interface that client Must learn to use
- How Do I Recognize the right pattern?
- Is it important to recognize the right pattern?

TANK WAR

Tank	Move()	Attack()	Defend()
Aggressive	Run to Enemy	Fire	Cover Fire
Defensive	Run away	Wait	Hide
Gurilla	Hapazard	Fire + Move	Duck

- Create Necessary Class Hierachy to Implement the Tanks!!!

TANK DESIGN


<<abstract>>
Tank

Attack()
Defend()
Move()
AggressiveTank
DefensiveTank
GurillaTank

DESIGN IMPLICATION

- Which company will create a defensive Tank?
  - Will There be a Gun installed on such a Tank?
  - Who would buy such a Tank?
- How will Aaggressive Tank return home after winning the war?
  - Remember, It always moves towards enemy
- Can Gurilla Tank ever move straight?

Think!!!
There is nothing called Aaggressive or Defensive Tank!!!

"Aaggressive" and "Defensive" are not Tank Types, They are Tank Moods!!!

DESIGN APPROACH 2

```
enum TankMode { Aaggressive, Defensive Gurilla }

class Tank{

    TankMode mode;

    public void Move(){

        switch(mode) {
            case TankMode.Aggresive :  RunToEnemy(); break;
            case TankMode.Defensive :  RunAway(); break;
            case TankMode.Gurilla: HapazardMove(); break;
        }

    }
    public void Attack() { …} //similar switch case
    public void Defend() {…}
}
```

SWITCH CASE A CLOSER LOOK!!!

```
public void Move(){

        switch(mode) {

            case TankMode.Aggresive :  RunToEnemy(); break;

            case TankMode.Defensive :  RunAway(); break;

            case TankMode.Guarding : HapazardMove(); break;
        }

    }
```

One Name

This is **Not** polymorphism – Its avoiding Polymorphism!!!

Different Form

Based on Context

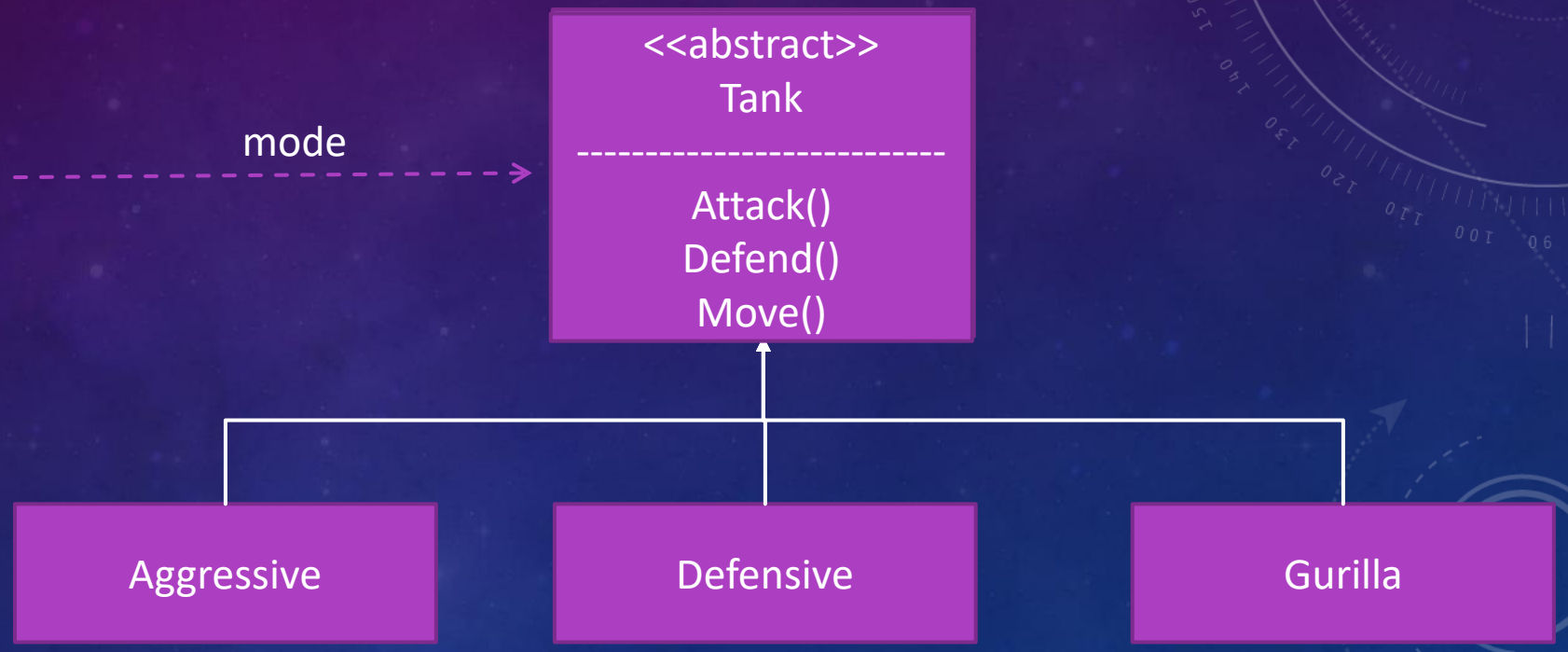
What is tomorrow we have a new Mode???

SWITCH CASE IMPLICATION

- SWITCH CASE VIOLATES SRP
- EACH CASE IS A DIFFERENT RESPONSIBILITY
- THEY ARE THE PRIMITIVE ALTERNATIVE OF POLYMORPHISM
- THEY VIOLATE OCP

THERE SHOULD BE NO ROOM FOR SWITCH-CASE IN AN OBJECT ORIENTED DESIGN

TANK DESIGN (REVISION 3)



TANK REVISION 3

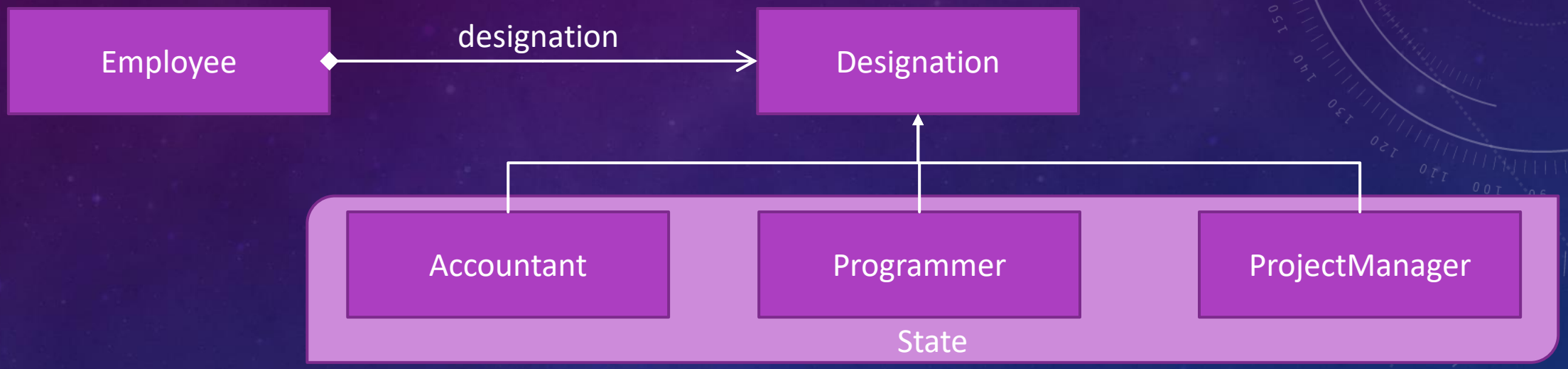
```
interface TankBehavior{
        void attackAction();
        void defendAction();
        void moveAction();
}

class AggresiveBehavior implements TankBehavior{
        public void attackAction() {  fireAtEnemey(); }
        public void defendAction() { coverFire(); }
        public void moveAction() { runTowardsEnemy(); }

}

class DefensiveBehavior implements TankBehavior {…}
class GurillaBehavior impelements TankBehavior{…}
```

```
class Tank {

        TankBehavior behavior;

        public void attack(){
                behavior.attackAction();
        }
        public void defend() {
                behavior.defendAction();
        }
        public void move(){
                behavior.moveAction();
        }

}
```

STATE PATTERN

- WHEN YOU CHANGE ONE OF THE POLYMORPHIC STATE OF AN OBJECT CHANGE ITS BEHAVIOR IN SUCH A WAY THAT APPEARS TO HAVE CHANGE ITS TYPE
- CHANGING THE **STATE** IS AKIN TO CHANING THE **TYPE** OF AN OBJECT
- THESE CHANGES ARE SUPPOSED TO BE INTERNAL
  - CLIENT MAY NOT BE AWARE OF SUCH A CHANGE

EMPLOYEE (STATE PATTERN)



Lets Create an Employee → **Employee emp=new Employee("Rajiv Bagga");**

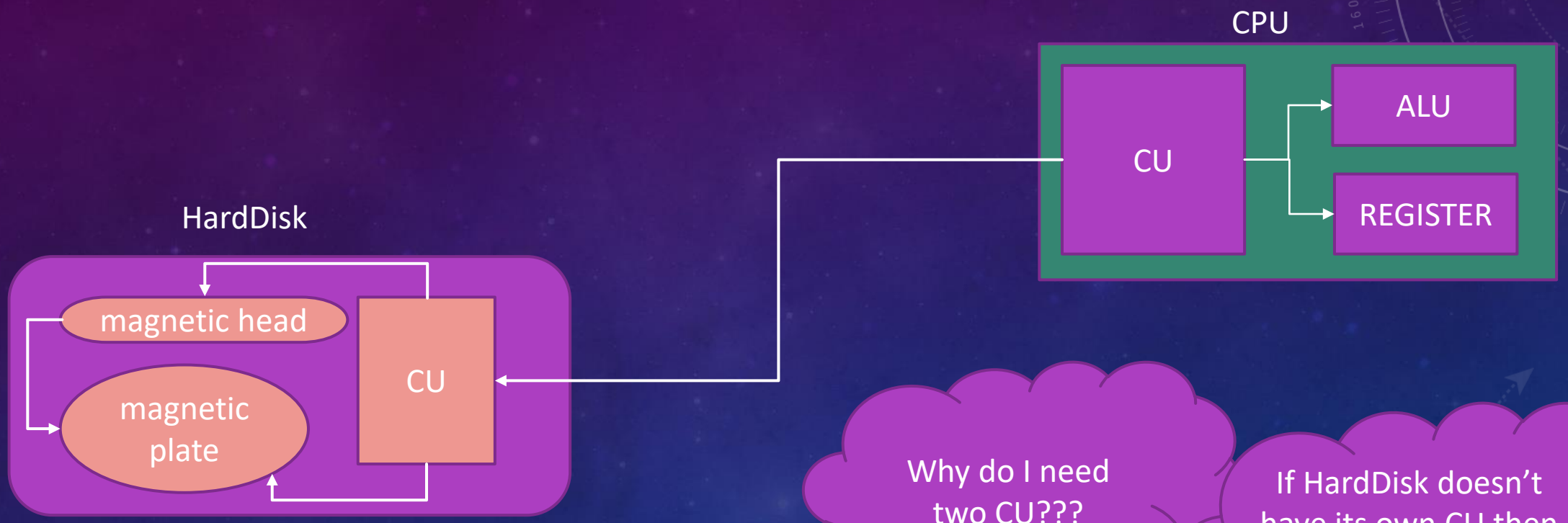
Make Him Programmer → **emp.setDesignation(new Programmer());**

Promote Him as Project Manager→ **emp.setDesignation(new ProjectManager());**

WHAT IS DECOUPLING

- DECOUPLING IS MAKING COMPONENTS INDEPENDENT OF EACH OTHER.
- COHESION STUDIES HOW WELL THE INTERNALS OF A SYSTEM ARE CONNECTED
- DECOUPLING STUDIES HOW INDEPENDENT IS ONE COMPONENT FROM ANOTHER
- DECOUPLING IS GETTING RID OF BAD COHESION.
- DESIGN GOAL IS HIGH COHESION (FUNCTION) AND LOW COUPLING
- Isn’t HIGH COHESION AND LOW COUPLING A CONTRADICTION

HIGH COHESION LEADS TO LOW COUPLING

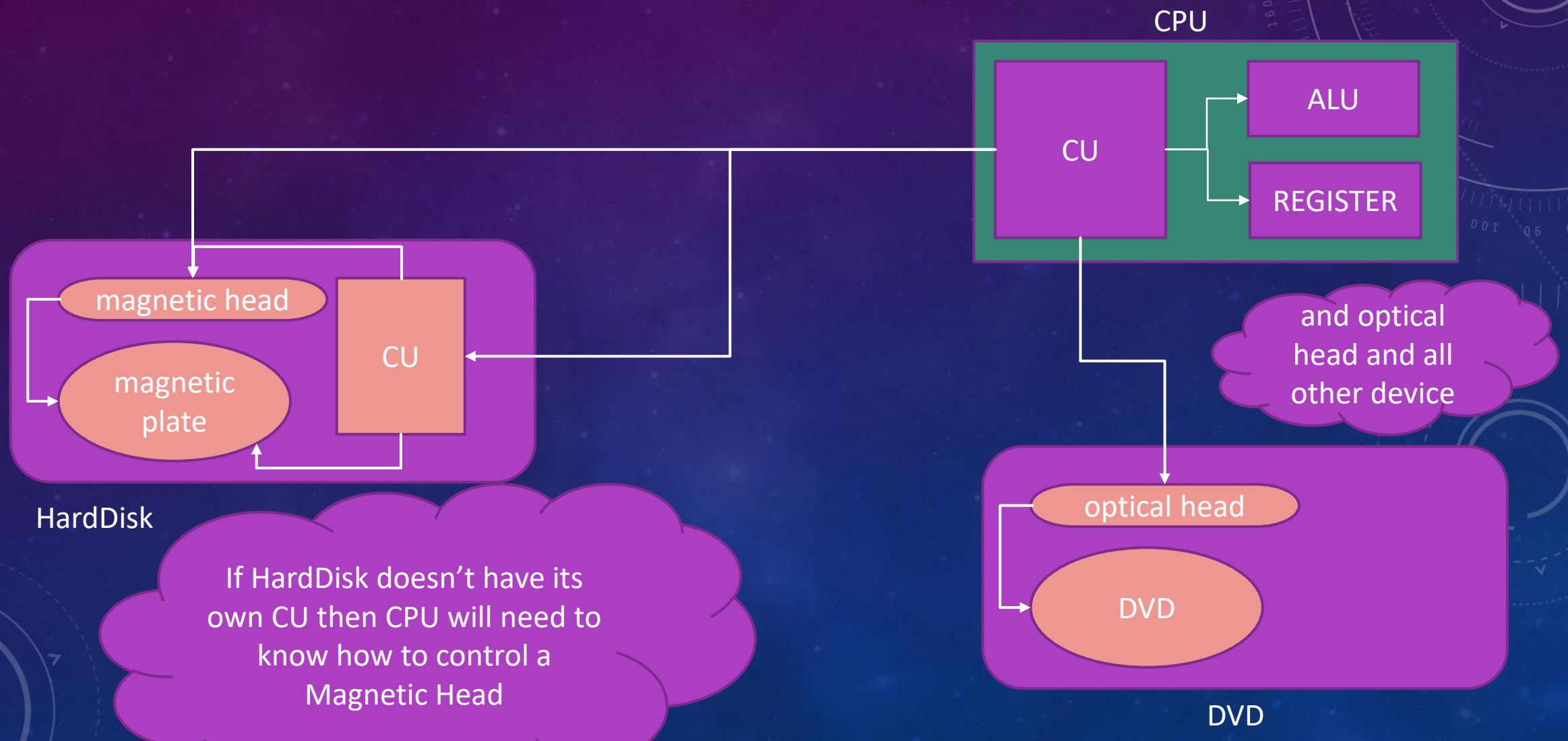


Why do I need two CU???

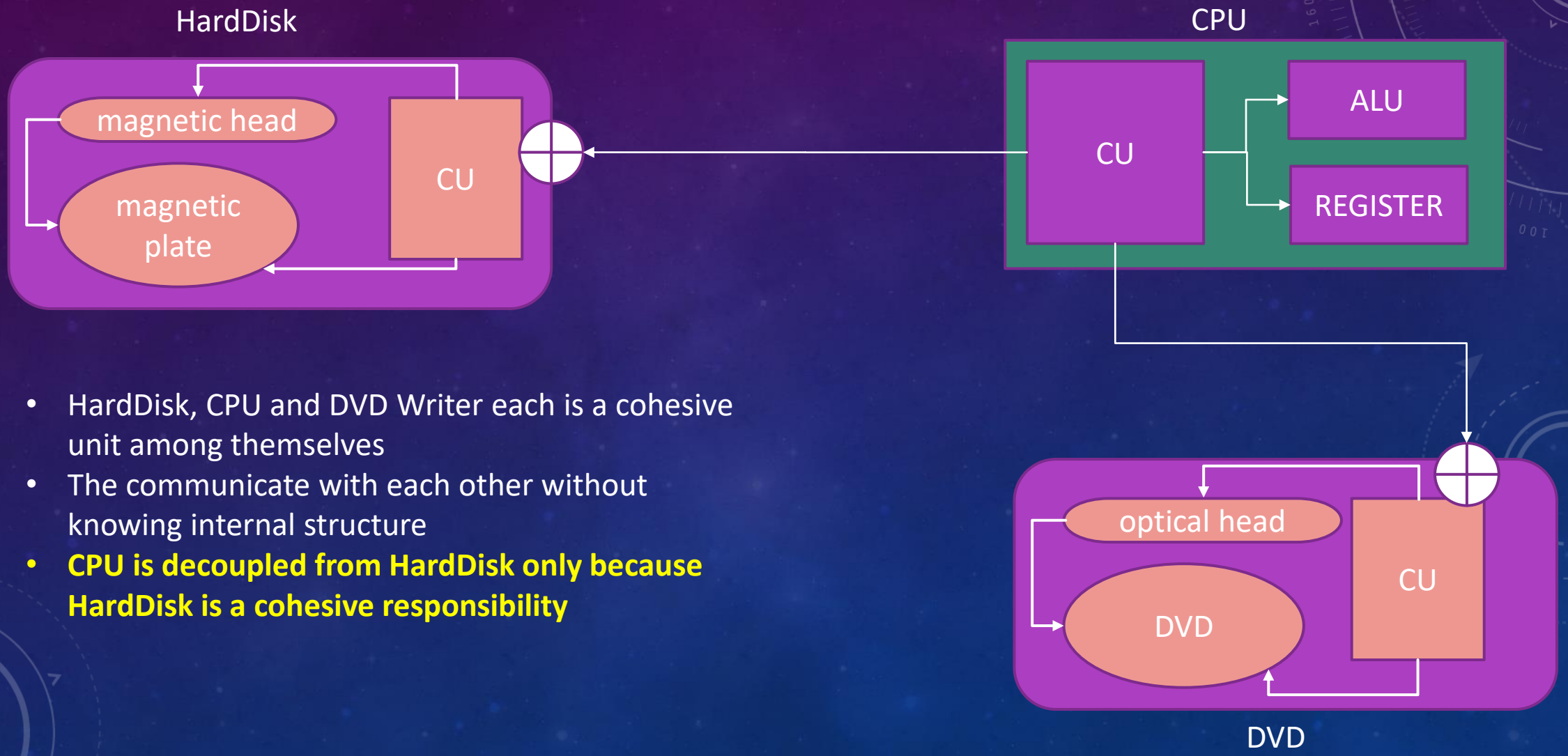
If HardDisk doesn't have its own CU then CPU will need to know how to control a Magnetic Head

High Cohesion

HIGH COHESION LEADS TO LOW COUPLING



HIGH COHESION LEADS TO LOW COUPLING



- HardDisk, CPU and DVD Writer each is a cohesive unit among themselves
- The communicate with each other without knowing internal structure
- **CPU is decoupled from HardDisk only because HardDisk is a cohesive responsibility**

Decoupling is Getting Rid of Bad Cohesion

BEHAVIORAL PATTERNS

HOW OBJECTS COMMUNICATE WITH EACH OTHER

WHAT IS BEHAVIORAL PATTERN

- CAN ALSO BE REFERRED AS COMMUNICATIONAL PATTERN
- BEHAVIORAL PATTERNS DEFINES HOW OBJECTS BEHAVE (COMMUNICATE) WITH EACH OTHER WITHOUT KNOWING THE STRUCTURAL DETAILS
- IT EMPHASIZES ON DECOUPLED DESIGN
  - COMPONENTS CAN BE DEVELOPED INDEPENDENTLY
  - COMPONENTS CAN TALK TO EACH OTHER WITHOUT KNOWING THEIR STRUCTURE

STRUCTURAL VS BEHAVIORAL PATTERNS

- STRUCTURAL PATTERNS ARE ABOUT COHESION
  - HOW MANY OBJECTS COMBINE TO WORK AS A SINGLE UNIT
- BEHAVIORAL PATTERNS ARE ABOUT LOOSE COUPLING
  - HOW INDEPENDENT IS ONE COMPONENT FROM ANOTHER
- STRUCTURAL PATTERNS ARE ABOUT INTERNAL CONNECTIVITY OF PARTS OF A SYSTEM
- BEHAVIORAL PATTERNS ARE ABOUT INDEPENDENCE OF ONE SYSTEM FROM ANOTHER.
- REMEMBER THE TWO SET COMPLEMENT EACH OTHER
  - WITHOUT PROPER COHESION THERE CAN BE NO DECOUPLING

CASE STUDY – LIST

Represents a List who value can be accessed using a integer index.

<<interface>>
IndexedList

---

add(Object o)
Object get(int pos)
void set(int pos, Object o)
int length()

Uses a Linked Set of Nodes to represent the values

Uses array to store the values. The array is resized dynamically based on the need

LinkedList

Node first
Node last
int count

-locate()

Different Internal structure same interface

DynamicArray

Object [] values
int size
int growSize

- ensureCapacity()

LINKED LIST VS DYNAMIC ARRAY (STRUCTURE)

```
class LinkedList{

    class Node {
        public Object value;
        public Node next;
        public Node previous
    }

    Node first, last;
    int count;

    …

}
```

```
class DynamicArray{

    Object [] values;
    int count;
    int growSize;





    …

}
```

LINKED LIST VS DYNAMIC ARRAY (SPECIAL METHODS)

```
class LinkedList{

    …

    private Node locate(int ndx){
        //search for Node at an index
    }

}
```

```
class DynamicArray{

    …

    private void ensureCapacity(){
        //grow the array if needed
    }

}
```

Internal needs may be different

LINKED LIST VS DYNAMIC ARRAY (COMMON METHODS)

```
class LinkedList{

    …
    public void add(Object value){

        Node n=new Node(value);
        …
    }

    public int get(int pos){
        Node n= locate(pos);
        …
    }

}
```

Different structure
Different implementation

```
class DynamicArray{

    …
    public void add(Object value){

        ensureCapacity();
        …
    }

    public int get(int pos){
        //return from array
        …
    }

}
```

LINKED LIST VS DYNAMIC ARRAY (TOSTRING)

```
class LinkedList{

    public String toString(){

        String str= “LinkedList(\t”;

        for(Node n=first;  n!=null; n=n.next){
            str+= n.item+”\t”;
        }

        str+=“)”;

        return str;
    }
}
```

```
class DynamicArray{

    public String toString(){

        String str= “DynamicArray(\t”;

        for(int i=0;i<count;i++){
            str+= values[i]+”\t”;
        }

        str+=“)”;

        return str;
    }

}
```

Important Code!!!

LETS CREATE BASIC TEST CLIENT FOR LINKEDLIST

```
void main(){

        LinkedList list=new LinkedList();

        Object [] testData = { 2, 3, 9 , 4 , 8 } ;

        for ( Object value in testData)
                list.add(value);

        System.out.println(list)

}
```

expected output:

LinkedList(2 3 9 4 8)

WE NEED MORE FEATURES - INDEXOF

```
void main(){
        LinkedList list=new LinkedList();
        fillList(list); //adds 2, 3, 9, 4, 8

        System.out.println("index of 9 is " + list.indexOf(9));
        System.out.println("index of 2 is " + list.indexOf(2));
        System.out.println("index of 5 is " + list.indexOf(5));
}
```

expected output:

index of 9 is 2
index of 2 is 0
index of 5 is -1

5 is not present in the list

```
class LinkedList{
…
        public int indexOf( Object value ) {
                int ndx = 0;

                for(Node n = first ; n != null ; n= n.next ) {
                        if ( n.value.equals(value))
                                return ndx;
                        else
                                ndx++;

                }
                return -1;

        }
…
}
```

WE NEED MORE FEATURES - SHOWALL

```
void main(){
        LinkedList list=new LinkedList();
        fillList(list); //adds 2, 3, 9, 4, 8

        list.showAll();
}
```

```
class LinkedList{
…
        public void showAll() {




                for(Node n = first ; n != null ; n= n.next ) {
                        System.out.println( n.value);
                }




        }
…
}
```

expected output:

2
3
9
4
8

WE NEED MORE FEATURES - SAVE

```
void main(){
        LinkedList list=new LinkedList();
        fillList(list); //adds 2, 3, 9, 4, 8

        list.save(new PrintStream("c:/temp/numbers.txt"));
}
```

```
2
3
9
4
8
```

This data should be saved to the specified file

```
class LinkedList{
…
        public void save(PrintStream ps) {



                for(Node n = first ; n != null ; n= n.next ) {
                        ps.println(n.value);
                }

                ps.Flush();



        }
…
}
```

WE NEED MORE FEATURES - AVERAGE

```
void main(){
        LinkedList list=new LinkedList();
        fillList(list); //adds 2, 3, 9, 4, 8

        double avg= list.average();
        System.out.prinltn("average is "+ avg)
}
```

```
average is 5.2
```

```
class LinkedList{
…
        public double average() {


                double sum=0;
                int count = 0;

                for(Node n = first ; n != null ; n= n.next ) {
                        sum += n.value ;
                        count ++ ;
                }

                return sum / count ;
        }
…
}
```

You can't average Object!!!

WHAT IS THE OBSERVATION???

```
class LinkedList{

    public int indexOf( Object value){
        ...
    }

    public double average() {
        ...
    }

    public void showAll() {

    }

    public void save(PrintStream ps){

    }

}
```

Each of These Function is Iterating

```
for(Node n=first; n!=null ; n= n.next ) {
    ...
}
```

Can we use a terminiology more generic than Iteration???

Redundant Code!!!

Partially Redundant

WHAT IS THE OBSERVATION???

```
class LinkedList{

    public int indexOf( Object value){
        …
    }

    public double average() {
        …
    }

    public void showAll() {

    }

    public void save(PrintStream ps){

    }

}
```

Can you average horses?

We don’t always use console !!!

Which format to Save - Xml/JSON/CSV ?

WHAT IS THE OBSERVATION???

```
class LinkedList{

    public int indexOf( Object value){
        …
    }

    public double average() {
        …
    }

    public void showAll() {

    }

    public void save(PrintStream ps){

    }

}
```

Too Many Methods

Violation of OCP

More Possibilities

show() → showLine(), showWide(),
average() → sum(), min(), max()
save() → saveXml(), saveJson(), saveCsv()

WHY IS OCP VIOLATED?

Violation of SRP

```
...

    public void add ( Object value){
        //adds to List
    }

    public Object get(int pos) {
        //gets from List
    }

    public void set(int pos, Object value) {
        //change within List
    }

    public int size(){
        //size of List
    }

...
```

Operates on the **List**

Operates on the **items** of the List and Not the List

Not the responsibility of the **List**

```
...

    public double average() {
        //averages items of the List
    }

    public void showAll() {
        //shows the items of the List
    }

    public void save(PrintStream ps){
        //saves the items of the List
    }

...
```

MORE OBSERVATION…

```
class LinkedList{

    public int indexOf( Object value){
        …
    }

    public double average() {
        …
    }

    public void showAll() {

    }

    public void save(PrintStream ps){

    }

}
```

We need similar functionality in Dynamic Array Also

Can We Reuse the same code in both cases???

```
class DynamicArray{

    public int indexOf( Object value){
        …
    }

    public double average() {
        …
    }

    public void showAll() {

    }

    public void save(PrintStream ps){

    }

}
```

AVERAGE FUNCTION

```
class LinkedList{

    public double average() {
        double sum=0;
        int count=0;

        for(Node n=first; n!=null; n=n.next){
            sum += n.value;
            count++;
        }

        return sum/count;
    }

}
```

```
class DynamicArray{

    public double average() {
        double sum=0;
        int count=0;

        for(int I = 0;  i<count; i++ ) {
            sum += values[i];  //value;
            count++;
        }

        return sum/count;
    }

}
```

Why can't we reuse the same code???

Because the Two Lists have different Internal Structure

AVERAGE FUNCTION – DIVING DEEPER

```
class LinkedList{

    public double average() {
        double sum=0;
        int count=0;

        for(Node n=first; n!=null; n=n.next){
            sum += n.value;
            count++;
        }

        return sum/count;
    }

}
```

1. What is Average???
   - Sum of Values by count of values

2. Why should average logic know about Node???

This function itself is a violation of SRP

1. Calculating Arithmetic Average

2. Knowing LinkedList structure.

AVERAGE FUNCTION – DIVING DEEPER

```
class LinkedList{

    public double average() {
        double sum=0;
        int count=0;

        for(Node n=first; n!=null; n=n.next){
            sum += n.value;
            count++;
        }

        return sum/count;
    }

}
```

1. Two Unrelated code is merged together
   - Bad Cohesion.
2. Separating Bad Cohesion is Decoupling
3. How?

Apply D.R.Y.

1. Encapsulate Whatever Repeats

2. Abstract whatever changes

APPLY DRY ENCAPSULATE WHATEVER REPEATS

```
public double average() {
        double sum=0;
        int count=0;
        for(Node n=first; n!=null; n=n.next){
                sum += n.value;
                count++;
        }
        return sum/count;
}

public int indexOf(Object obj){
        int ndx = 0;
        for(Node n = first ; n != null ; n= n.next ) {
                if ( n.value.equals(value))
                        return ndx;
                else
                        ndx++;
        }
        return -1;
}
```

APPLY DRY ABSTRACT WHATEVER CHANGES

```
public double average() {
        double sum=0;
        int count=0;
        for(Node n=first; n!=null; n=n.next){
                sum += n.value;
                count++;
        }
        return sum/count;
}

public int indexOf(Object obj){
        int ndx = 0;
        for(Node n = first ; n != null ; n= n.next ) {
                if ( n.value.equals(value))
                        return ndx;
                else
                        ndx++;
        }
        return -1;
}
```

what is the abstraction for

```
public  ???          ??? (???) {
        ???
        for(Node n = first ; n != null ; n= n.next ) {
                ???
        }
        return ???;

}
```

APPLY DRY ABSTRACT WHATEVER CHANGES

```
public double average() {
        double sum=0;
        int count=0;
        for(Node n=first; n!=null; n=n.next){
                sum += n.value;
                count++;
        }
        return sum/count;
}

public int indexOf(Object obj){
        int ndx = 0;
        for(Node n = first ; n != null ; n= n.next ) {
                if ( n.value.equals(value))
                        return ndx;
                else
                        ndx++;
        }
        return -1;
}
```

what is the abstraction for

```
public  T     ???          (???) {
        ???
        init();
        for(Node n = first ; n != null ; n= n.next ) {
                ???
        }
        return ???;

}
```

APPLY DRY ABSTRACT WHATEVER CHANGES

```
public double average() {
        double sum=0;
        int count=0;
        for(Node n=first; n!=null; n=n.next){
                sum += n.value;
                count++;
        }
        return sum/count;
}

public int indexOf(Object obj){
        int ndx = 0;
        for(Node n = first ; n != null ; n= n.next ) {
                if ( n.value.equals(value))
                        return ndx;
                else
                        ndx++;
        }
        return -1;
}
```

what is the abstraction for

```
public  T     ???          (???) {
        init();
        for(Node n = first ; n != null ; n= n.next ) {
                process( n.value);
        }
        return ???;

}
```

APPLY DRY ABSTRACT WHATEVER CHANGES

```
public double average() {
        double sum=0;
        int count=0;
        for(Node n=first; n!=null; n=n.next){
                sum += n.value;
                count++;
        }
        return sum/count;
}

public int indexOf(Object obj){
        int ndx = 0;
        for(Node n = first ; n != null ; n= n.next ) {
                if ( n.value.equals(value))
                        return ndx;
                else
                        ndx++;
        }
        return -1;
}
```

what is the abstraction for

```
public  T     ???          (???) {
        init();
        for(Node n = first ; n != null ; n= n.next ) {
                process( n.value);
        }
        return  finish()          ;

}
```

APPLY DRY

```
public double average() {
        double sum=0;
        int count=0;
        for(Node n=first; n!=null; n=n.next){
                sum += n.value;
                count++;
        }
        return sum/count;
}

public int indexOf(Object obj){
        int ndx = 0;
        for(Node n = first ; n != null ; n= n.next ) {
                if ( n.value.equals(value))
                        return ndx;
                else
                        ndx++;
        }
        return -1;
}
```

ABSTRACT WHATEVER CHANGES

```
public  T     ???        (???   x ) {
        x.init();
        for(Node n = first ; n != null ; n= n.next ) {
                x.process(n.value);
        }
        return  x.finish()        ;

}
```

Note
init() , process() & finish()
s not part of the list

How would we get this
'x'
object???

What is the Type
of 'x'?

APPLY DRY ABSTRACT WHATEVER CHANGES

```
public double average() {
        double sum=0;
        int count=0;
        for(Node n=first; n!=null; n=n.next){
                sum += n.value;
                count++;
        }
        return sum/count;
}

public int indexOf(Object obj){
        int ndx = 0;
        for(Node n = first ; n != null ; n= n.next ) {
                if ( n.value.equals(value))
                        return ndx;
                else
                        ndx++;
        }
        return -1;
}
```

```
public  T     ???          ( ??? Task     x ) {
        x . init();
        for(Node n = first ; n != null ; n= n.next ) {
                x. process(n.value);
        }
        return  x. finish()        ;

}
```

```
public interface     Task        {

        void init();
        void process(Object value);
        T finish();

}
```

what should we call this interface?

APPLY DRY ABSTRACT WHATEVER CHANGES

```
public double average() {
        double sum=0;
        int count=0;
        for(Node n=first; n!=null; n=n.next){
                sum += n.value;
                count++;
        }
        return sum/count;
}

public int indexOf(Object obj){
        int ndx = 0;
        for(Node n = first ; n != null ; n= n.next ) {
                if ( n.value.equals(value))
                        return ndx;
                else
                        ndx++;
        }
        return -1;
}
```

```
public  T   execute   ( Task    x ) {
        x . init();
        for(Node n = first ; n != null ; n= n.next ) {
                x. process(n.value);
        }
        return  x. finish()        ;

}
```

what should we call this function?

```
public interface    Task        {

        void init();
        void process(Object value);
        T finish();

}
```

FINAL DESIGN

```
public interface Task<I,R> {

        void init();
        void process(I value);
        R finish();
}
```

```
public interface List<T> {

        …
        <R> R execute( Task<T,R> task ) ;
        …
}
```

```
public class LinkedList<T> implements List<T> {

        Node first;
        …
        public <R> R execute ( Task<T,R> task ) {

                task.init();

                for(Node n=first; n!=null; n=n.next) {
                        task.process( n.value);
                }

                return n.finish();
        }
        …
}
```

CLIENT SIDE - AVERAGE

```
public class LinkedList<T> implements List<T> {

    Node first;
    …
    public <R> R execute ( Task<T,R> task ) {

        task.init();

        for(Node n=first; n!=null; n=n.next) {
            task.process( n.value);
        }

        return n.finish();
    }
    …
}
```

```
public class AverageTask<T extends Number> implements
Task< T, Double> {
    int count;
    double sum;
    public void init() { count=0; sum=0; }
    public void process(T value) {  sum+=value; count++; }
    R finish() { return sum/count; }
}
```

```
void main(){

    AverageTask<Integer> avg=new AverageTask<>();

    double result= list.execute(avg);
}
```

FUNCTIONAL PROGRAMMING

- THERE ARE THREE KEY ELEMENTS OF THE DESIGN

1. SERVICE LAYER
   - PROVIDES BASIC STEPS WHICH ARE COMMON FOR DIFFERENT OPERATION
   - IMPLEMENTS STRUCTURAL ASPECT RELATED TO SERVICE LAYER
2. CLIENT
   - THE USER OF THE SERVICE
   - NEEDS TO USE THE SERVICE
3. FUNCTION OBJECT
   - THE OBJECT THAT CLIENT PASSES TO THE SERVICE LAYER
   - SERVICE LAYER AND FUNCTION OBJECT WORKS TOGETHER TO COMPLETE THE REQUIREMENT OF CLIENT
   - CLIENT OWNS THE SERVICE OBJECT (GENERALLY)
   - SERVICE USES THE FUNCTION OBJECT

CALLBACK OBJECT

- A CALLBACK IS A FUNCTION OBJECT
  - OBJECT THAT REPRESENTS A TASK (ACTION)
  - GENERALLY A PARTIAL LOGIC THAT WORKS IN COMBINATION WITH SOME OTHER SERVICE
- A CALLBACK IS GENERaALLY OWNED BY A CLIENT
  - IT INCLUDES CLIENT SPECIFIC LOGIC
- CALLBACK IS PASSED TO THE SERVICE LAYER BY THE CLIENT
- SERVICE LAYER USES THE CALLBACK IN A SPECIFIC CONTEXT

CALLBACK, OWNER, USER

- WITH RESPECT TO CALLBACK OBJECT THERE ARE TWO PARTIES

1. OWNER OF CALLBACK
   - GENERALLY CLIENT
   - OWNER NEVER USES THE OBJECT ITSELF
   - ONLY PURPOSE IS TO PASS TO THE USER LAYER
2. USER OF CALLBACK
   - USER USES CALLBACK PASSED BY CLIENT
   - USER IS NOT AWARE OF THE ACTUAL CALLBACK PASSED
   - IT KNOWS CALLBACK BY ITS ABSTRACTION

BEHAVIORAL PATTERNS

DONT ASK HOW TO CREATE DIFFERENT BEHAVIORAL PATTERN

- ALMOST ALL BEHAVIORAL DESIGN PATTERNS ARE CALLBACK OBJECTS
- ALL BEHAVIORAL PATTERN FOLLOWS ALMOST SIMILAR STYLE OF PROGRAMMING SEMANTICALLY
- PATTERNS DIFFER IN THEIR INTENT OF USE
- DIFFERENT INTENT – DIFFERENT PATTERN

ASK WHY TO CREATE??

COMMAND PATTERN

- A CALLBACK TO PERFORM A TASK OR ACTION OR COMMAND
- SERVICE LAYER ACTS AS FACILITY PROVIDER
  - IT PROVIDES THE NECESSARY INFRASTRUCTURE REQUIRED TO EXECUTE THE COMMAND
  - IT EXECUTES THE COMMAND UNDER THE RIGHT CONTEXT
- MAIN ACTIVITY IS HANDLED BY THE **COMMAND** CALLBACK
- LIMITED INTERACTION BETWEEN SERVICE (FACILITY) AND THE COMMAND LAYER

COMMAND PATTERN - THREAD

- MULTI-THREADING IS GENERALLY IMPLEMENTED USING COMMAND DESIGN PATTERN
- **Thread** CLASS IS THE SERVICE LAYER
- **Runnable** Interface represents the **COMMAND** Object

COMMAND & MULTI-THREADING

```
//command interface
interface Runnable {
        void run();
}
```

```
//service layer
class Thread {
        Runnable runnable;

        public Thread(Runnable runnable){
                this.runnable=runnable;
        }
        public void start(){
                //launch a new thread
                //execute the command
                runnable.run();
        }

}
```

User

```
//command object
class FileSearchTask implements Runnable {
        public void run(){
                //search task
        }
}
```

command callback

```
//client
void main(){
        FileSearchTask task=new FileSearchTask();

        new Thread(task)
                . start();
}
```

Owner

MORE EXAMPLE OF COMMANDS

- PRINT COMMAND
  - MAY ENCAPSULATE A COMMAND PASSED TO PRINT MANAGER
  - PRINT MANAGER BASED ON PRIOIRTY AND AVAILABILITY MAY EXECUTE THE PRINT COMMAND
- GAME MOVE COMMAND
  - THINK OF A MULTIPLAYER GAME
  - ACTION BY ONE PLAYER MAY BE ENCAPSULATED AS A COMMAND OBJECT
  - THE OBJECT CAN BE PASSED ACROSS NETWORK
  - THE RECEIVING NETWORK WILL EXECUTE THE COMMAND TO REPLICATE THE MOVE
- SQL COMMAND
  - MAY ENCAPSULATE SQL COMMAND TO BE EXECUTED ON A DB
- SCHEDULED MAILER COMMAND
  - A SCHDULE KEEPER FRAMEWORK MAY HOLD ON TO COMMAND OBJECT TILL THE SCHEDULED TIME AND THEN EXECUTES IT

WHAT PATTERN WAS REPRESENTED BY LIST PROCESS AND TASK?

- ASK WHAT WAS THE GOAL?
- WHAT WERE WE DOING BEFORE WE INTRODUCED Process() function?

- WE WERE TRYING TO ADD MANY FACILITIES
  - AVERAGE
  - INDEXOF
  - SAVE
  - SHOW
  - SUM
  - MIN
- THE TASK CALLBACK ALLOWS US TO ADD NEW FUNCTIONALITY WITHOUT KNOWING THE STRUCTURE OF THE OBJECT.

Vistor Design Pattern

VISITOR DESIGN PATTERN

- VISITOR IS A **CALLBACK** WHICH ADDS NEW CAPABILITY TO THE SERVICE LAYER
- VISITOR ALLOWS YOU TO EXECUTE A **TASK** WITHOUT KNOWING THE INTERNAL STRUCTURE OF THE USER
- SERVICE LAYER PROVIDES NECESSARY DATA.
- THIS OF VISITOR AS AN EXTENSION SOCKET LIKE USB.
  - IT ALLOWS YOU TO PLUG ANY USEFUL SERVICE
  - THIS INTURN EXTENDS THE CAPABILITY OF COMPUTER

COMMAND VS VISITOR

- COMMAND EXECUTES A TASK WITH THE HELP OF SERVICE LAYER
  - THE TASK BENEFITS THE CLIENT
  - IT MAY NOT ADD VALUE TO THE SERVICE
- VISITOR LIKE COMMAND EXECUTES A TASK WITH THE HELP OF SERVICE LAYER
  - IT ADDS VALUE TO THE SERVICE LAYER

A vistor can be considered as a **Command** to add new capability

STRATEGY PATTERN (DECISION MAKER)

- STRATEGY IS ABSTRACTION FOR A FAMILY OF ALGORITHM
  - A FUNCTION OBJECT
- STRATEGY IS A CALLBACK TO SPECIALIZE A GENERIC ALOGRITHM
- SERVICE LAYER ENCAPSULATE A GENERIC ALGORITHM
- FEW STEPS OF THE GENERIC ALGORITHM MAY VARY BASED ON SPECIFIC CLIENT USE CASE
  - THOSE STEPS CAN BE SPECIALIZED BY **STRATEGY CALLBACK** PASSED BY THE CLIENT

CAN BE FIND THE AVERAGE OF A LIST OF HORSES?

- AVERAGE → (SUM OF NUMBERS) / (COUNT OF ITEMS)
- WE CAN COUNT TOTAL NUMBER OF HORSES IN A LIST
- BUT HOW DO I SUM A LIST OF HORSE?
- WRONG QUESTION.
- RIGHT QUESTION IS WHAT TO SUM (OR AVERAGE) ABOUT A LIST OF HORSE
  - AGE
  - PRICE
  - TOP SPEED
  - RACES WON
  - …

REVISITING AVERAGE

```
interface DoubleConverter<T> {
        double toDouble( T item) ;
}
```

```
public double average ( List<T> items, DoubleConverter<T> x   ){


        double sum = 0;

        double count = 0;


        for(int i=0;i<  items.length(); i++ ) {

                T  item = items.get(i);

                sum + =   x.toDouble ( item);

                count ++;

        }

        return sum/count;

}
```

Item is not a Number.
How to some it?

Simple Find a
way to convert
'T' to double

REVISITING AVERAGE

```
interface DoubleConverter<T> {
        double toDouble( T item) ;
}
```

```
public double average ( List<T> items, DoubleConverter<T> x   ){


        double sum = 0;

        double count = 0;


        for(int i=0;i<  items.length(); i++ ) {

                T  item = items.get(i);

                sum + =   x. toDouble ( item);

                count ++;

        }

        return sum/count;

}
```

```
class HorsePrice implements DoubleConverter<Horse> {
        public double toDouble(Horse horse){
                return horse.getPrice();
        }
}
```

```
void main(){

        LinkedList<Horse> horses = …

        double avg= average(horses, new HorsePrice());

}
```

MORE STRATEGIES

- SEARCH()
- SORT()
- FORMATTER()

OBSERVER A.K.A. PUBLISHER SUBSCRIBER

- DECOUPLES THE PRODUCER AND CONSUMER OF THE INFORMATION
- KEY PLAYERS
- OBSERVABLE
  - ANY OBJECT WHOSE STATE CHANGES INFREQUENTLY IN AN UNDERMINISTIC WAY
  - THERE ARE MANY OBSERVERS WHO MAY BE INTERESTED IN THE STATE CHANGE
- OBSERVER
  - OBJECTS THAT ARE INTERESTED IN OBSERVING THE STATE CHANGE OF OBSERVABLE

How will Observers know that Observable has changed.

APPROACH 1 – OBSERVER POLLS FOR UPDATE

Observable

constant polling required. No way to know when stock details is updated.CC

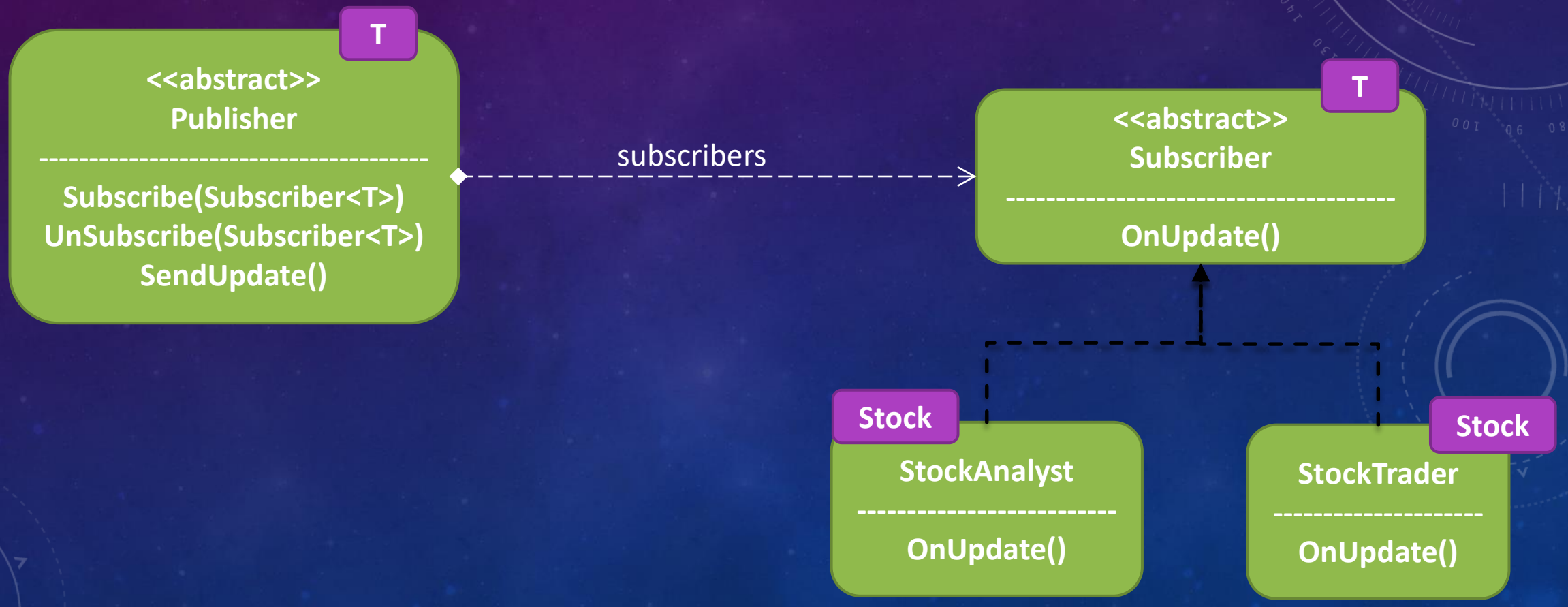
Polling

Observers

APPROACH 2 – OBSERVABLE PUSHES THE UPDATE

APPROACH 3 – PUBLISHER SUBSCRIBER PATTERN

PUBLISHER SUBSCRIBER CLASS DIAGRAM



PUBLISHER-SUBSCRIBER - WALKTHROUGH

```
interface Subscriber<T> {
        void onUpdate(T value);
}
```

```
public class Publisher<T> {

        List<Subscriber> subscribers=…

        public void subscribe(Subscriber s){
                subscribers.add(s);
        }


        public void sendUpdates(){
                T data = getData();
                for(Subscriber<T> s : subscribers){
                        s.onUpdate(data);
                }
        }

}
```

```
public class StockAnalyst extends Subscriber<Stock> {

        public void onUpdate (Stock stock){
                //analyse trend
        }

}

public class StockTrader extends Subscriber<Stock>{
        public void onUpdate (Stock stock){
                //buy sell decison
        }

}
```

MULTIPLE PATTERNS

- A design may include multiple patterns at one place.
- Each Pattern may have its own Role

CLOSER LOOK AT PUBLISHER

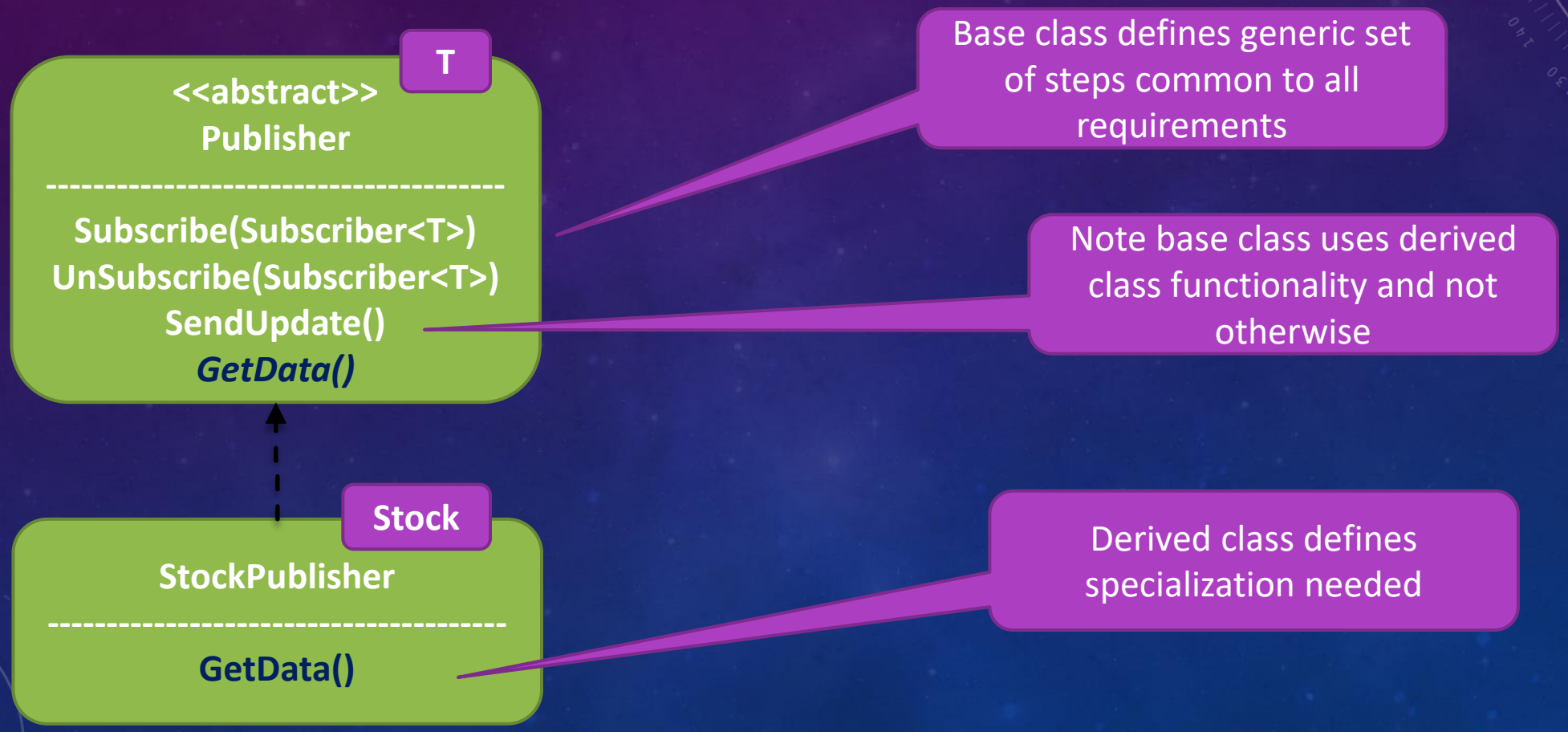
```
interface Subscriber<T> {
        void onUpdate(T value);
}
```

```
public    abstract    class Publisher<T> {

        List<Subscriber> subscribers=…

        public void add(Subscriber s){
                subscribers.add(s);
        }

        public void sendUpdates(){
                T data = getData();
                for(Subscriber<T> s : subscribers){
                        s.onUpdate(data);
                }
        }

        protected abstract T getData();
}
```

How will a generic publisher get the data???

```
public class StockPublisher extends Publisher<Stock> {

        protected T getData(){
                //your logic to get latest data here
        }

}
```

TEMPLATE DESIGN PATTERN



INHERITANCE VS REUSE

- Look into the method present in this class
- Is it really a **publisher logic?**
- **What is the name you would choose for this class?**

```
public class StockPublisher extends Publisher<Stock> {

    protected T getData(){
        //your logic to get latest data here
    }

}
```

```
public class StockDataProvider implements DataProvider<Stock>
{

    protected T getData(){
        //your logic to get latest data here
    }

}
```

PUBLISHER REVISITED

```
public class Publisher<T> {

    List<Subscriber> subscribers=…
    DataProvider<T> provider=…

    public void add(Subscriber s){
        subscribers.add(s);
    }

    public void sendUpdates(){
        T data = provider.getData();
        for(Subscriber<T> s : subscribers){
            s.onUpdate(data);
        }
    }

}
```

```
interface Subscriber<T> {
    void onUpdate(T value);
}
```

```
interface DataProvider<T> {
    T getData();
}
```

```
public class StockDataProvider implements
DataProvider<Stock>{

    protected T getData(){
        //your logic to get latest data here
    }

}
```