# DESIGN PATTERNS (CREATIONAL)

BY

VIVEK DUTTA MISHRA

## OBJECT CREATION CONCERNS

HOW TO CREATE AN OBJECT

## LIMITATIONS OF CONSTRUCTOR

- CONSTRUCTOR IS ALWAYS ABOUT **NEW**
- CONSTRUCTORS HAVE A MEANINGLESS NAME
- CONSTRUCTORA ARE NON-POLYMORPHIC
- CONSTRUCTOR TIES YOU TO IMPLEMENTATION
- CONSTRUCTOR BREAKS DESIGN RULES
- CONSTRUCTORS ARE NOT OBJECT ORIENTED

## CONSTRUCTOR → NEW

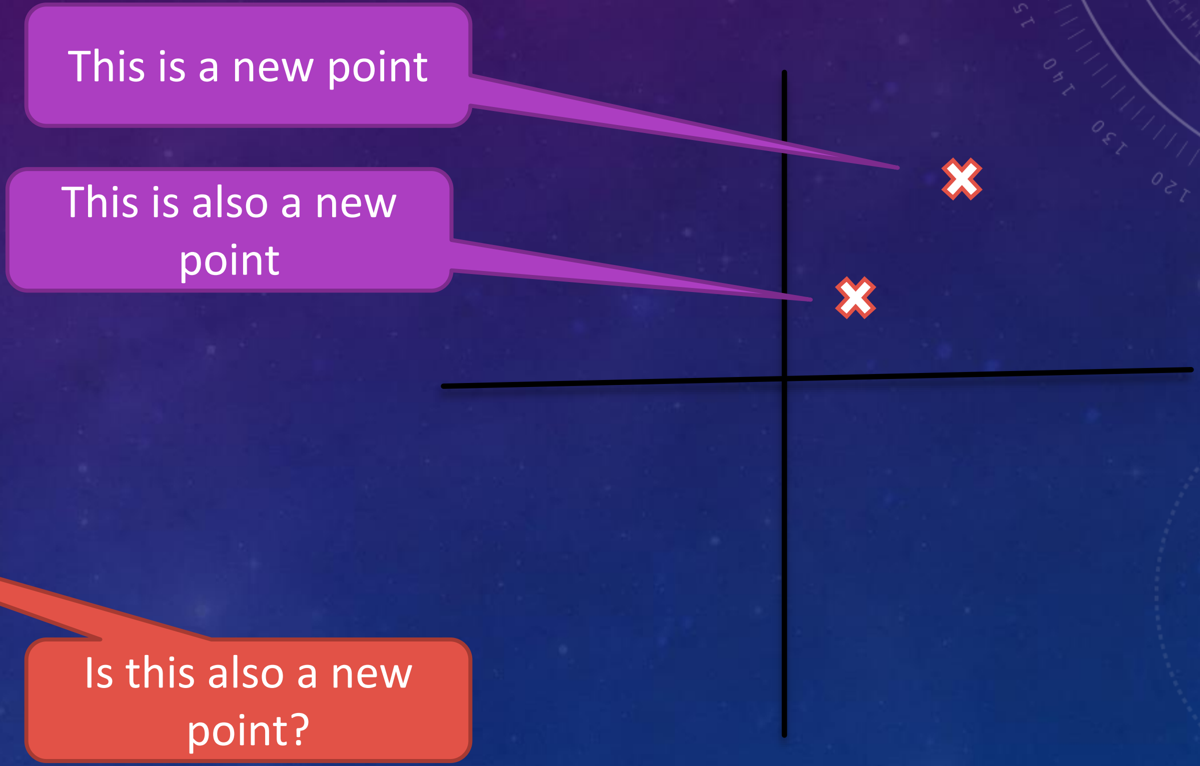
- THERE IS A DIFFERENCE BETWEEN NEEDING AND OBJECT AND NEEDING A NEW OBJECT
- CONSTRUCTOR IS ALWAYS ABOUT **NEW**
- NO LANGUAGE CONSTRUCT TO RETURN **EXISTING** OBJECT
- NO OBJECT REUSE
- MAY LEAD TO OPTIMIZATION ISSUE
- MAY LEAD TO CONCURRENCY ISSUE

## POINT USE CASE

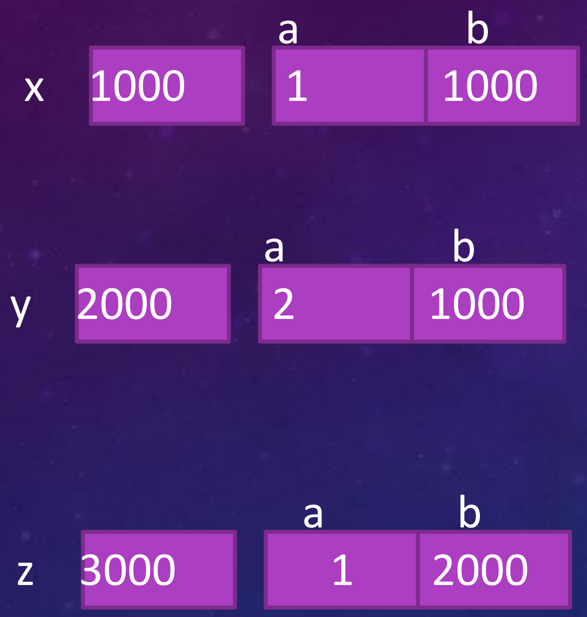
**Point p1 = new Point(3,4);**

**Point p2= new Point(1,1);**

**Point p3 = new Point(3,4);**



# BANK ACCOUNTS USE CASE



```
//10:00AM
BankAccount x=new BankAccount(1);
//10:01AM
BankAccount y= new BankAccount(2);
//10:02 AM
BankAccount z=new BankAccount(1);
```

**what will be the final balance if I withdraw 500 from account number 1?**

Are Any of them new Accounts or they all are existing?

## CONSTRUCTOR – MAIN PROBLEM

- IS CONSTRUCTOR STATIC OR NON-STATIC?
  - ARE THEY CLASS LEVEL OR OBJECT LEVEL?
- CLASS LEVEL A.K.A STATIC
  - BELONGS TO CLASS
  - DOESN'T NEED OBJECT TO INVOKE THEM
  - NO THIS POINTER
- OBJECT LEVEL A.K.A. NON STATIC
  - BELONGS TO OBJECT
  - CONTAINS THIS POINTER
  - INVOKED WITH OBJECT REFERENCE

## WHY CONSTRUCTOR IS NOT CLASS LEVEL

- CONTAINS THIS POINTER, STATIC DONT HAVE THIS POINTER
- STATIC METHODS NEED A CLASS NAME REFERENCE; CONSTRUCTORS DONT

## WHY CONSTRUCTOR IS NOT OBJECT LEVEL

- OBJECT LEVEL METHODS ARE INVOKED USING OBJECT REFERENCE
- CONSTRUCTORS CANT BE INVOKED USING OBJECT REFERENCE. EVER.
- OBJECT LEVEL METHODS CAN BE DEFINED AS A PART OF INTERFACE
  - CONSTRUCTOR CAN BE A PART OF ANY INTERFACE

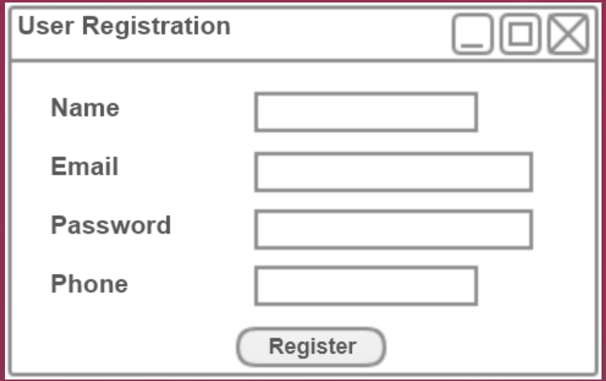
CONSTRUCTORS ARE THE CREATORS OF THE OBJECT NOT THE PART OF THE OBJECT

## CONSTRUCTORS HAVE A MEANINGLESS NAME

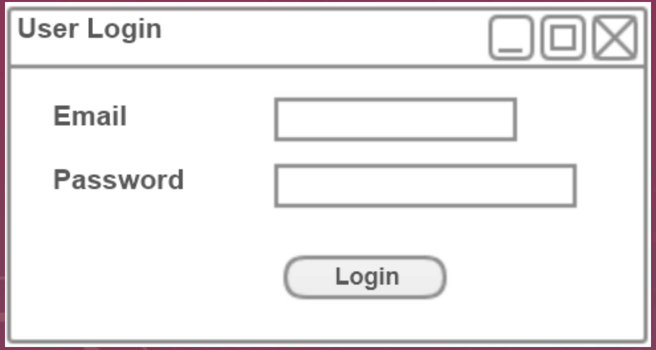
- CLASS NAME REPRESENTS THE OBJECT IT IS SUPPOSED TO MODEL
- A NAME VALID FOR A PRODUCT MAY NOT (GENERALLY) BE VALID FOR ITS CREATOR
- A NAME IS MEANINGFUL IF WE CAN UNDERSTAND ITS PURPOSE
- CONSTRUCTOR CAN SPECIFY AN OBJECT IS BEING CREATED BUT NOT
  - WHY
  - HOW
- I CANT CHOOSE CONSTRUCTOR NAME.
  - HOW CAN I BE SURE IF IT IS MEANINGFUL

# USER MANAGEMENT – USE CASE

Presentation Tier


User Registration

| | |
|---|---|
| Name | |
| Email | |
| Password | |
| Phone | |


User Login

| | |
|---|---|
| Email | |
| Password | |

```
class UserUI{

    public void OnRegister(){
        User user = new User(name,
                             email,
                             password,
                             phone);
    }

    public void OnLogin(){
        User user=new User(email,
                           password);
    }
}
```

Is this a **new** User?

Business Layer

```
class User{

    public void User(String name,
                     String email,
                     String password,
                     String phone){

        db.execute("insert into…");
    }

    public void User(String email,
                     String password){

        db.execute("select *…");

    }
}
```

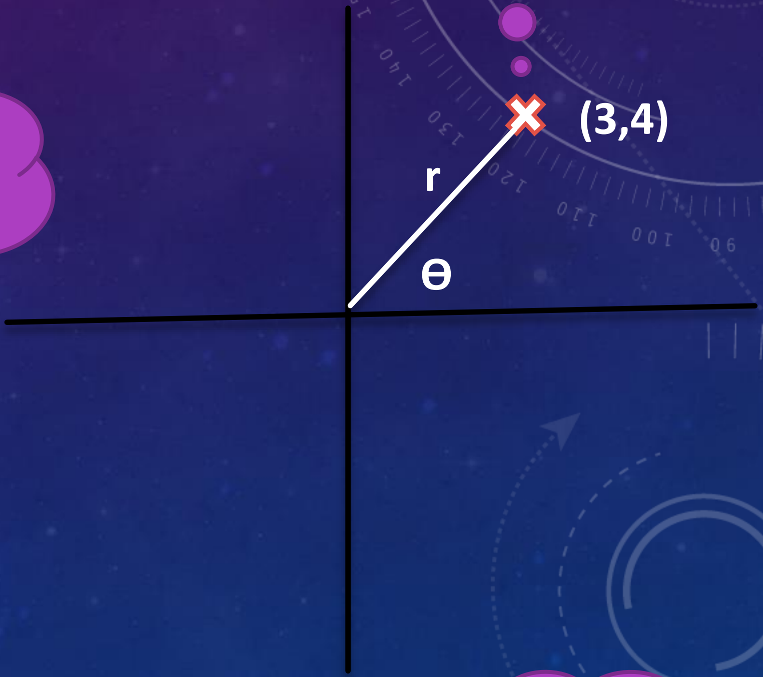
How do I know what the constructor's responsibility is?

## POINT REVISITED

```
public class Point{

    double x;  //define getX()
    double y;  //define getY()

    public Point(double x, double y){
        this.x=x;
        this.y=y;
    }
    public double getR(){ return Math.sqrt(x*x+y*y); }

    public double getTheta(){ return Math.atan(y/x); }


    public Point(double r, double theta){
        x= r* Math.cos(theta);
        y= r* Math.sin(theta);
    }

}
```

Why can't I create these 2 constructors?

Is there another way to represent same point?



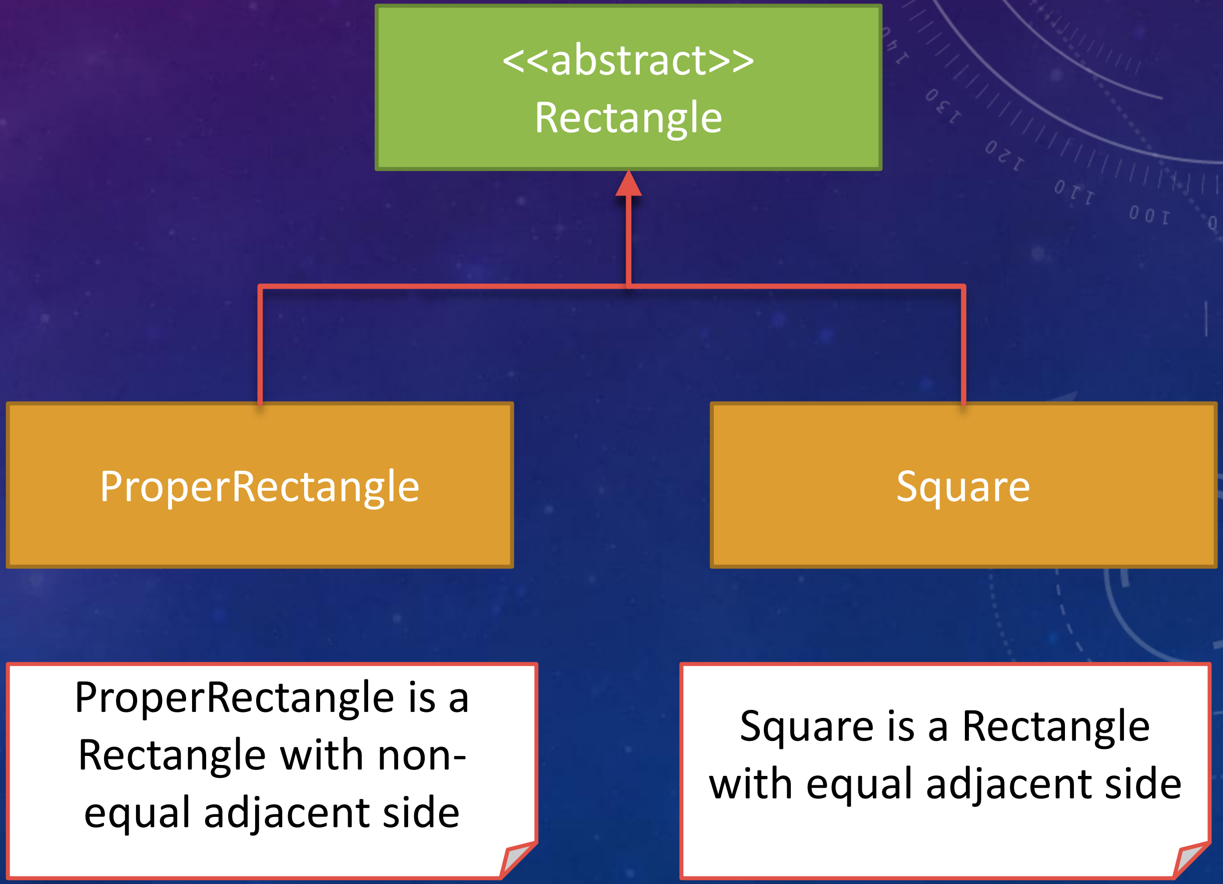
can I create point by supplying r and ϴ?

## CONSTRUCTORS ARE NON-POLYMORPHIC

- POLYMORPHISM IS DIFFERENT OBJECTS BEHAVIOR WITHIN SAME CONTEXT
- POLYMORPHISM IS ACHIEVED BY USING VIRTUAL KEYWORD
- POLYMORPHISM IS WHEN A ABSTRACT REFERENCE REFERS TO A DERIVED OBJECT …
- CONSTRUCTOR WORKS BEFORE OBJECT IS CREATED
  - OBJECT DOESN’T EXIST TO BE Rreferred
  - OBJECT DOESN’T EXIST TO BEHAVE
- NO POLYMORPHISM →
  - NO REPLACEMENT
  - NO LSP
  - NO DIP

## RECTANGLE-SQUARE PROBLEM

```
public abstract class Rectangle{

        public abstract double area();
        public abstract double perimeter();
        public abstract void draw();

}

public class ProperRectangle extends Rectangle{

}

public class Square extends Rectangle{

}
```



<<abstract>>
Rectangle
ProperRectangle
Square
ProperRectangle is a Rectangle with non-equal adjacent side
Square is a Rectangle with equal adjacent side

## LETS CREATE A PROPER RECTANGLE

**Rectangle rectangle = new ProperRectangle(8,8);**

Is this a ProperRectangle or Square?

In Domain it should be ProperRectangle

But constructor of ProperRectangle can't create a Square

Constructors are non-polymorphic

## CONSTRUCTOR BINDS US TO IMPLEMENTATION

```
Rectangle r1 = new ProperRectangle(8,4);

Rectangle r2 = new Square(8);
```

- To create Object You need to know constructor
- To know constructor is to know the **concrete** class
- With the knowledge of **concrete** class there can be No dependency inversion
- When **concrete** class changes source code need to be modified.

## CONSTRUCTOR IS NOT OBJECT ORIENTED

- Constructor Doesn’t belong to Class
  - What is it doing in the class?
- Constructor Doesn’t belong to Object
  - Is it an object oriented design.
- Every Object needs a **creator.**
  - But constructor is not an Object

## CONSTRUCTOR LIMITATIONS

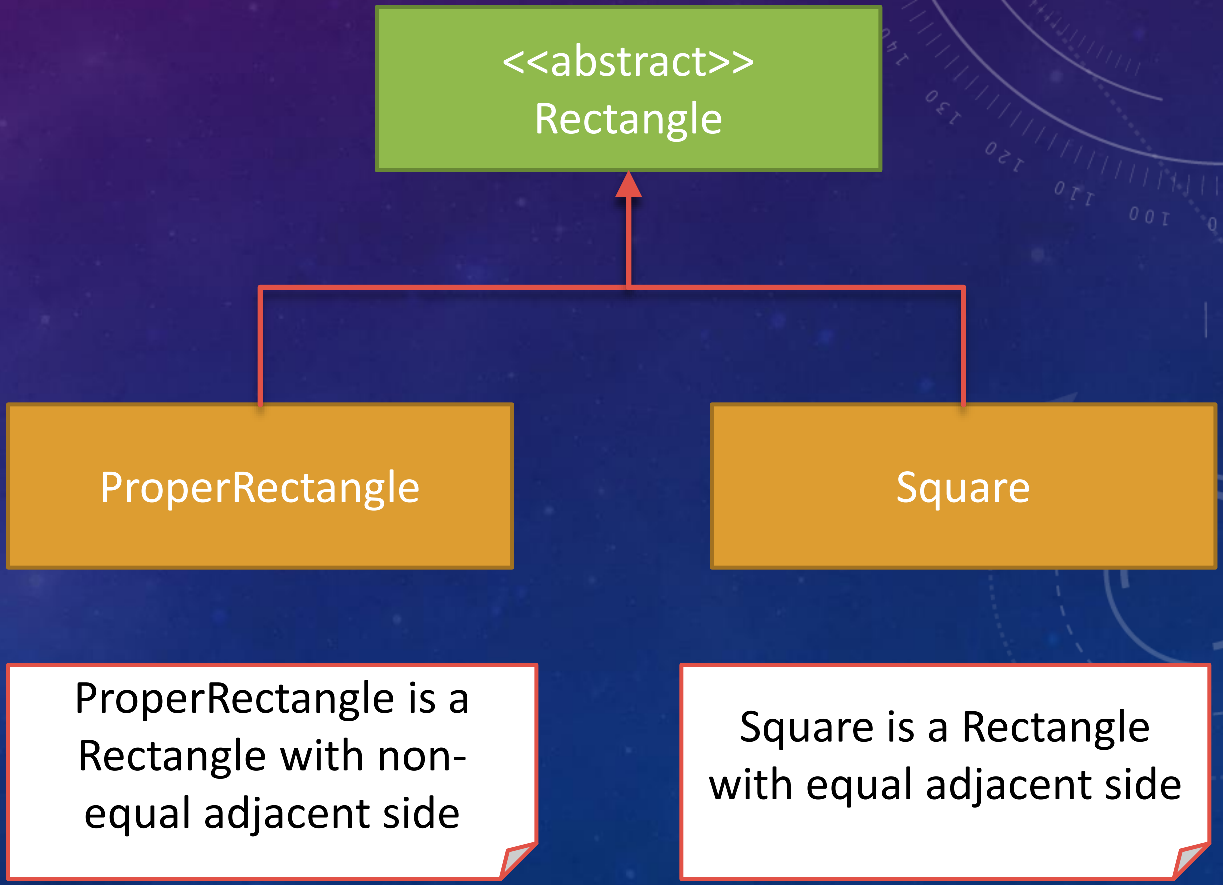
- Constructor Are Not Object Oriented
- Constructors Fail to Reuse Objects
- Constructor Are Non-Polymorphic
- Constructor Name is Meaningless
- Constructor can’t be a Part of contract (interface)
  - Responsibility can’t be ascertained.
- Constructor Violate Design Principles like
  - LSP
  - DIP

## SOLUTION

- AVOID CONSTRUCTOR
  - CONSTRUCTORS CANT BE AVOIDED.
  - THEY CAN BE HIDDEN
- PROVIDE AN ALTERNATIVE CREATOR METHOD TO REPLACE CONSTRUCTOR

## RECTANGLE-SQUARE PROBLEM REVISITED

```
  public abstract class Rectangle{…}

  public class ProperRectangle extends Rectangle{…}

  public class Square extends Rectangle{…}

public class RectangleCreator{

        public static Rectangle Create(double x, double y){
                if(x==y)
                        return new Square(x);
                else
                        return new ProperRectangle( x, y);
        }

}
```



<<abstract>>
Rectangle
ProperRectangle
Square
ProperRectangle is a Rectangle with non-equal adjacent side
Square is a Rectangle with equal adjacent side

## RECTANGLE CREATOR IN ACTION

```
void Client(){

    Rectangle r1 = RectangleCreator.Create(4,3);

    Rectangle r2 = RectangleCreator.Create(4,4);

}
```

Creates a Proper Rectangle

Creates a Square

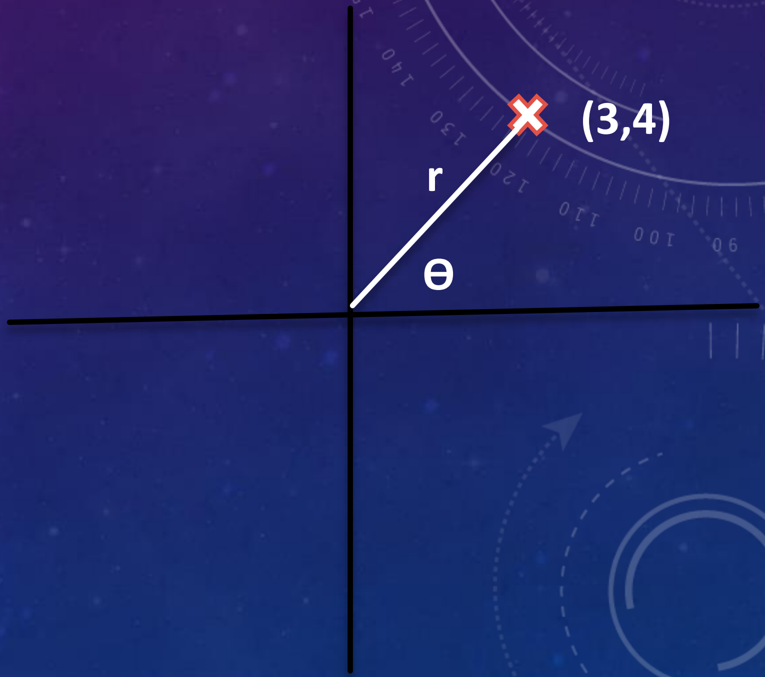
Create creating Object Polymorphically at Runtime???

Client is not aware about concrete classes ProperRectangle and Square

Dependency Inversion

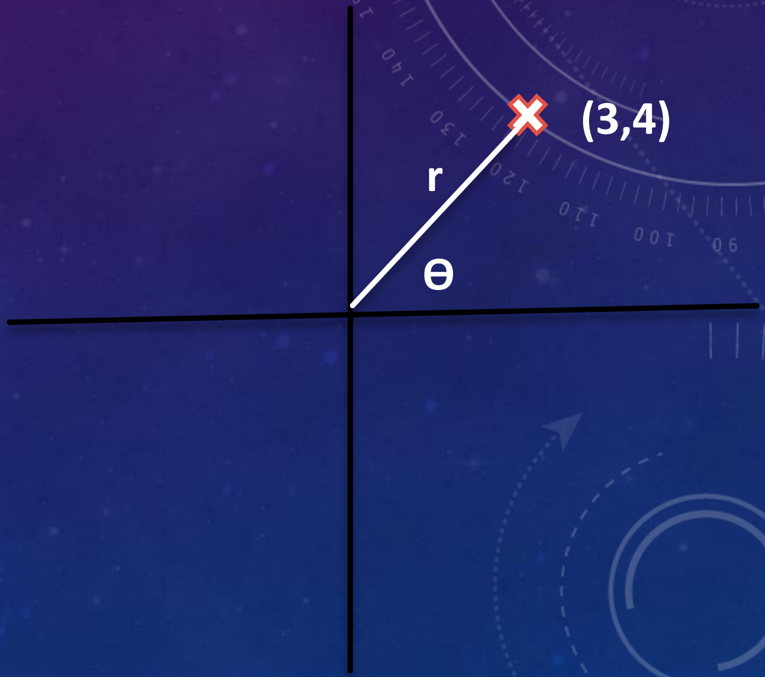
## POINT REVISITED

```
public class Point{

    private Point(double x, double y){
        this.x=x;
        this.y=y;
    }

   public static Point GetCartisian(double x, double y){
       return GetPoint( x , y );
   }
   public static Point GetPolar(double r, double theta){
       return GetPoint( r * Math.Cos(theta), r * Math.Sin(theta);
   }
   …
   private static Point GetPoint(double x, double y){
       …
       Point p=new Point(x,y);
       …
       return p;
   }
}
```



## REUSING EXISTING POINTS

```
public class Point{

    private Point(double x, double y){
        this.x=x;
        this.y=y;
    }


   List<Point> cache = new List<Point>();

   private static Point GetPoint(double x, double y){

       foreach(var p in cache)
           if( p.x==x && p.y==y)
               return p;
       Point p=new Point(x,y);
       cache.Add(p)
       return p;
   }

}
```



## WORKING WITH POINT OBJECTS

```
Point p1 = Point.GetCartisian(3,4);
```

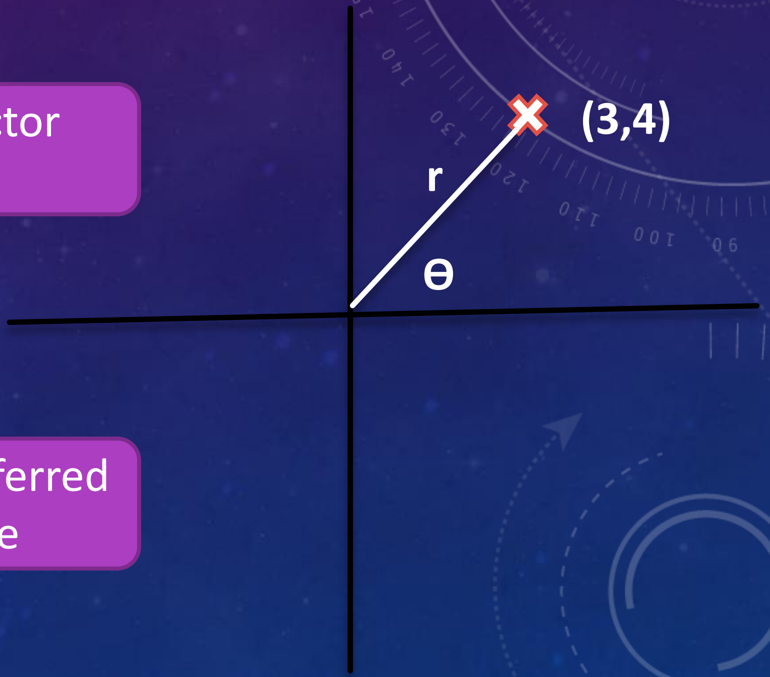
Creates a Point 3,4

```
Point p2= Point.GetPolar(5, 45);
```

Creates a Vector Point

```
Point p3= Point.GetCartisian(3,4);
```

Reuses Point referred by p1 above



# USER MANAGEMENT – REVISITED

Presentation Tier

```
class UserUI{

    public void OnRegister(){
        User user = new User(name,
        User user = UserManager.Create(name,
                                email,
                                password,
                                phone);
    }

    public void OnLogin(){
        User user=new User(email,
        User user=UserManager.Validate(email,
                                password);
    }

}
```

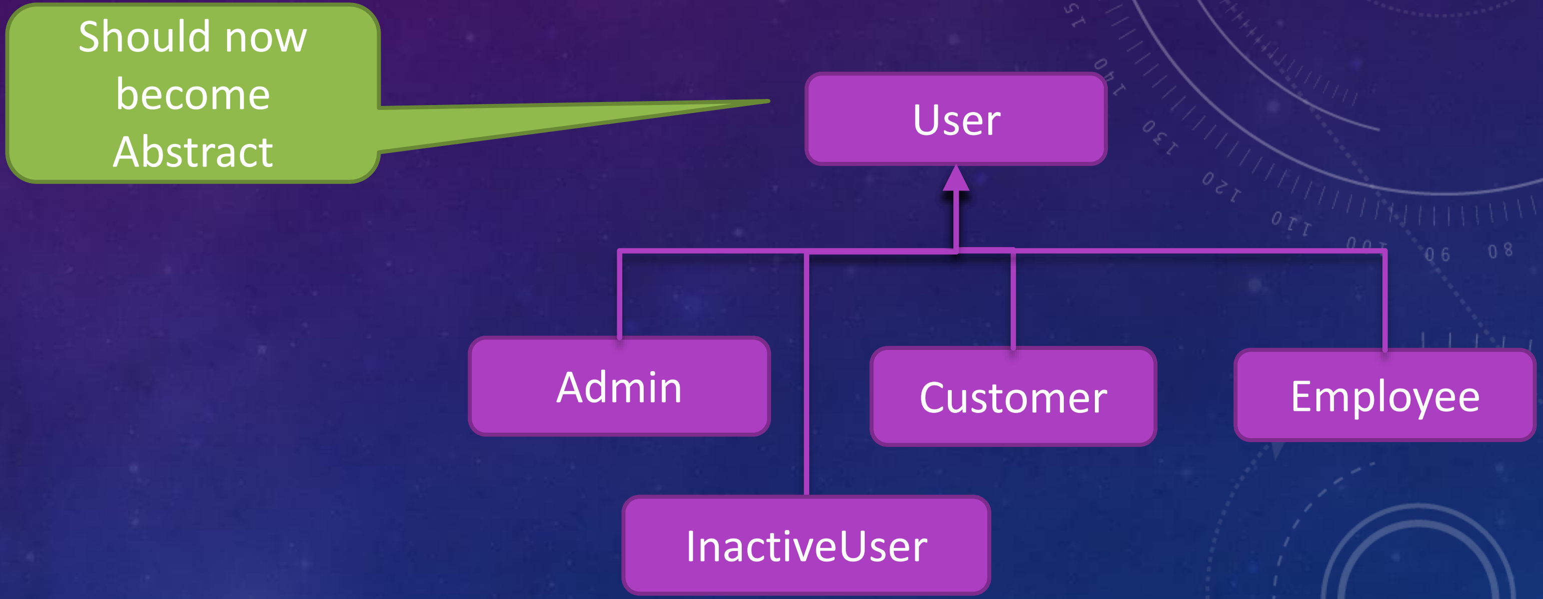
More meaningful names

Business Layer

```
class User{…}
class User{

class UserManager{
    public User(String name,
    public static  User Create(String name,
                        String email,
                        String email,
                        String password,
                        String password,
                        String phone){
                        String phone){

        db.execute("insert into…");
        db.execute("insert into…");
    }
        return new User(…);
    }
    public User(String email,
                        String password){
    public static  User Validate(String email,
                        String password){
        db.execute("select *…");

        db.execute("select *…");
    }
        return new User(…);
    }
}

}
```

# WHAT IF REQUIREMENT CHANGES TOMORROW

- We Need Different Types of Users
  - Admin
  - Customer
  - Employee
- User Will be Registered as InactiveUser.
- No change is Registration and Login UI
- User Type will be change via a new backend screen



# CODE CHANGES (CONSTRUCTOR VERSION)

```
public abstract class User{}
public class Employee : User{ … }
public class Customer: User{…}
public class Admin : User {…}

public class User{
public abstract class InactiveUser : User{

        public InactiveUser(String name, String email, String phone, String password)
                db.execute("insert into …");
        }

        ~~public User(String email, String password){~~
        ~~        db.execute("select * from …");~~
        }
}
```

Induces changes to Presentation Tier

**Presentation Tier**

```
public void RegisterUser(){

        User user = new User( name, email, phone, password);
        User user = new InactiveUser( name, email, phone, password);

}
```

# CODE CHANGES (LOGIN)

```
public abstract class User{}
public class Employee : User{ … }
public class Customer: User{…}
public class Admin : User {…}

public class User{
public abstract class InactiveUser : User{

        public User(String name, String email, Str
                db.execute("insert into …");
        }
```

~~password~~

~~…");~~

Data Access in Presentation Tier!!!

**Presentation Tier**

```
public void LoginUser(){

        User user = new User(email, password);

        var result = db.execute(   elect * from …");
        User user=null;
        if (result.HasNext){
                switch( result['USER)T     ]){
                        case ADMIN:  user        Admin(…); break;
                        case 'EMPLOYEE': us       v Employee(…); break;
                        …
                }
        }


}
```

We Can't Create Object of Abstract class User.
Which object to create depends on a database column 'USER_TYPE'

# CODE CHANGES (MANAGER VERSION)

```
public abstract class User{}
public class Employee : User{ … }
public class Customer: User{…}
public class Admin : User {…}

public abstract class InactiveUser : User{}
public class User{ … }

public class UserManager{

        public static User Create(String name, String email, String phone, String password){
                db.execute("insert into …");
                return new User(…);
                return new InactiveUser(…);
        }

        …
}
```

**Presentation Tier**

```
public void RegisterUser(){

        User user = UserManager.Create( name, email, phone, password);


}
```

Business Layer Has Several Changes

But No Change In Presentation Tier

## CODE CHANGES IN LOGIN (MANAGER VERSION)

Presentation Tier

```
public void LoginUser(){

    User user = UserManager.Validate(email, password);


}
```

```
public class UserManager{

    public static User Validate(String email, String password){
        var result = db.execute( “select * from …”);
        User user=null;
        if (result.HasNext){
            switch( result[‘USER)TYPE’]){
                case ADMIN:  user =new Admin(…); break;
                case ‘EMPLOYEE’: user=new Employee(…); break;
                …
            }
        }
    }
```

Data Access Logic Changes Here

But No Change In Presentation Tier

## CREATOR FUNCTION ADVANTAGE

- **We Replaced Constructor With Our Creator Functions.**
- **Creator Functions are great because they are not special.**
- **We have more control on them**
  1. **We Can Choose The Name -> Create(), Validate(), GetCartisian(), GetPolar() etc**
  2. **We May Create New Object Or Return Existing Object**
  3. **We May Return Different type of Object Depending on the context (Polymorphism???)**
  4. **Client Need Not know the exact type of Object (Dependency Inversion)**

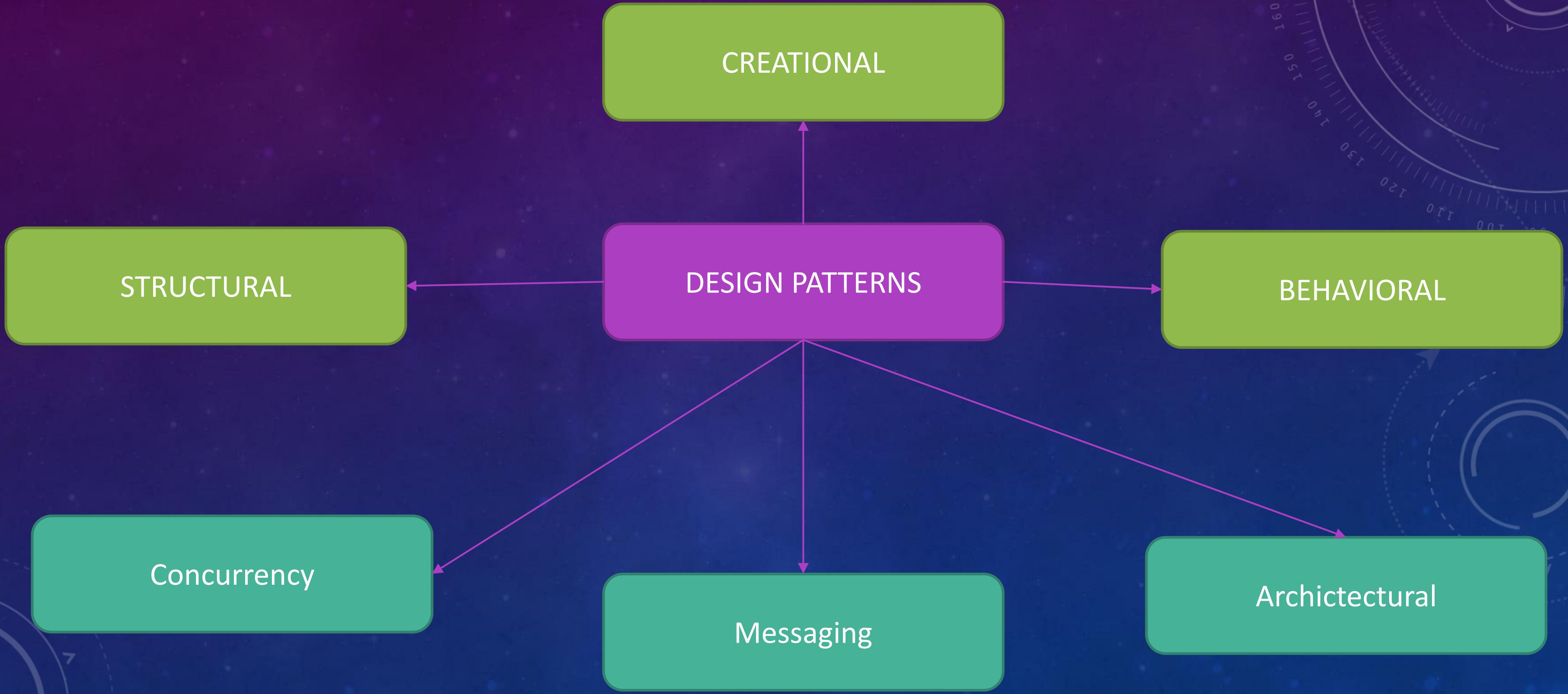
## DESIGN PATTERNS

- DESIGN → GOOD
- PATTERNS → WHAT REPEATS
- DESIGN PATTERN
  - A GOOD PRACTICE REPEATED ACROSS LANGUAGE, DOMAIN, TECHNOLOGY AND GEOGRAPHY
  - OFTEN ADDS FEATURES/ELEMENTS NOT AVAILABLE AS PART OF THE LANGUAGE
  - A HACK TO IMPROVISE  FOR BETTER DESIGN.
  - A HACK TO GET AROUND LANGUAGE LIMITATION/CONSTRAINTS

## WHAT MAKES DESIGN PATTERNS GREAT?

- Based on a Research on How ‘Experts Do The Design’
- Patterns Are Not Created; They Are Discovered
- Patterns Are Like Law
- Each Pattern Has a Name
  - Sometimes more than One Name
- Many Developer Break the Pattern Discovery Rules

## PATTERN CATEGORIES



## CREATIONAL PATTERNS

- How to Create An Object
- How to Avoid Constructor

## STATIC CREATOR – WHY AND WHY NOT?

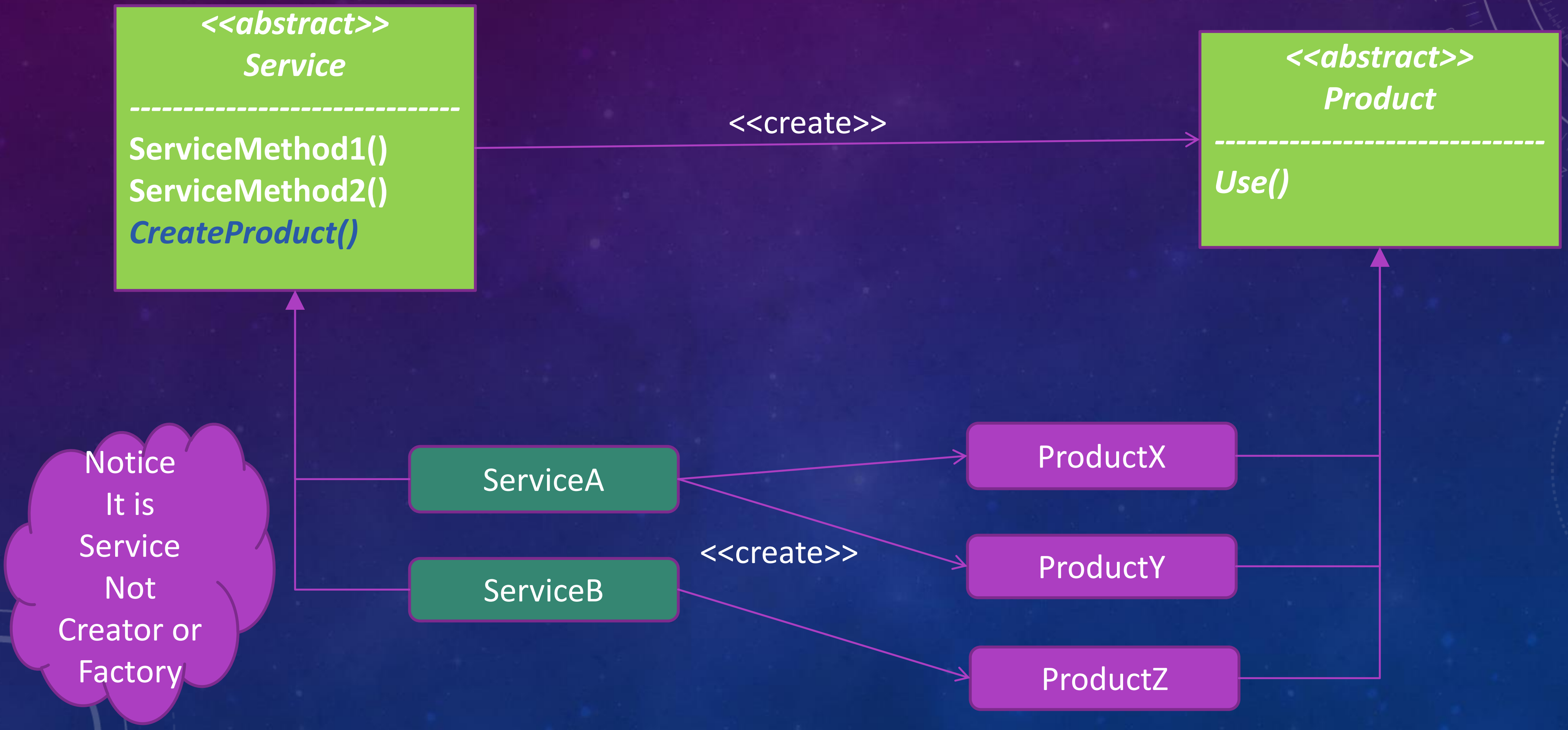
- STATIC CREATOR CANT BE POLYMORPHIC
- STATIC CREATORS DON'T SUPPORT DIP/LSP
- NOTE, STATIC CREATOR STILL SUPPORT
  - MORE MEANINGFUL NAME
  - DIFFERENTIATE NEW AND EXISTING
  - CREATE OBJECT OF DIFFERENT TYPES
  - DECOUPLE IMPLEMENTATION CLASSES FROM CLIENT

## GOF RECOMMENDATION

- AVOID STATIC PREFEFR VIRTUAL
  - RECOMMENDATION WERE C++ BASED.
  - WE SHOULD USE ABSTRACT WHEREEVER POSSIBLE
- VIRTUAL/ABSTRACT ALLOWS
  - LSP
  - DIP
  - POLYMORPHISM

THESE RECOMMENDATION APPLY NOT ONLY TO CREATIONAL PATTERNS BUT MOST OF THE OTHER PATTERNS

# FACTORY METHOD PATTERNS



## FACTORY METHOD PATTERN

- DEFINES ABSTRACTION FOR CREATING A PRODUCT
  - HIDES CONSTRUCTOR
  - HIDES CONCRETE PRODUCT DETAILS FROM THE CLIENT
- TWO PARALLEL HIERARCHY
  - PRODUCTS
  - SERVICE THAT CREATES THE PRODUCT
- FOR A NEW PRODUCT
  - A NEW SERVICE OBJECT MAY BE CREATED

Also know as Virtual constructor Pattern

## FACTORY METHOD (NOT FACTORY CLASS)

- NOTICE THE CREATOR METHOD IS PRESENT IN A SERVICE CLASS WHY?
- FACTORY METHOD MAY BE PRESENT IN A NON-FACTORY, NON-CREATOR CLASS.
- CREATING THE OBJECT MAY NOT BE THE KEY RESPONSIBILITY
- IS IT VIOLATING SRP?
  - CREATION MAY BE A SUB-RESPONSIBILITY

## CASE STUDY

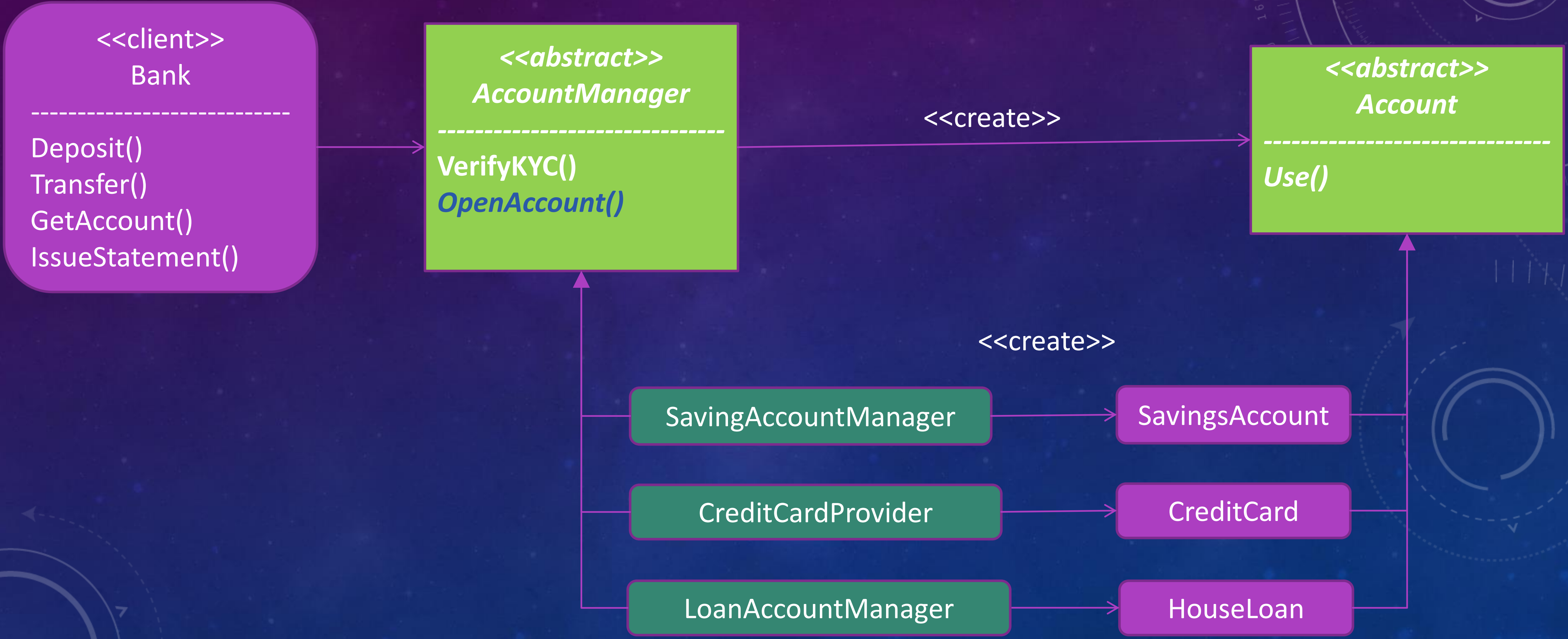
- LINKED LIST
  - IT'S A CONTAINER, NOT A FACTORY
  - KEY RESPONSIBILITY IS TO STORE AND MANAGE OBJECTS
  - INCLUDES AN ITERATOR TO ALLOW NAVIGATION THROUGH THE LIST.
  - GetEnumerator IS A FACTORY METHOD
    - DIFFERENT COLLECTION CREATE DIFFERENT ENUMERATOR OBJECTS

C++ → SmartPointer
Java → Iterator

## CASE STUDY - BANK

  - NOT A FACTORY
- CORE RESPONSIBILITY
  - MANAGE ACCOUNTS
  - TRANSFER FUNDS
  - FINANCIAL TRADE
- WHERE FACTORY METHOD COMES IN
  - OPENS ACCOUNT

## FACTORY METHOD BANK



## USING BANK

```
void main(){
        Bank icici= ...
        int account= icici.OpenAccount(...); //creates BankAccount but returns Account Number
        icici.Deposit( account, 20000);

}
```

Factory Method

NOT A Factory Object

Is static factory an absolute No???

## BEYOND FINAL THOUGHTS…

```
void main(){

    RBI rbi = Govt.getRBI();

    Bank icici = rbi . getBank("ICICI");

    int account = icici.OpenAccount(…); //creates BankAccount but returns Account Number

    icici.Deposit( account, 20000);

}
```

At somepoint its either constructor or static factory

But How to create rbi? where would this end?

How to create 'icici' object?

## ABSTRACT FACTORY A.K.A TOOLKIT

## UI A CASE STUDY

- CLIENT NEEDS A NEW UI SYSTEM
- UI SHOULD LOOK AND BEHAVE AS STEEL
  - SHINE LINE STEEL
  - CLICK SHOULD SOUND LIKE STEEL
  - SHOULD BE HEAVY

## UI - POC

changes
6 → POC
1000→ Real World

```
void POC(){

    SteelForm     form     = new SteelForm();
    SteelButton   button   = new SteelButton();
    SteelTextBox textBox = new SteelTextBox();

    form.Add(button);
    form.Add(textBox);

    form.Draw();
}
```

What happens we client wants to change from Steel to Rubber?

How many places it changes? what changes?

In real world application there are 500 objects of 20 different types (check box, progress bar etc)

changes
3 →POC
500→ Real World

## UI – DEPENDENCY INVERSION

```
void POC(){

        SteelForm form= new SteelForm();
        SteelButton button = new SteelButton();
        SteelTextBox textBox= new SteelTextBox();

        AbstractForm form= new SteelForm();
        AbstractButton button = new SteelButton();
        AbstractTextBox textBox= new SteelTextBox();



    form.Add(button);
    form.Add(textBox);

    form.Draw();
}
```

Now How many places it needs to change why?

In real world application there are 500 objects of 20 different types (check box, progress bar etc)

changes
3 →POC
20→ Real World

## UI – FACTORY METHOD PATTERN

```
void POC(){

    AbstractFormFactory     ff     = new SteelFormFactory();
    AbstractButtonFactory   bf   = new SteelButtonFactory();
    AbstractTextBoxFactory tf = new SteelTextBoxFactory();

    AbstractForm      form      = ff.CreateForm();    //~~new SteelForm();~~
    AbstractButton    button    = bf.CreateButton(); //~~new SteelButton();~~
    AbstractTextBox textBox = tf.CreateTextBox(); //~~new SteelTextBox();~~

    form.Add(button);
    form.Add(textBox);

    form.Draw();
}
```

How many places it changes?
what changes?

In real world application there are 500 objects of 20 different types (check box, progress bar etc)

## UI – ABSTRACT FACTORY

changes
1 →POC
1→ Real World

```
void POC(){

    AbstractUIFactory ui= new SteelUIFactory();
    AbstractFormFactory     ff     = new SteelFormFactory();
    AbstractButtonFactory   bf   = new SteelButtonFactory();
    AbstractTextBoxFactory tf = new SteelTextBoxFactory();

    AbstractForm      form      =  ff.CreateForm();
    AbstractButton    button    = bf.CreateButton();
    AbstractTextBox textBox = tf.CreateTextBox();


    form.Add(button);
    form.Add(textBox);

    form.Draw();
}
```

How many places it changes?
what changes?

In real world application there are 500 objects of 20 different types (check box, progress bar etc)

## UI – ASSIGNMENT

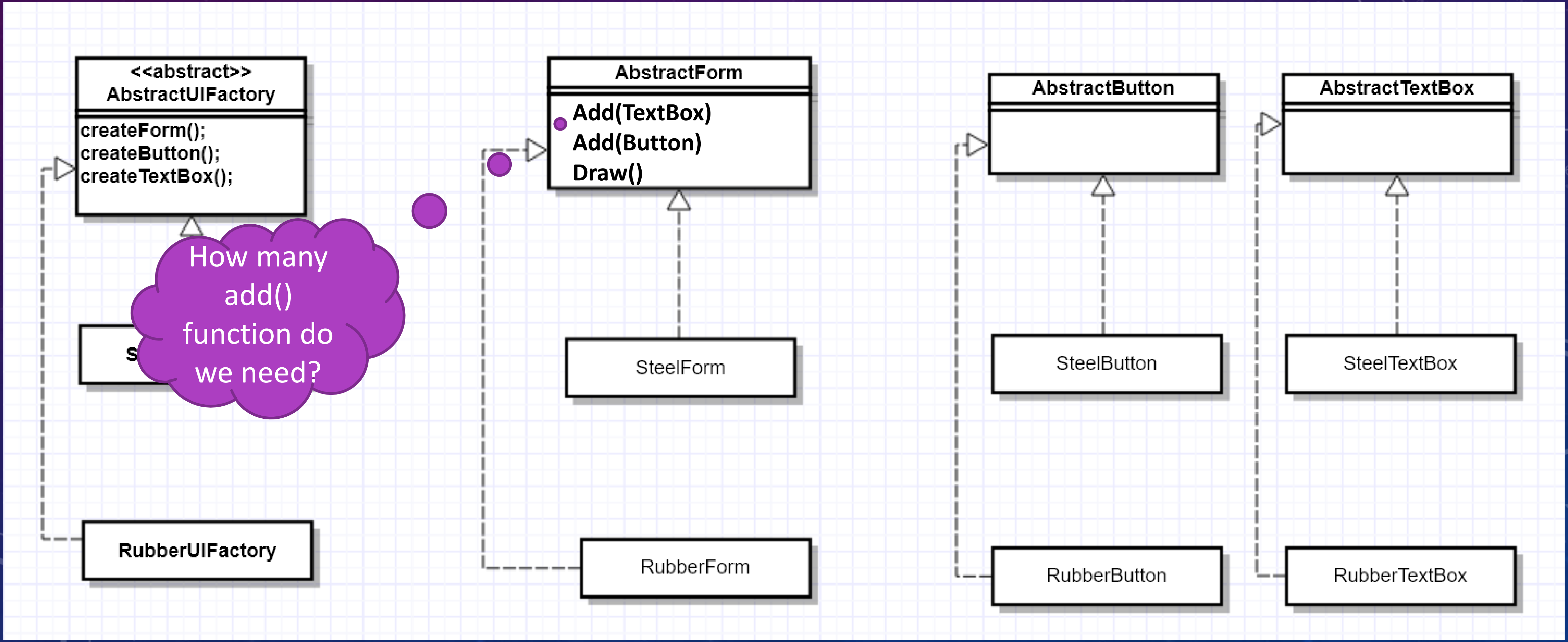
```
void POC(){

    AbstractUIFactory ui= new SteelUIFactory();
    AbstractForm      form     =  ui.CreateForm();
    AbstractButton    button   = ui.CreateButton();
    AbstractTextBox textBox =  ui.CreateTextBox();



    form.Add(button);
    form.Add(textBox);

    form.Draw();
}
```

**expected output:**

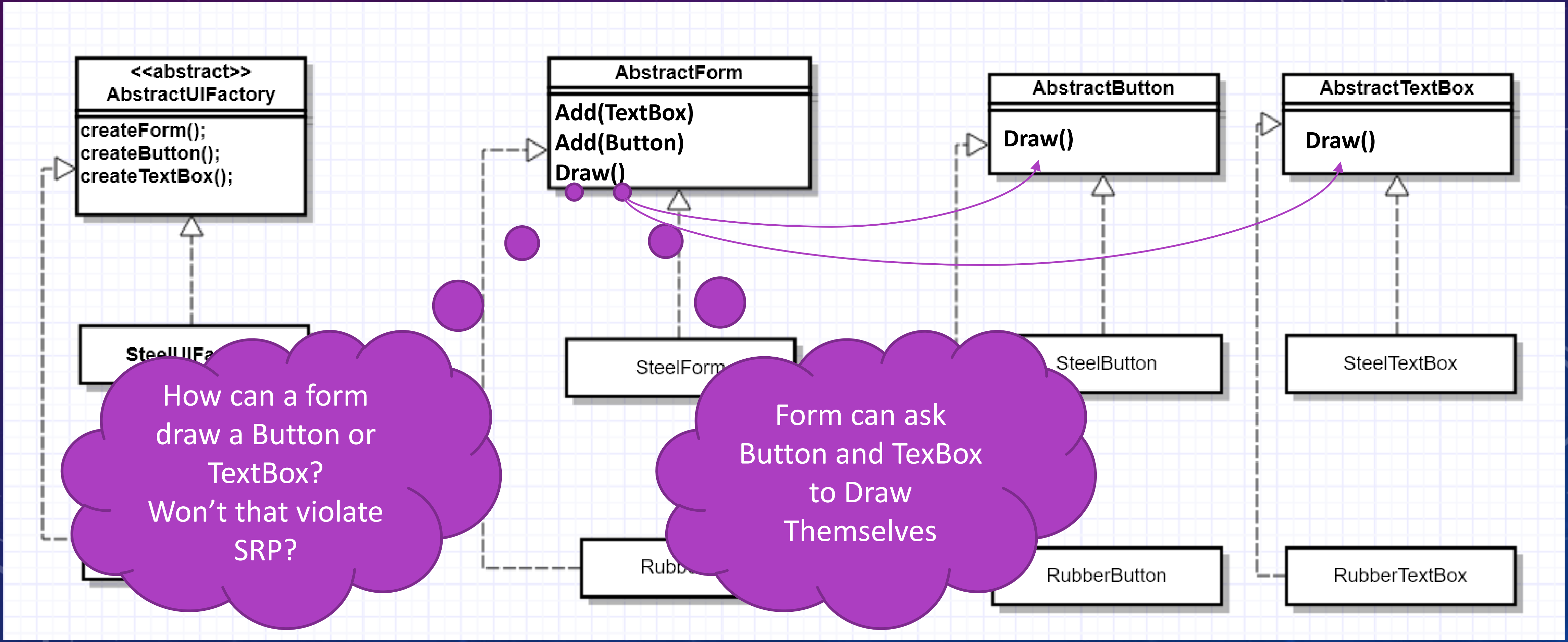
```
Steel Form Drawn
        SteelButton Drawn
        SteelTextBox Drawn
```

1. No Change in POC
2. Define Necessary Abstractions
3. Create at least two sets (steel,rubber)
4. At least 3 objects (form,button,textbox)
5. Real world will have 20 types
6. Implement Add
    1. To Add TextBox and Button
7. Implement Form Draw to
    1. To Draw Form, TextBox, Button etc
8. Create Proper Packaging of Components

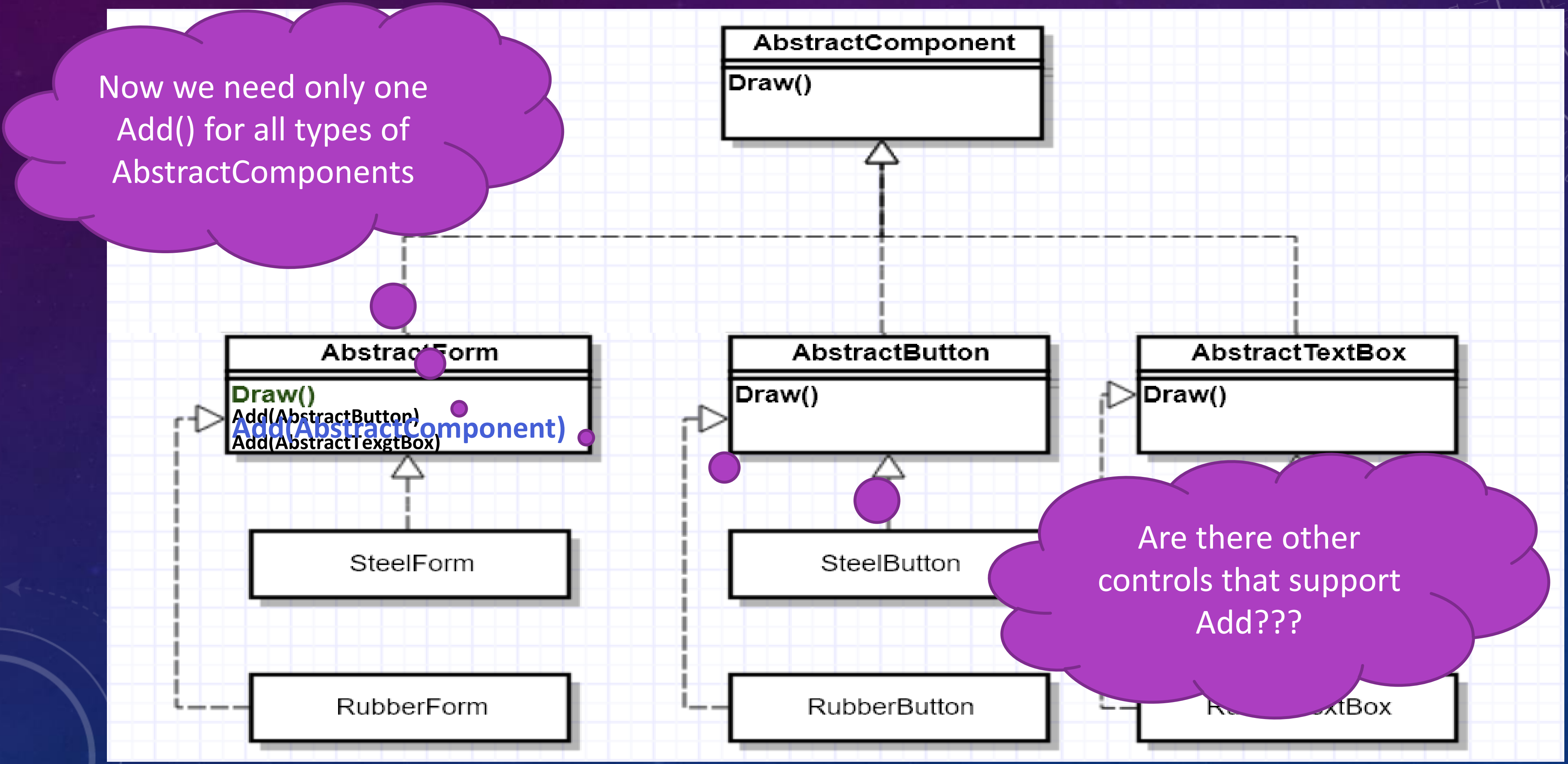
## UI – CLASSES (VERSION 1)



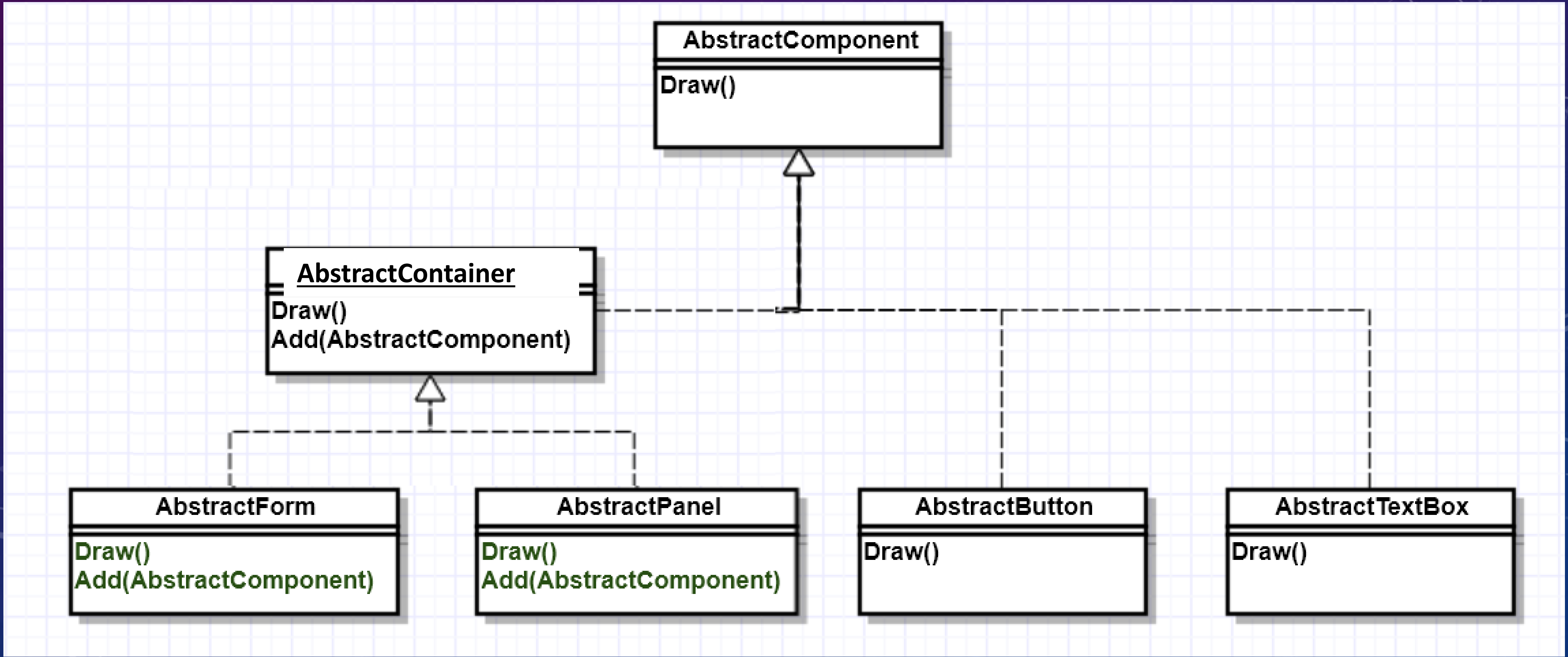
# UI – CLASSES (VERSION 1)



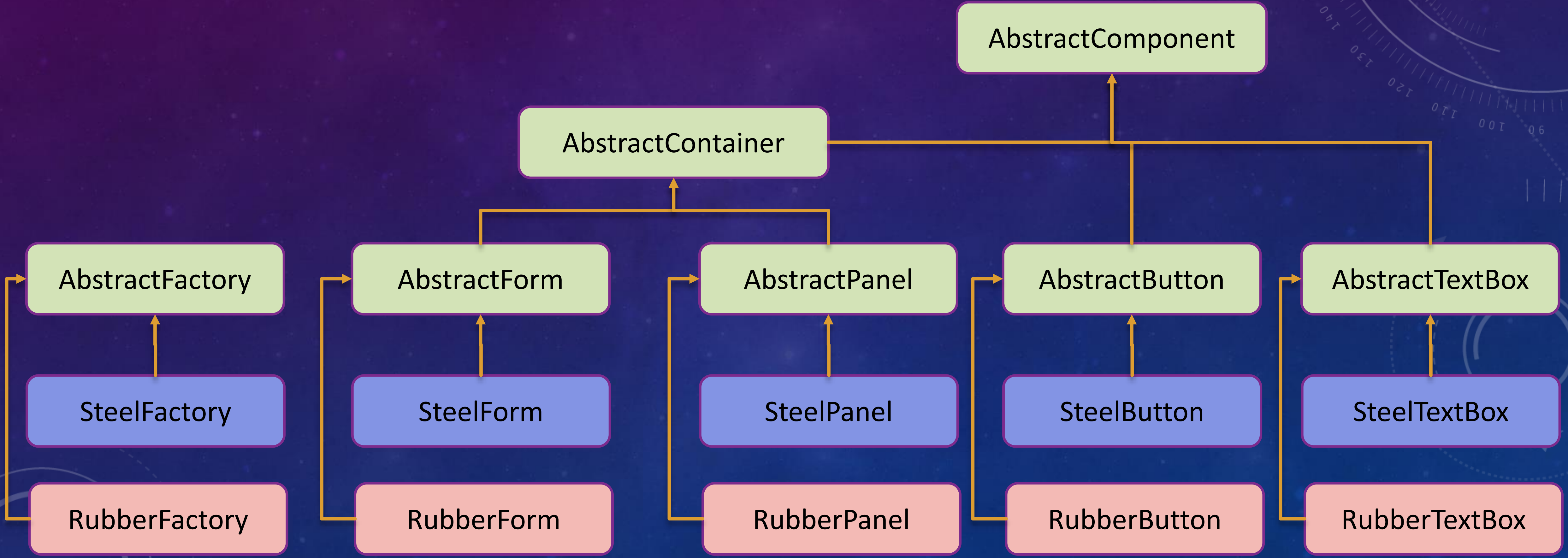
# ENCAPSULATE WHATEVER REPEATS (HERE DRAW)



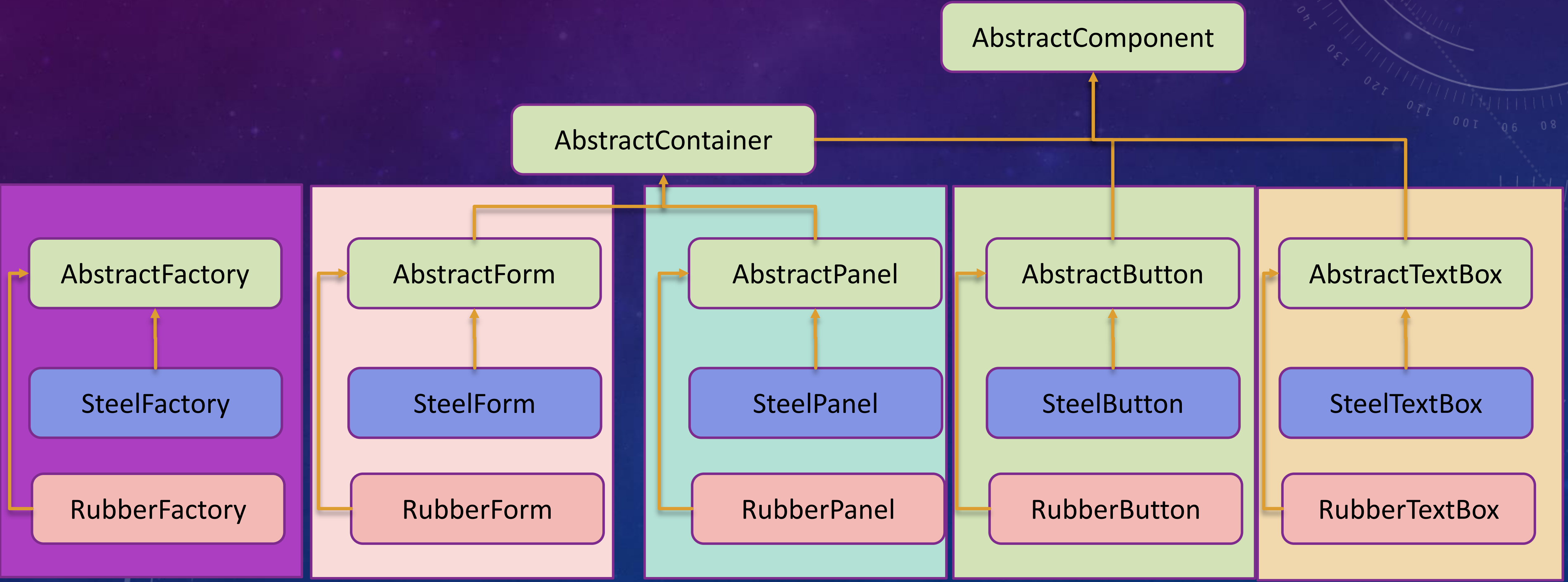
## ANOTHER COMPONENT THAT SUPPORTS ADD



## HOW DO WE PACKAGE THE COMPONENTS



# OPTION 1

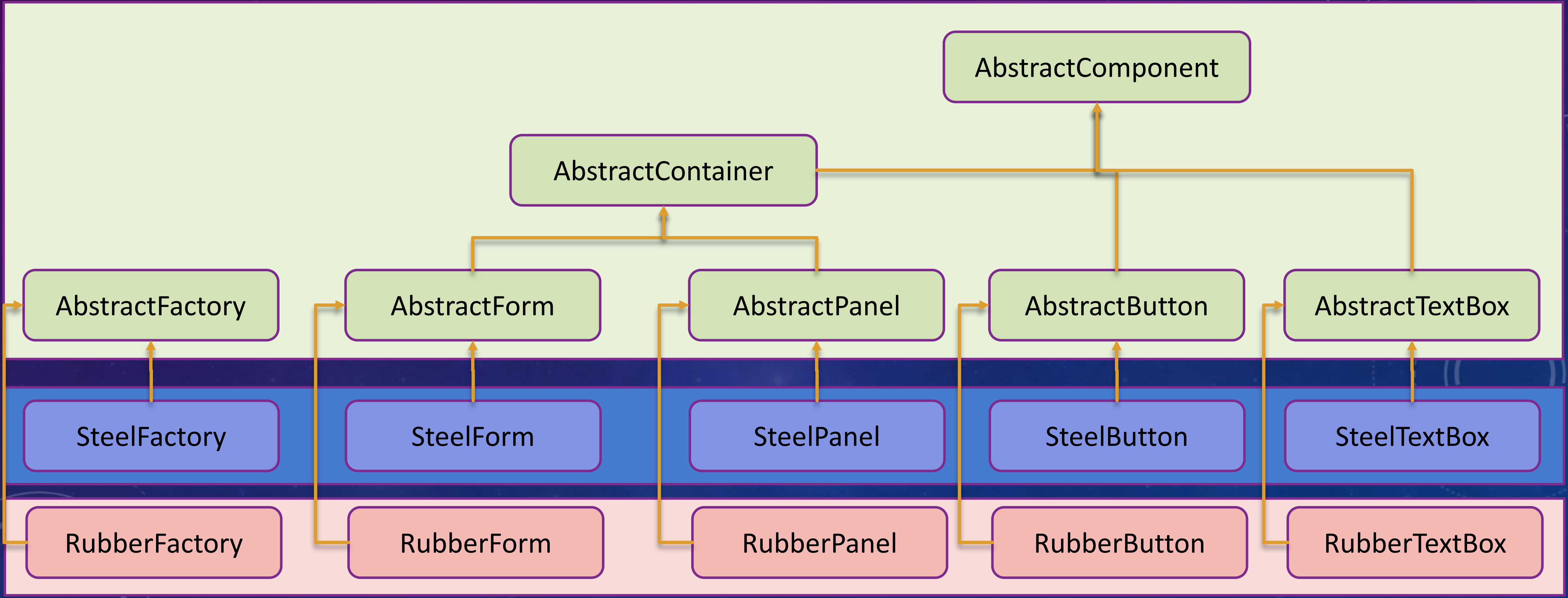

AbstractComponent
AbstractContainer
AbstractFactory
SteelFactory
RubberFactory
AbstractForm
SteelForm
RubberForm
AbstractPanel
SteelPanel
RubberPanel
AbstractButton
SteelButton
RubberButton
AbstractTextBox
SteelTextBox
RubberTextBox

## DESIGN IMPLICATION

Questions?

1. How Many Package/DLL/JAR is required by the client?
2. If Client demands Glass Components, How many components need to change?

# OPTION 2


AbstractComponent
AbstractContainer
AbstractFactory
AbstractForm
AbstractPanel
AbstractButton
AbstractTextBox
SteelFactory
SteelForm
SteelPanel
SteelButton
SteelTextBox
RubberFactory
RubberForm
RubberPanel
RubberButton
RubberTextBox

## DESIGN IMPLICATION

Questions?

1. How Many Package/DLL/JAR is required by the client?
2. If Client demands Glass Components, How many components need to change?

## SINGLETON

- WHAT
  - THERE SHOULD BE A SINGLE OBJECT FOR A PARTICULAR TPE
- WHY
  1. CONCEPTUAL SINGLETON
     - OBJECT IS SINGLETON IN THE DOMAIN
       - EG. CHAIRMAN OBJECT, RBI, PRESIDENT ETC
  2. CONVINIENT SINGLETON
     - NOT CONCPTUAL SINGLETON
     - HEAVY WEIGHT RESOURCE
     - PERFORMANCE
  3. ZERO WEIGHT OBJECT
     - ALL INSTANCE ARE SAME
     - NO POINT IN CREATING MULTIPLE OBJECTS

## SINGLETON VERSION 1

```
class Singleton
{
    private Singleton(){}

    private static Singleton instance=null;

    public static Singleton GetInstance(){

        if(instance==null)
        {
            instance=new Singleton();
        }
        return instance;
    }

}
```

1. CREATE PRIVATE CONSTRUCTOR TO AVOID UNWANTED OBJECTS
2. STATIC CREATOR TO GET THE ONLY OBJECT
3. STATIC REFERENCE TO THE ONLY OBJECT
4. GetInstance CREATES INSTANCE IF NOT ALREADY CREATED

## SINGLETON VERSION PROBLEM

```
class Singleton
{
      private Singleton(){}

      private static Singleton instance=null;

      public static Singleton GetInstance(){

            if(instance==null)
            {
                  instance=new Singleton();
            }

            return instance;
      }

}
```

If Multiple Threads Reach this point and instance==null.

The all end up creating  new instances breaking singleton.

**Problem: Thread Unsafe**
**Severity: Severe**
**Frequency:  Rare**

## SINGLETON VERSION 2 THREAD SAFE

```
class Singleton
{
      private Singleton(){}

      private static Singleton instance=null;
       private static Object lock=new Object();
      public static Singleton GetInstance(){

            lock(object){

              if(instance==null)
              {
                  instance=new Singleton();
              }
            }

            return instance;
      }
```

Even If Multiple Threads Reach the lock

only one can enter it

**Problem: Performance**
**Severity: Severe**
**Frequency: High**

## SINGLETON VERSION 3 DOUBLE CHECKED LOCKS

```
class Singleton
{
    private Singleton(){}

    private static Singleton instance=null;
     private static Object lock=new Object();
    public static Singleton GetInstance(){

      if(instance==null){
        lock(object){

          if(instance==null)
          {
              instance=new Singleton();
          }
        }
      }
        return instance;
    }
```

Avoids un-necessary locking if instance is already created.

Effective as long as instance ==null

**Double Checked Lock is considered as a Design Pattern (Non-GoF)**

# DOUBLE CHECKED LOCKS

- WHAT?
  - A CONCURRENCY PATTERN TO AVOID UNWANTED LOCK
  - NOT A GOF PATTERN
  - AVOIDS UNWANTED LOCKS
- WHY?
  - LOCKING IS AN EXPENSIVE PROCESS
  - IT REQUIRES OS RESOURCE MANAGEMENT
- HOW
  - CHECK IF LOCK IS REALLY REQUIREDNOT A GOF PATTERN
  - TAKE THE LOCK
  - PERFORM YOUR BUSINESS OPERATION

## SINGLETON VERSION 4

```
class Singleton
{
    private Singleton(){}

    static Singleton instance= new Singleton();

    public static Singleton GetInstance(){

        return instance;
    }

}
```

**GOOD: STATIC INITALIZATION IS THREAD SAFE BY DESIGN**

**BAD: EARLY INITIALIZED**

## LAZY INITIALIZATION VS EAGER INITIALIZATION

- HEAVY WEIGHT OBJECTS NEED SPECIAL CREATION STRATEGY
- YOU MAY NEED IT LATE
- INITALIZING EARLY MAY WASTE RESOURCE
- INITIALIZE JUST BEFORE FIRST USE

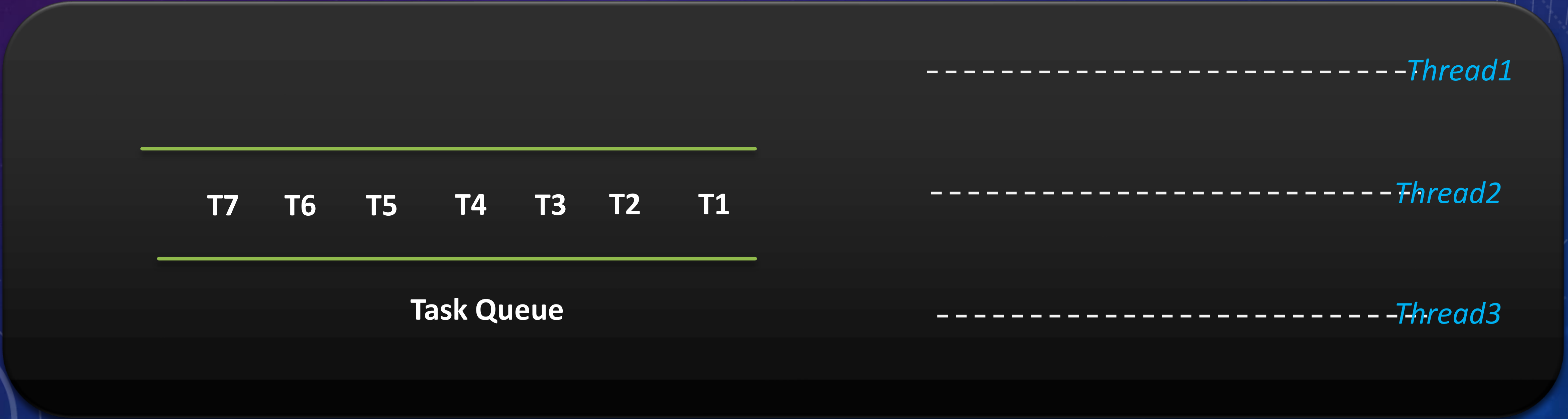
- EAGER INITALIZATION ISNT BAD IF
  - OBJECT IS NOT HEAVY WEIGHT.

## THREADPOOL USE CASE

- What is a Thread Pool
  - Thread Pool Allows a limited number of Threads to process a large number of tasks
- Why Thread Pool?
  - Thread Life cycle is managed by OS
    - Creation and Deletion Takes Time
  - Each Thread Requires a separate memory space
    - Typically 2 MB
  - More thread means more time wasted in Thread Switching

## HOW THREADPOOL WORKS

- Thread Pool Maintains a limited number of Threads
- It Maintains a Task Queue
- Tasks are Queued
- Each Thread Takes a Task from the Queue and Executes It
- Once a Thread Finishes a Task, It takes the next Task from the Queue
- Once Queue is Empty the threads wait for next Task to Arrive



## THREAD POOL EXAMPLE

```
public class ThreadPool
{
        List<Thread> threads;
        List<Task> tasks;

        public void AddTask(Task task) {tasks.Add(task); }

        void Init()
        {
                // CREATE 10 THREADS. (say 10)
        }

        void PoolTask()
        {
                //Threads Excutes Task from the Pool
        }


        public int GetPendingTaskCount(){…}
        public int GetCompletedTaskCount(){…}


}
```

```
void client(){

        ThreadPool pool1 = new ThreadPool();


        for( File f  :  FileSystem.GetFiles()) //50000 files
                pool1. AddTask( … ) ;

}
```

# THREAD POOL EXAMPLE

```
public class ThreadPool
{
        List<Thread> threads;
        List<Task> tasks;

        public void AddTask(Task task) {tasks.Add(task); }

        void Init()
        {
                // CREATE 10 THREADS. (say 10)
        }

        void PoolTask()
        {
                //Threads Excutes Task from the Pool
        }


        public int GetPendingTaskCount(){…}
        public int GetCompletedTaskCount(){…}


}
```

```
void client(){

        ThreadPool pool1 = new ThreadPool();

        ThreadPool pool2 = new ThreadPool();

        for( File f  :  FileSystem.GetFiles()) //50000 files
                pool1. AddTask( … ) ;

}
```

But there is a problem!!!

Now pool1 is busy processing
50000 jobs
pool2 has 10 idle threads

## THREAD POOL SINGLETON

```
public class ThreadPool
{
        List<Thread> threads;
        List<Task> tasks;
        public void AddTask(Task task) {tasks.Add(task); }
        void Init(){
                // CREATE 10 THREADS. (say 10)
        }
        void PoolTask(){
                //Threads Excutes Task from the Pool
        }
        public int GetPendingTaskCount(){…}
        public int GetCompletedTaskCount(){…}

        private ThreadPool();
        private static ThreadPool instance;

        public static ThreadPool getInstance(){
                //version 3 singleton
        }

}
```

```
void client(){

        ThreadPool pool1 = new ThreadPool();

        ThreadPool pool2 = new ThreadPool();

        for( File f  :  FileSystem.GetFiles()) //50000 files
                pool1. AddTask( … ) ;
                ThreadPool.getInstance().AddTask( … )
}
```

Now There will be no two instance of ThreadPool

# SINGLETON ALTERNATIVE

```
public class ThreadPool
{
    List<Thread> threads;
    List<Tasks> tasks;
    public void AddTask(Task task){ tasks.Add(task); }
    void Init(){
        // CREATE 10 THREADS. (say 10)
    }
    void PoolTask(){
        //Threads Excutes Task from the Pool
    }
    public int GetPendingTaskCount(){…}
    public int GetCompletedTaskCount(){…}
  private ThreadPool();
  private static ThreadPool instance;

  public static ThreadPool getInstance(){
        //version 3 singleton
  }
}
```

```
    static List<Thread> threads;
    static List<Tasks> tasks;
    public static void AddTask(Task task){ tasks.Add(task); }
    static void Init(){
    static void PoolTask(){
    public static int GetPendingTaskCount(){…}
    public static int GetCompletedTaskCount(){…}
```

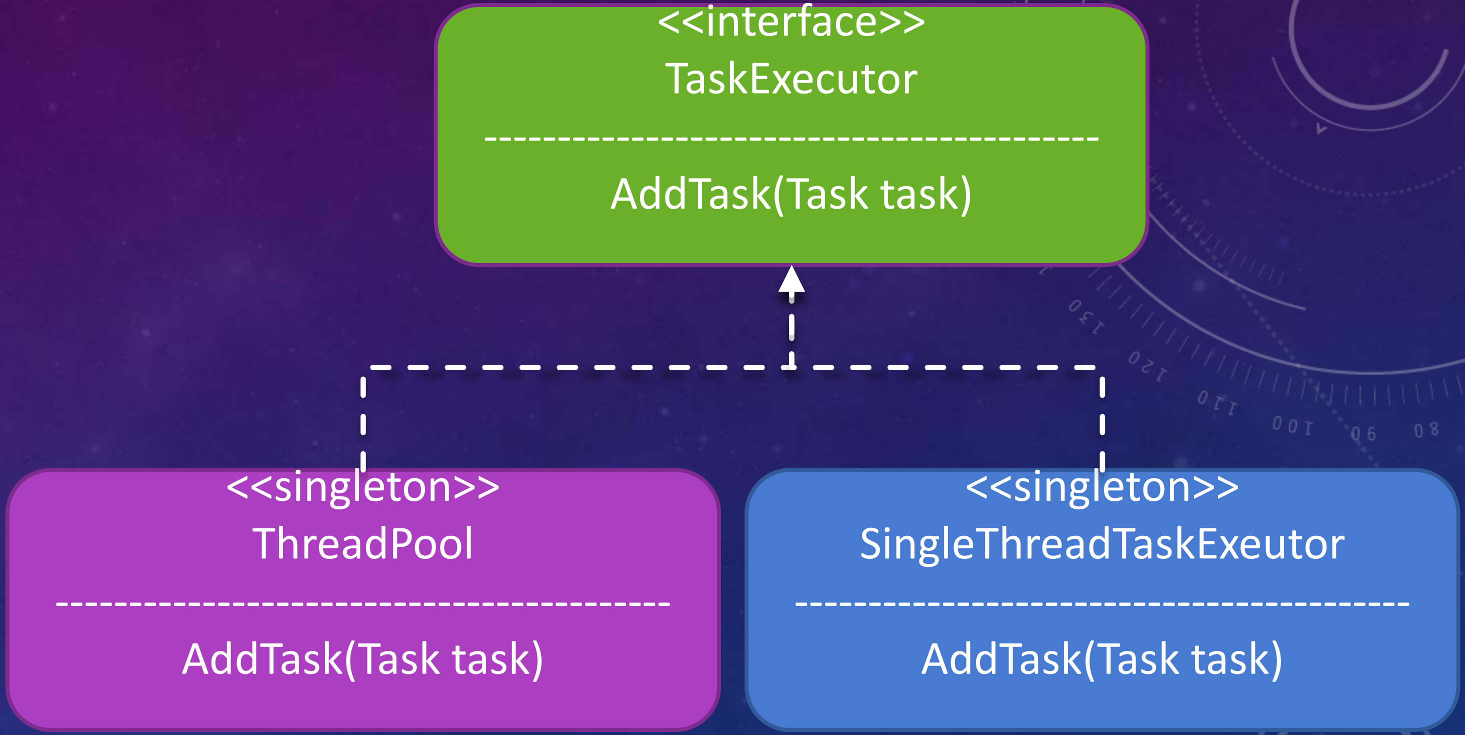
```
void client(){

    for( File f  :  FileSystem.GetFiles()) //50000 files
        ThreadPool.getInstance().AddTask( … )
}
```

```
        ThreadPool.AddTask( … )
```

If the key benefit of singleton is that only one set of fields for the class will be created why can't we do the same thing using static?

## SINGLETON VS STATIC

- Static → No Object
- Singleton → 1 Object
- Singleton (ThreadPool) can implement interface
- It support LSP, DIP, OCP
- Can Replace (or Replaced with other implementation)
- **Static can't implement**
  - **Interface**
  - **DIP, LSP, OCP**



## SINGLETON USES STATIC

```
public class ThreadPool
{
	private ThreadPool();
	private static ThreadPool instance;

	public static ThreadPool getInstance(){
		//version 3 singleton
	}

}
```

static may be bad…

… But Singleton can't be created without using it

static is bad when it implies no object

Here static represents the only object that exists. Its not trying to avoid the object.

## SINGLETON CONCERNS

No matter what you do, Singleton can be broken!!! There are ways to break it

Singleton is about clean coding at a guard against un-intentional change. Its not a security model against hacker.

Singleton Makes testing harder

Still easier than static. If your singleton implements some interface testing may not be as difficult.

## WHO CREATES A CAR??? A FACTORY

## AND WHO CREATES THEM?

## A LIONESS (NOT A FACTORY)

## PROTOTYPE PATTERN

- SOMETIMES WE NEED AN OBJECT OF EXACTLY THE SAME TYPE AS THE ONE WE HAVE
- THIS IS SIMILAR TO REPRODUCTION IN LIVING BEINGS
- OBJECT OF TYPE LION PRODUCES (CREATES) ANOTHER OF ITS TYPE
- WE CAN CREATE NEW OBJECT FROM ANOTHER OBJECT.

## PROTOTYPE VS FACTORY

- IN FACTORY PATTERN OBJECT OF TYPE X IS RESPONSIBLE FOR CREATING OBJECT OF TYPE Y
  - A PENCIL CREATES A CIRCLE
  - A BANK CREATES BANK ACCOUNT
  - A CAR FACTORY CREATES CAR
- IN PROTOTYPE AN OBJECT OF TYPE X CREATES ANOTHER OBJECT OF TYPE X.
  - LION GIVES BIRTH TO LION
  - HUMANS GIVE BIRTH TO ANOTHER HUMAN
- WE KNOW A EGG WILL PRODUCE A BABY
  - IF ITS A CROW’S EGG IT WILL PRODUCE A CROW
  - IF ITS A PARROTS EGG IT WILL PRODUCE A PARROT

## PROTOTYPE VS CLONING

- CLONING PRODUCES ANOTHER EXACT DUPLICATE OF THE CURRENT OBJECT
- CLONED OBJECT HAS
    - SAME PROPERTY VALUE AS ORIGINAL
    - ITS LIKE A SHALLOW COPY
- PROTOTYPE HAS THE SAME TYPE
    - PROTOTYPE MAY NOT DUPLICATE THE STATE VALUE
    - THE NEW OBJECT IS JUST SAME AS OTHER OBJECT
    - MAY DUPLICATE SOME OF THE STATE