SOLID Design Principles & Implementations

By:
Noha Ahmed Thabet

Agenda

- Dependency Inversion
- Inversion of Control
- Dependency Injection
- Building Our Own IoC Container

Dependency Inversion Principle

- DIP Definition
- The Problem
- Case Study
- Refactoring to Apply DIP
- Related Fundamentals

Short Vocabulary Lesson

- **Dependency Inversion Principle (DIP)**
  - Principle used in architecting software
- **Inversion of Control (IoC)**
  - Specific pattern used to invert interfaces, flow and dependencies
- **Dependency Injection (DI)**
  - Implementation of IoC to invert dependencies
- **Inversion of Control Container**
  - Framework to do dependency injection

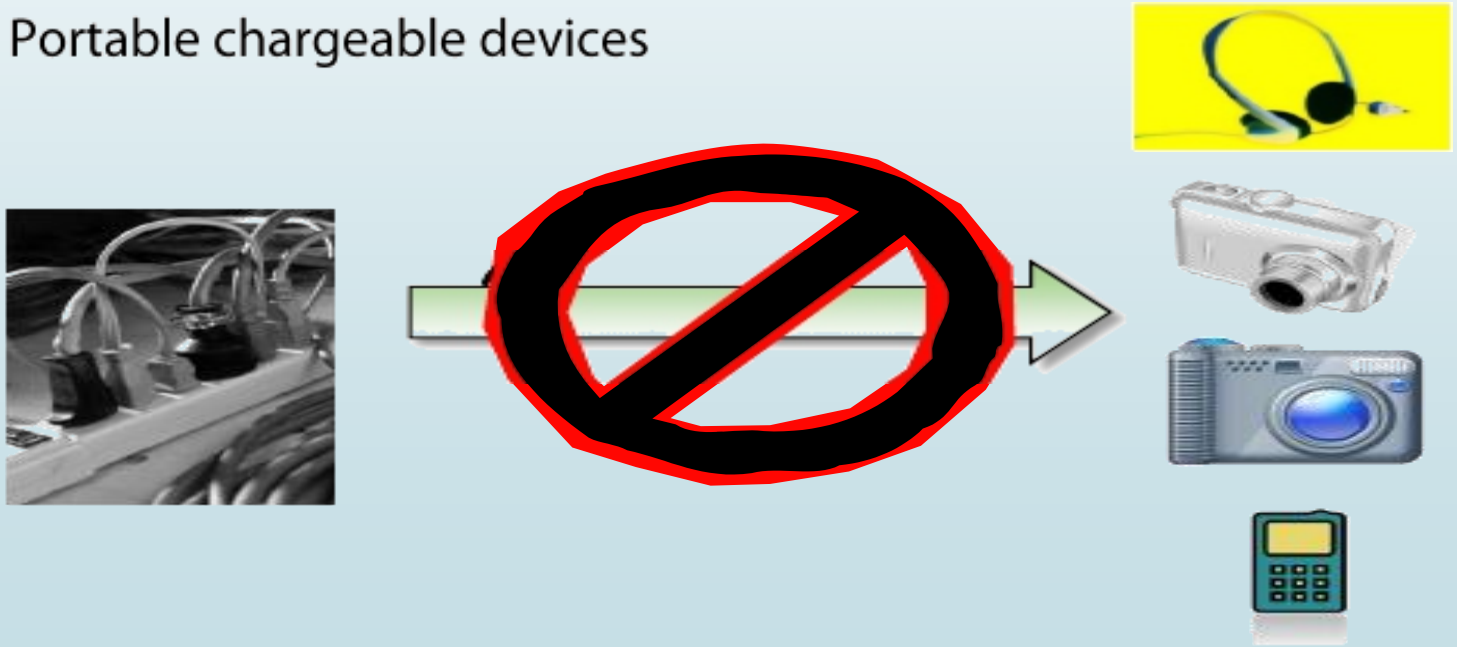
Dependency Inversion Principle

Dependency Inversion Principle

Instead of lower level modules defining an interface that higher level modules depend on, higher level modules define an interface that lower level modules implement

Example:

Portable chargeable devices



Dependency Inversion Principle

- "High-level modules should not depend on low-level modules. Both should depend on abstractions."

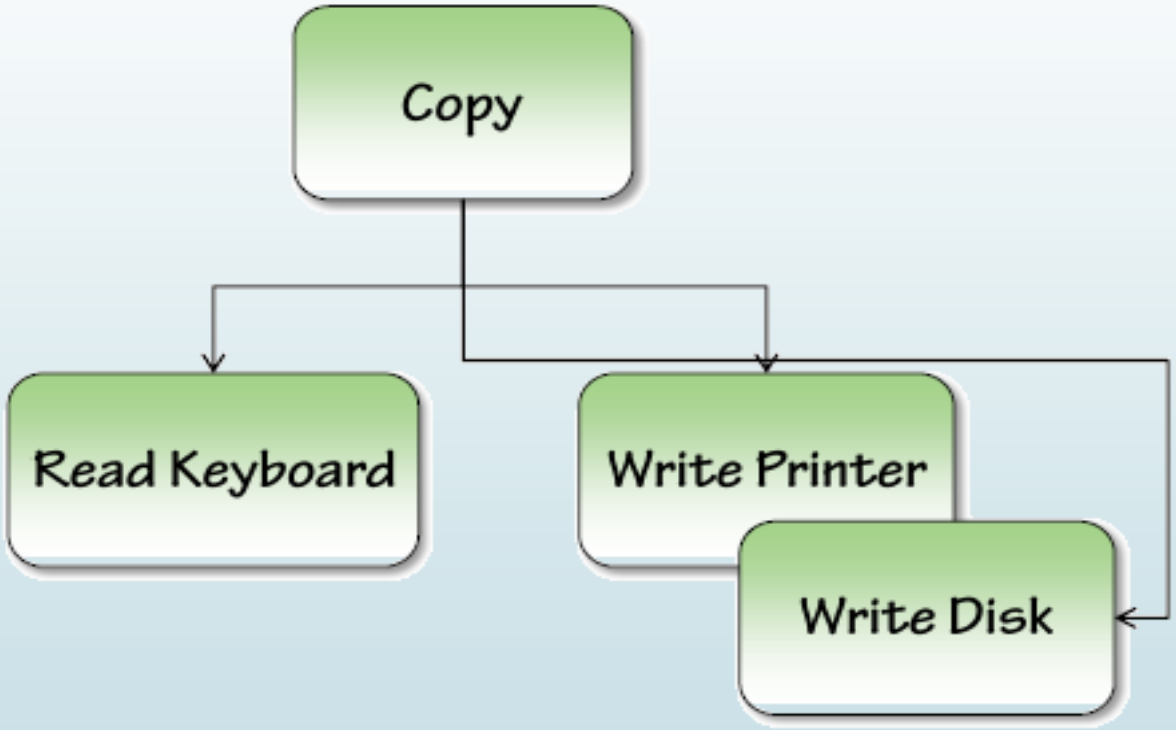
- "Abstractions should not depend on details, but rather details should depend on abstractions"

Robert Martin, Agile Principles, Patterns, and Practices in C#

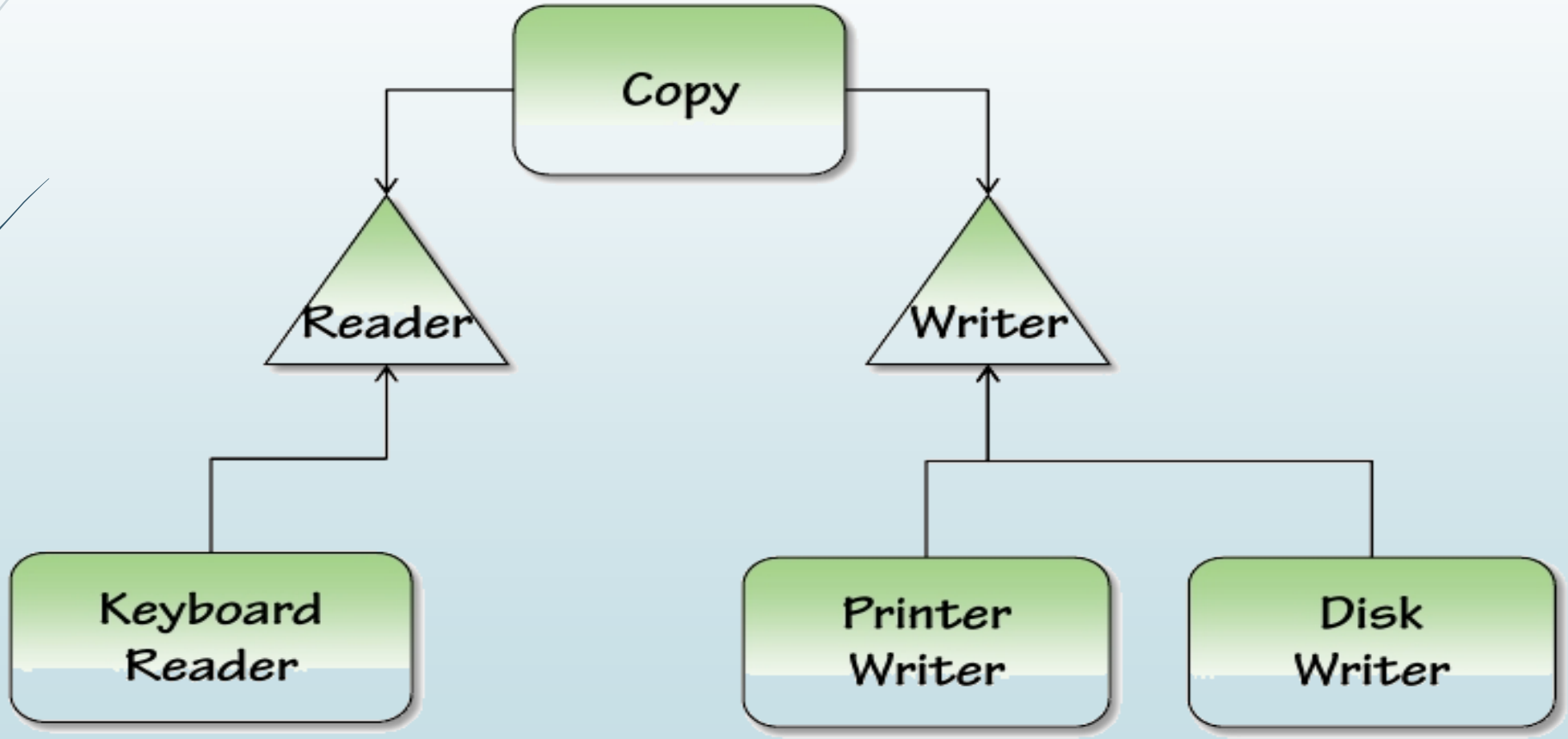
What are dependencies ?

- Framework
- Third Party Libraries
- Database
- File System
- Web services
- System Resources(Clock)
- The new Keyword
- Static methods

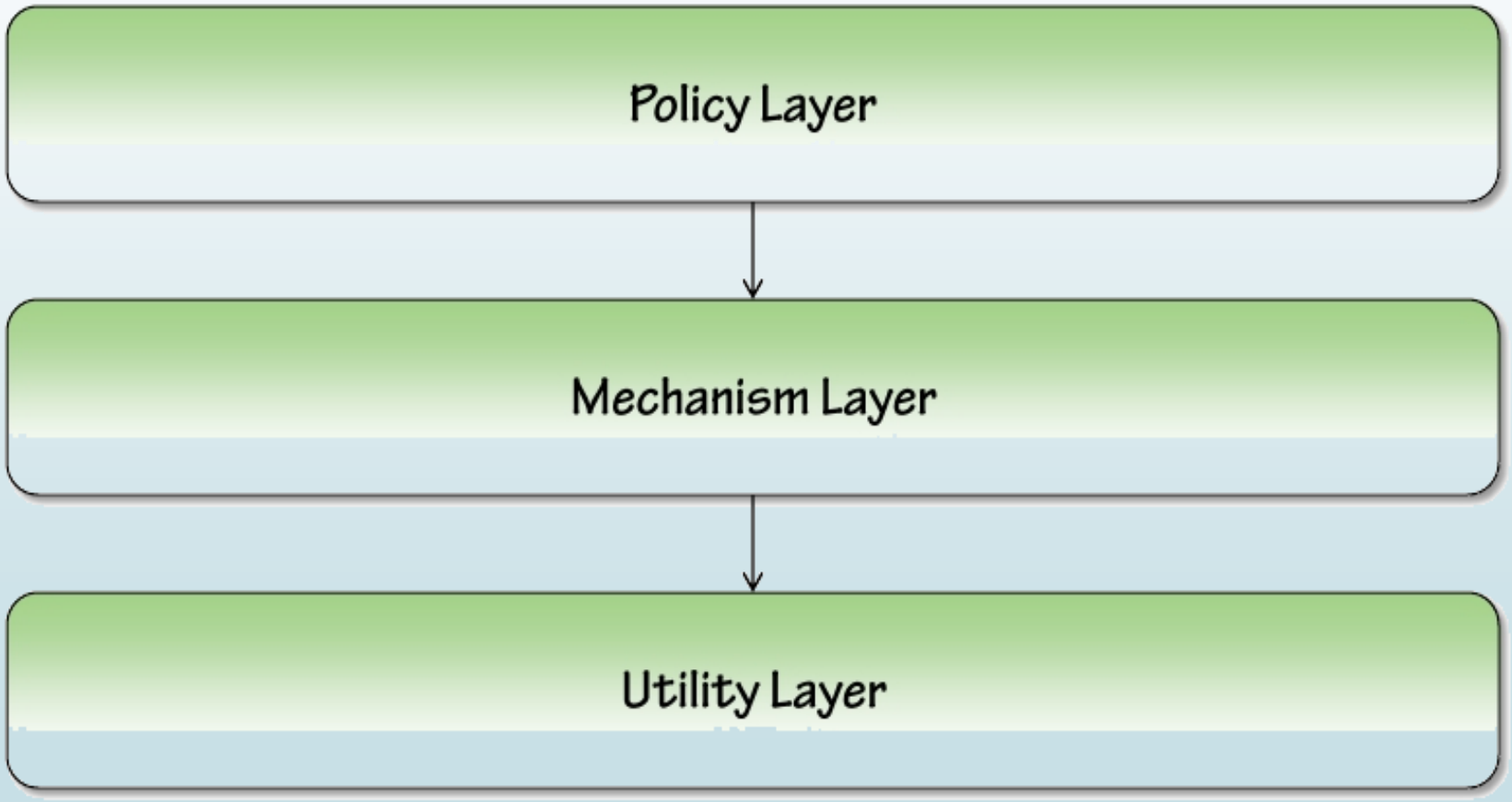
DIP Example “Copy Program”



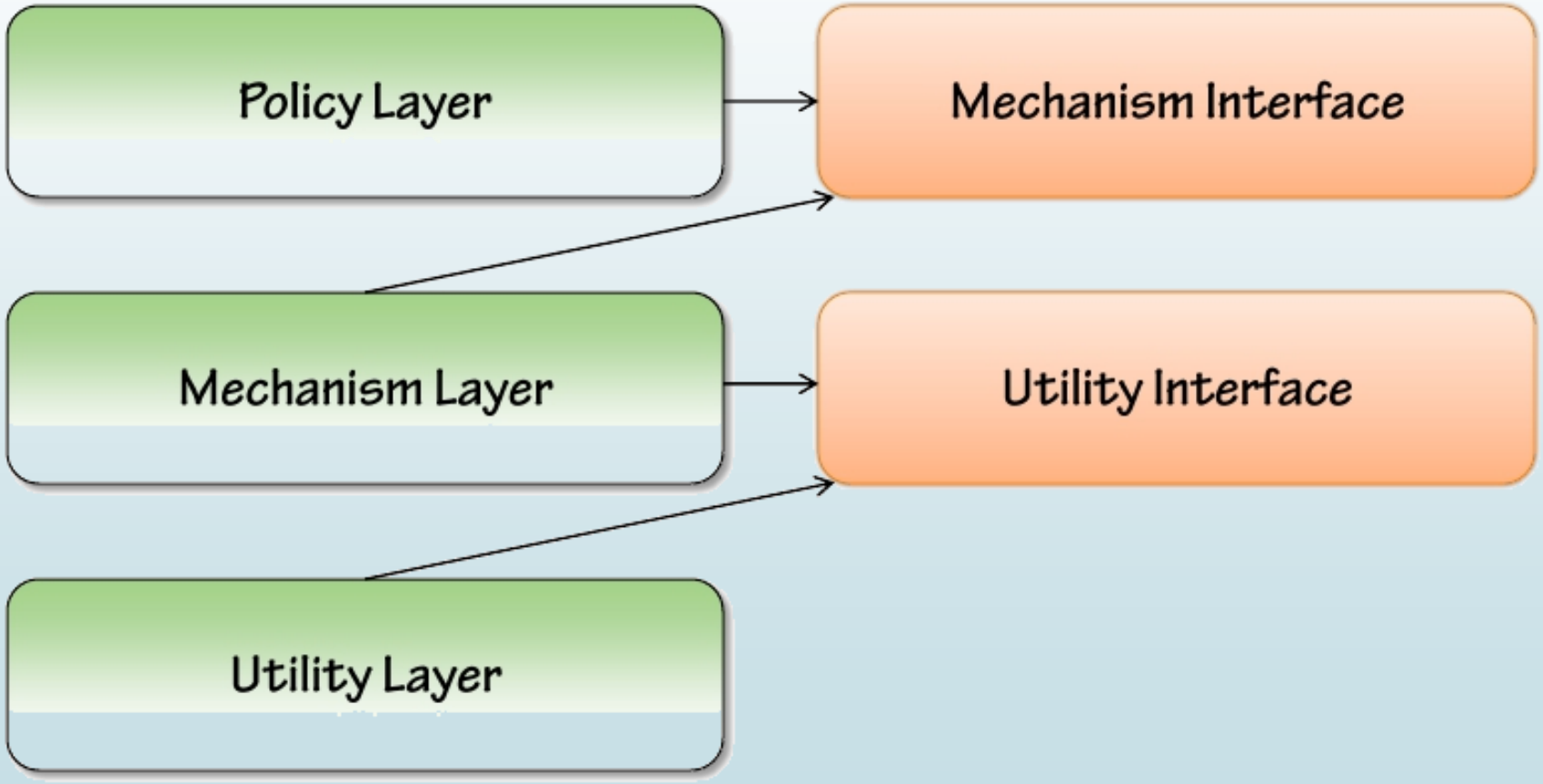
DIP Example “Copy Program” (Cont.)



Layering (Traditional)



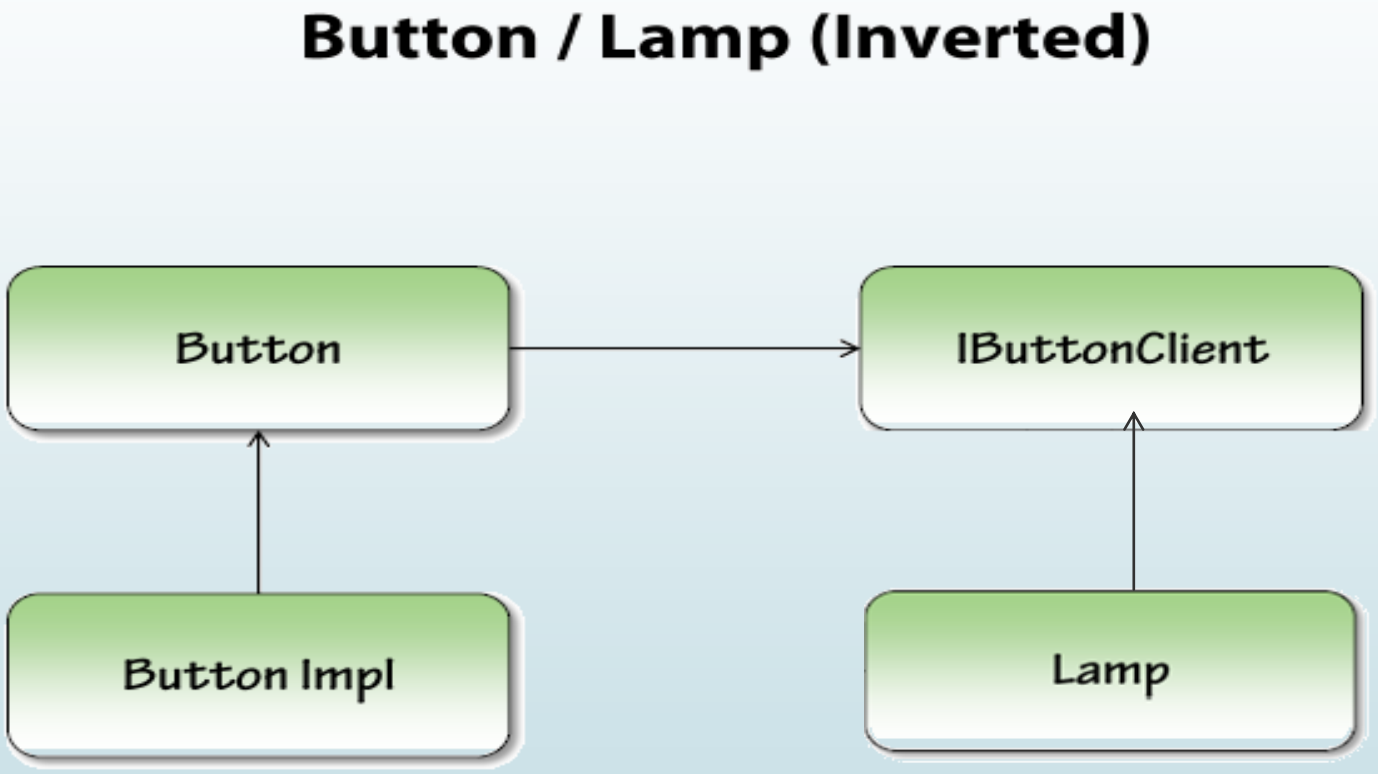
Layering (DIP)



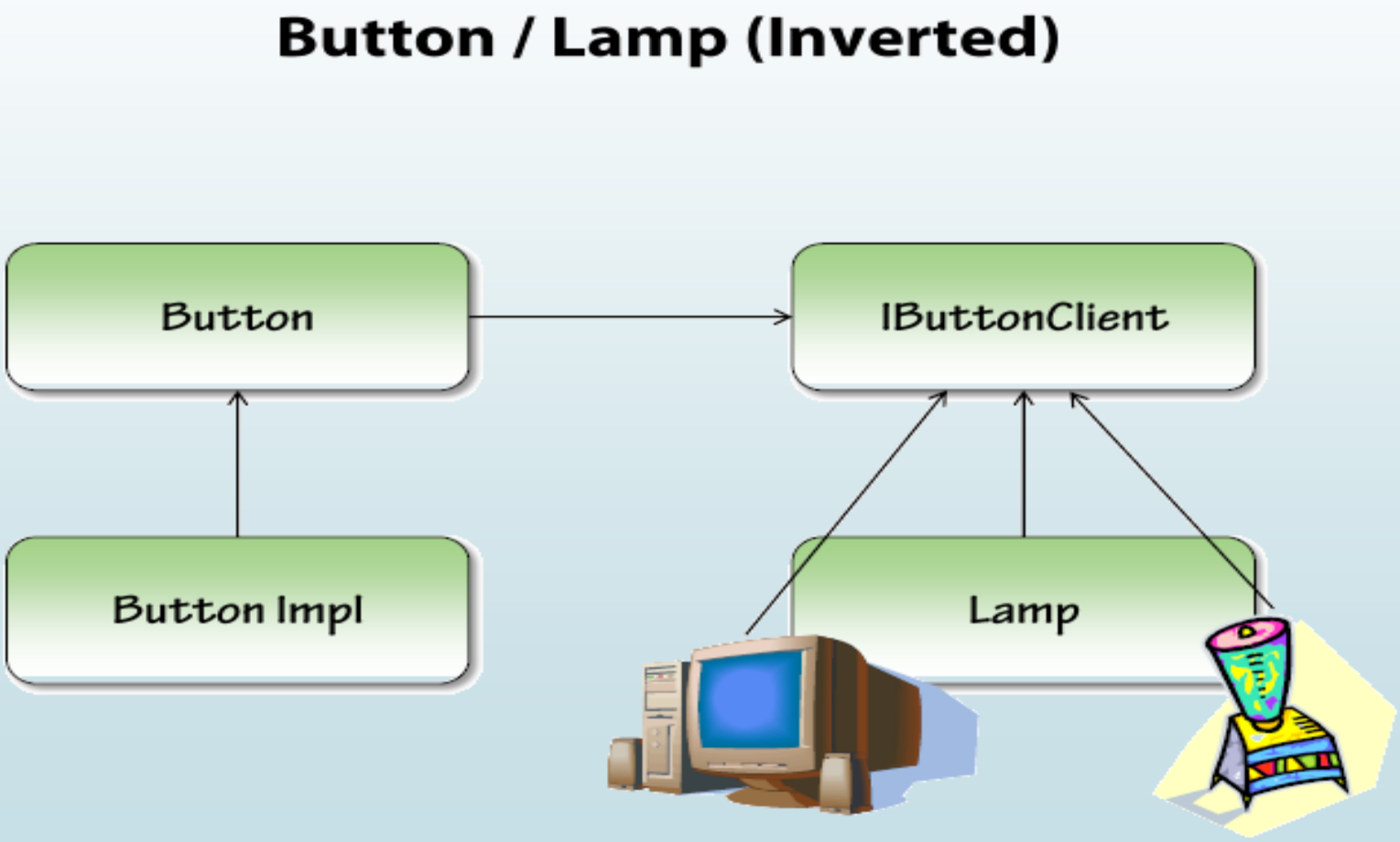
DIP Example

Button / Lamp

DIP Example (Cont.)



DIP Example (Cont.)



DIP Smells

- Using of new Keyword:

```
Foreach(var item in cart.Items)
{
    var inventorySystem = new InventorySystem();
    .....
}
```

DIP Smells

- Using static methods/properties:

```
message.Subject = “Your order placed on “ +
DataTime.Now.ToString();
```

OR

```
DataAccess.SaveCustomer(myCustomer);
```

Summary

- Depend on abstractions rather than concrete types
- Don't force high-level modules to depend on low-level modules through direct instantiation or static method calls
- Declare class dependencies explicitly in their constructors
- Related Fundamentals:
  - Single Responsibility Principle
  - Interface Segregation Principle
  - Façade Pattern

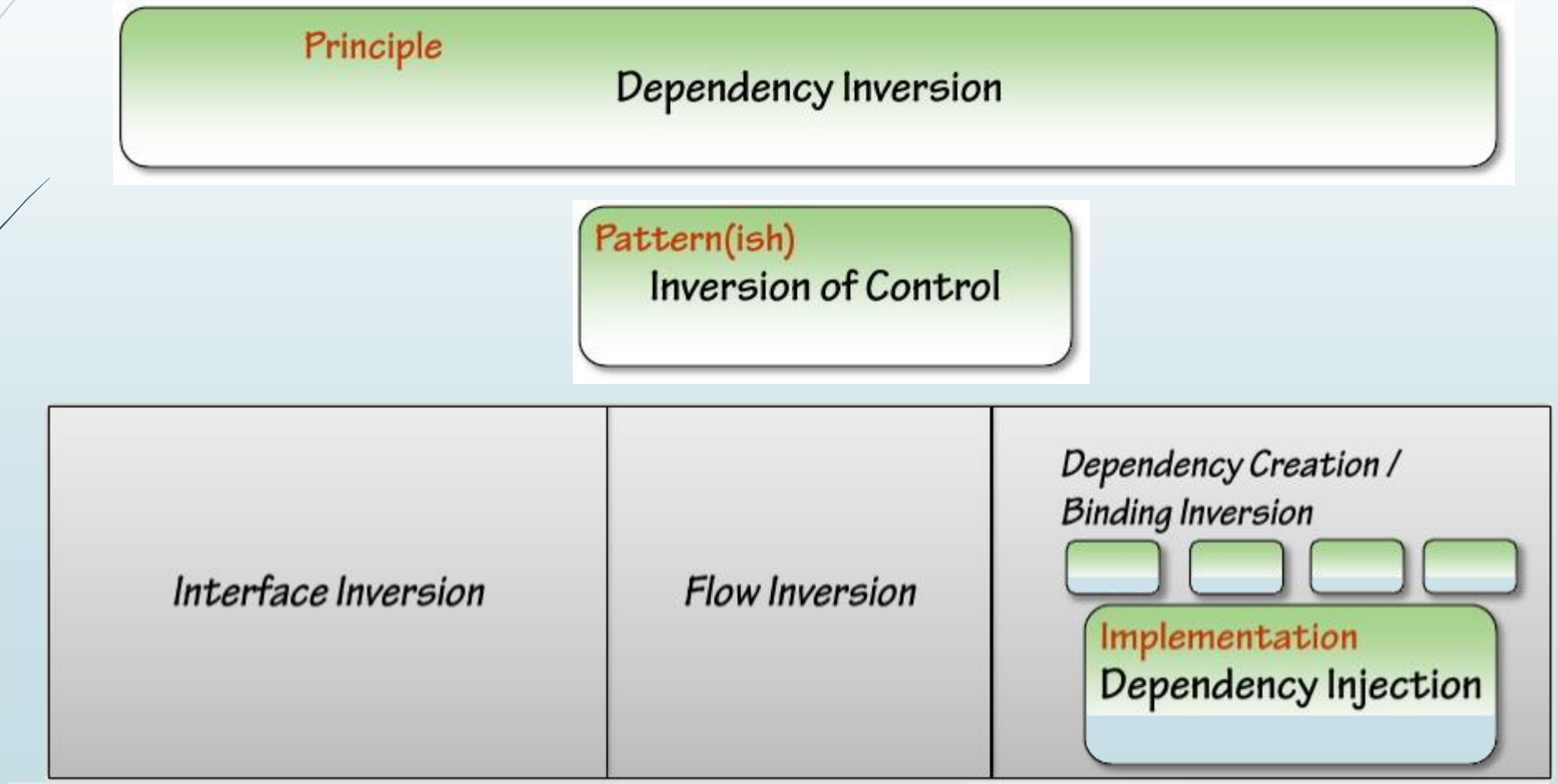
IOC and DIP

- How does IOC relate to Dependency Inversion Principle (DIP)?

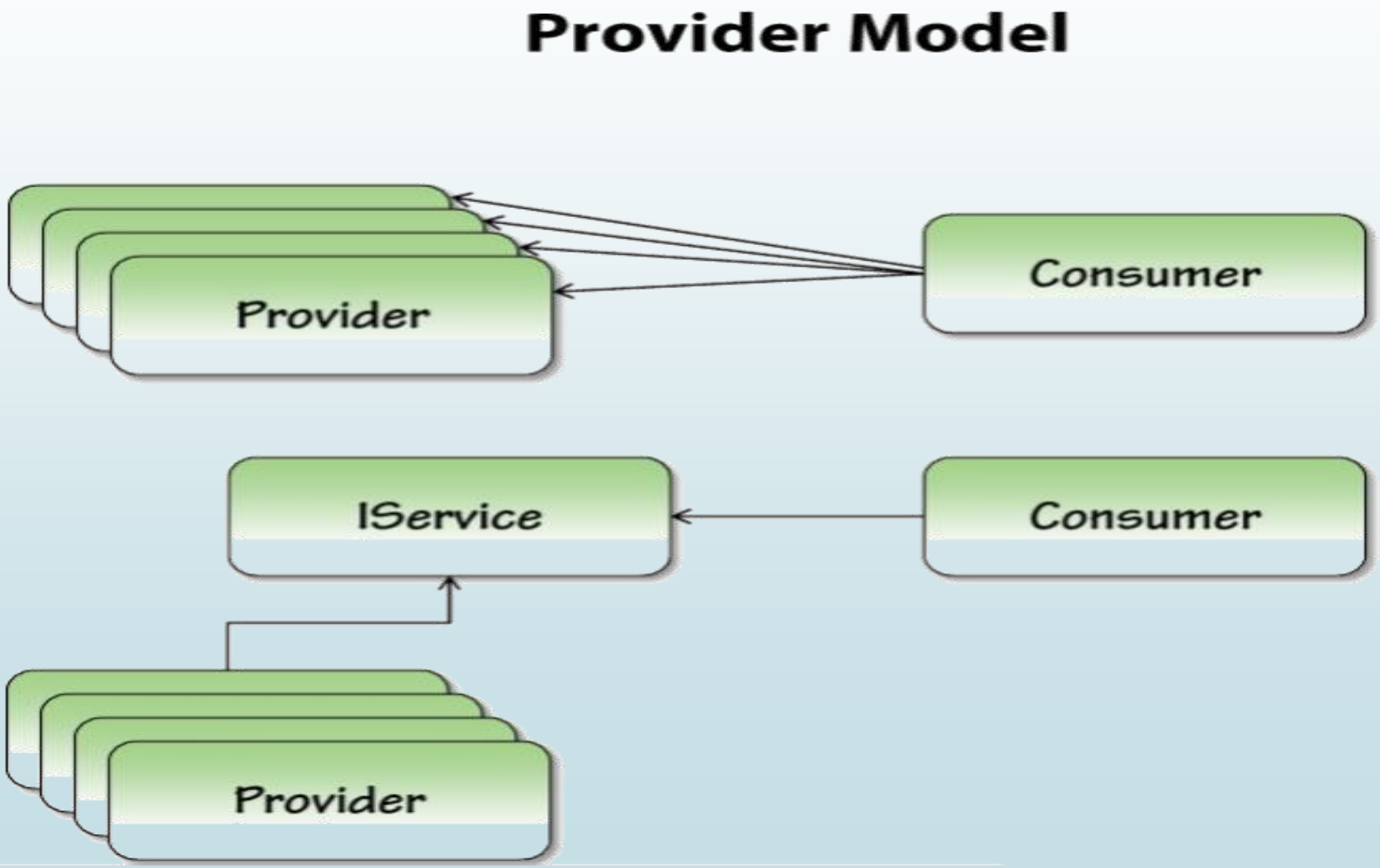
Inversion of Control Types

- Interface Inversion
- Flow Inversion
- Dependency Creation/Binding Inversion

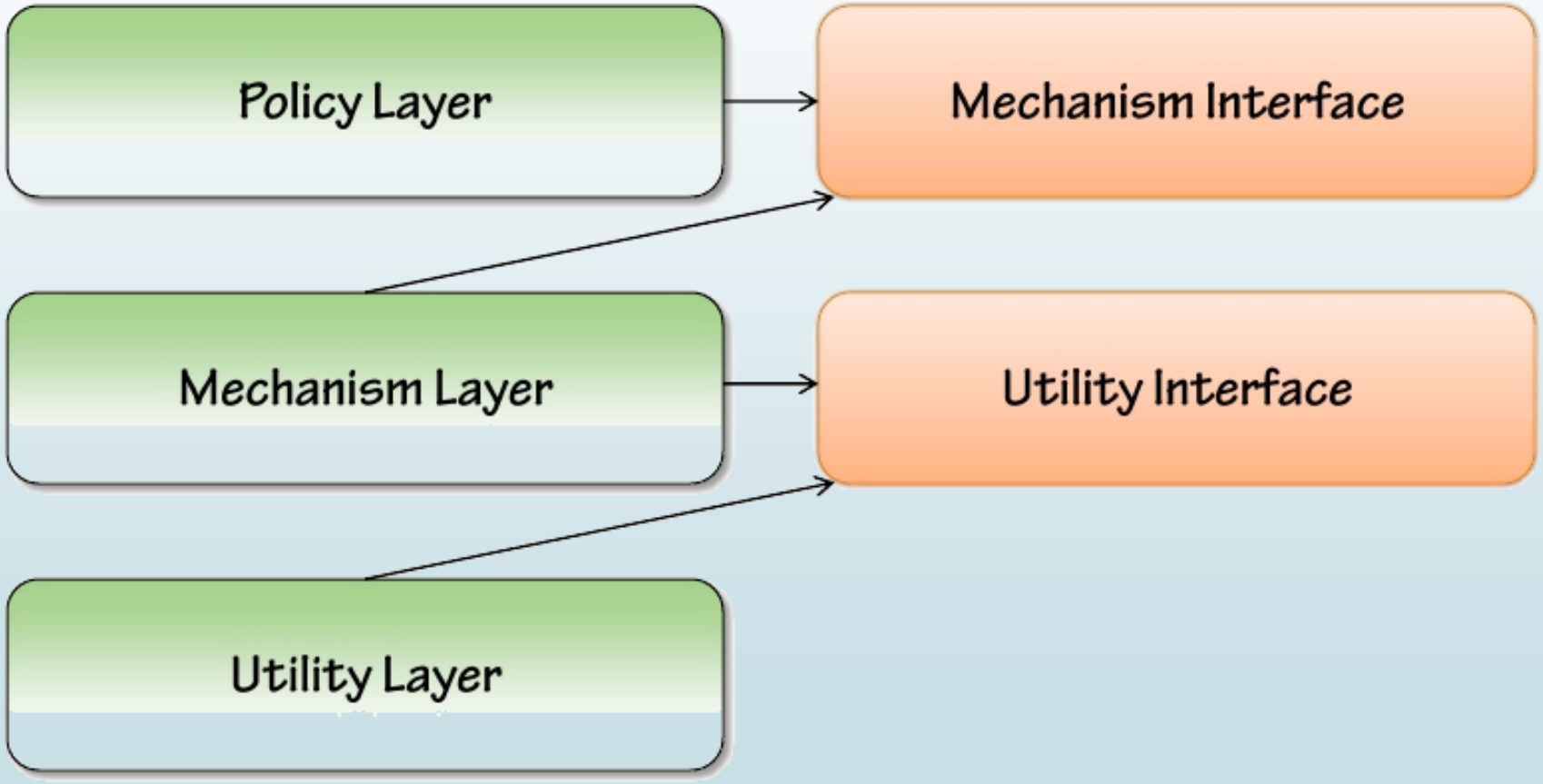
Fitting it all together



Interface Inversion

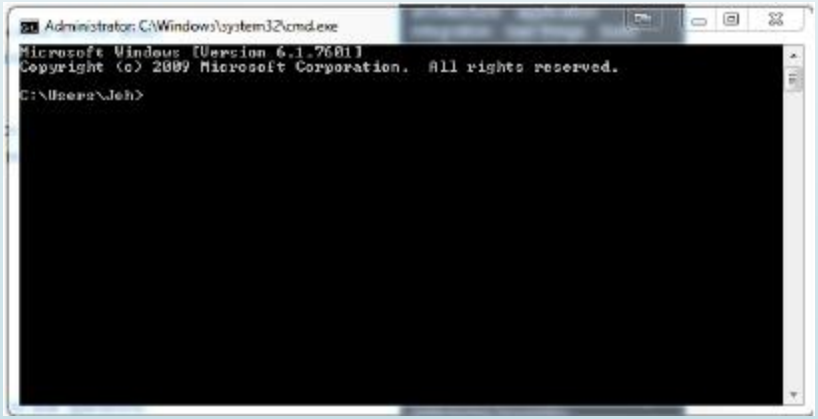


Layering (DIP)



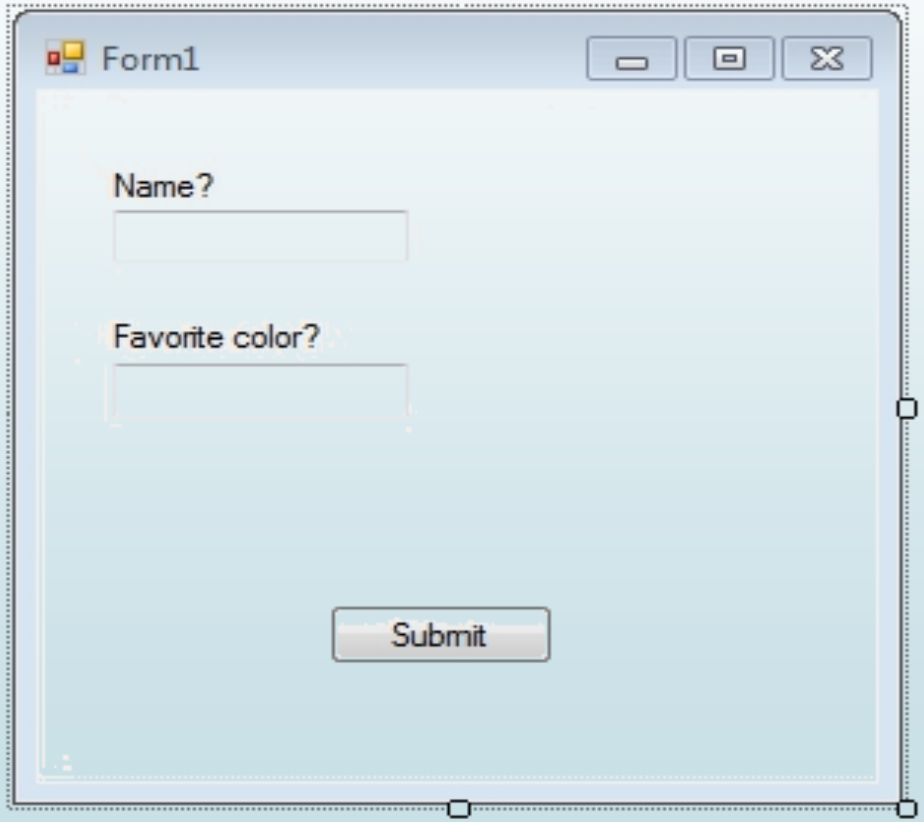
Flow Inversion

- Normal Flow = Procedural
- Think command line program
  - What is your name?
    - Bob
  - What is your favorite color?
    - Blue



Flow Inversion (Cont.)

- Inverted Flow = GUI

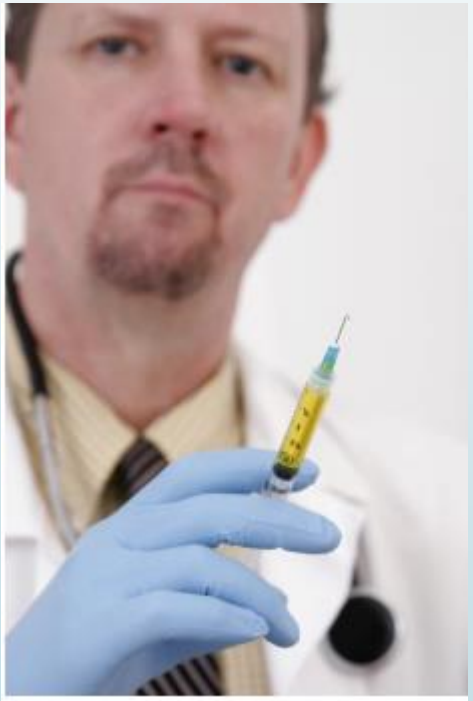


Creation Inversion

- More than just Dependency Injection
- Factory Pattern
  - Button button = ButtonFactory.CreateButton();
- Service Locator
  - Button button = ServiceLocator .Get<Button>;
- Dependency Injection
  - Button button = Container.Resolve<Button>;
- More …

Dependency Injection

- Definition:
  - A type of IoC where we move the creation and binding of a dependency outside of the class that depends on it.



IoC Container (Just Mapper)

Map this	To this
ICreditCard	MasterCard
ITheme	CalssicTheme
ICalcService	MyCalcService
…	….

- You need a method to add a new Map
  - `container.Register<Ibutton, CalssicButton>`
- And another one to get a type
  - `IButton b = container.Resolve<IButton>();`

Dependency Injection Types

- Setter Injection
- Constructor Injection
- Interface Injection

References

- Clean Code by Robert C. Martin
- Agile Principles, Patterns, and Practices by Robert C. Martin and Micah Martin
- http://www.oodesign.com/design-principles.html
- The Principles of OOD http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
- http://en.wikipedia.org/