

# DESIGN PATTERNS

IT Faculty

01/01/2020



KHOA CÔNG NGHỆ THÔNG TIN  
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

# Contents

---



1. Object Oriented Design

2. Design Patterns

3. Creational Patterns



# Object Oriented Design



# Introduction

---

## Design Patterns

### Creational Patterns

Abstract factory

Builder

Factory method

Prototype

Singleton

- **Object oriented design** is a process of planning a software system where objects will interact with each other to solve specific problems.
- The saying goes,  
*“Proper object oriented design makes a developer’s life easy, whereas bad design makes it a disaster.”*
- Class design principles: SOLID



# Single responsibility Principle (SRP)

---

## Principle

*"Every software module should have only one reason to change".*

- Software module: class, function etc.
- Reason to change: responsibility



# Single responsibility Principle (SRP)

## Design Patterns

### Creational Patterns

Abstract factory  
Builder  
Factory method  
Prototype  
Singleton





# Open Close Principle (OCP)

## Design Patterns

### Creational Patterns

Abstract factory  
Builder  
Factory method  
Prototype  
Singleton

## Principle

*"Software modules should be closed for modifications but open for extensions."*

- Solution which will not violate OCP
  - Use of inheritance



# Open Close Principle (OCP)

## Design Patterns

### Creational Patterns

- Abstract factory
- Builder
- Factory method
- Prototype
- Singleton



## Open Closed Principle

You don't need to rewire your MoBo to plug in "Mr Happy"





# Liskov substitution principle (LSP)

## Principle

*"Subclasses should be substitutable for base classes."*

- The best way to implement the LSP is by implementing correct inheritance hierarchy.

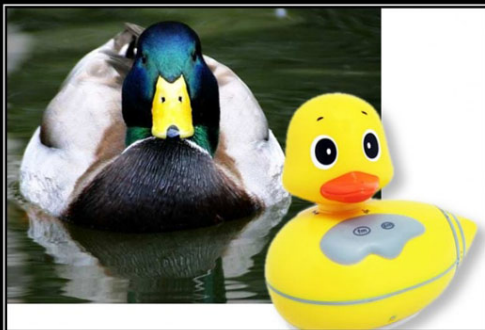


# Liskov substitution principle (LSP)

## Design Patterns

### Creational Patterns

Abstract factory  
Builder  
Factory method  
Prototype  
Singleton



## LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You  
Probably Have The Wrong Abstraction



# Interface Segregation principle (ISP)

---

## Principle

*"Clients should not be forced to implement interfaces they don't use."*

- We should prefer many client interfaces rather than one general interface and each interface should have a specific responsibility.



# Interface Segregation principle (ISP)

## Design Patterns

### Creational Patterns

Abstract factory  
Builder  
Factory method  
Prototype  
Singleton



## Interface Segregation Principle

You want me to plug this in *where?*



# Dependency Inversion principle (DIP)

---

## Design Patterns

### Creational Patterns

Abstract factory

Builder

Factory method

Prototype

Singleton

## Principle

*“High-level modules should not depend upon low-level modules. Both should depend upon abstractions.”*

*“Abstractions should not depend on details. Details should depend on abstractions.”*



# Dependency Inversion principle (DIP)

## Design Patterns

### Creational Patterns

Abstract factory  
Builder  
Factory method  
Prototype  
Singleton



## Dependency Inversion Principle

Would you solder a lamp directly  
to the electrical wiring in a wall?



# Design Patterns



# Introduction

---

## Design Patterns

### Creational Patterns

Abstract factory

Builder

Factory method

Prototype

Singleton

- One of the interesting things about software development is that when we create a software system, we are actually modeling a real-world system.
- To write the business software systems, the developers must thoroughly understand the business models.
- A **design pattern** is a common solution to a common problem in a given context.
- Patterns lend themselves perfectly to the concept of reusable software development.





# Why Design Patterns?

---

## Design Patterns

### Creational Patterns

Abstract factory  
Builder  
Factory method  
Prototype  
Singleton

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem.

- The **pattern name** is a handle we can use to describe a design problem, its solutions, and consequences in a word or two
- The **problem** describes when to apply the pattern. It explains the problem and its content.
- The **solution** describes the elements that make up the design, their relationships, responsibilities, and collaborations.
- The **consequences** are the results and trade-offs of applying the pattern.



# Classification of patterns

---

## Design Patterns

### Creational Patterns

Abstract factory  
Builder  
Factory method  
Prototype  
Singleton

Three main groups of patterns:

- **Creational patterns** create objects for us, rather than having us instantiate objects directly. This gives our program more flexibility in deciding which objects need to be created for a given case.
- **Structural patterns** help us compose groups of objects into larger structures, such as complex user interfaces or accounting data.
- **Behavioral patterns** help us define the communication between objects in our system and how the flow is controlled in a complex program.



# Creational Patterns

- Abstract factory
- Builder
- Factory method
- Prototype
- Singleton

# Intent

---



- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- A hierarchy that encapsulates: many possible “platforms”, and the construction of a suite of “products”.
- The `new` operator considered harmful.



# Problem

---

- If an application is to be portable, it needs to encapsulate platform dependencies.
- These “platforms” might include: windowing system, operating system, database, etc.
- Too often, this encapsulation is not engineered in advance, and lots of `#ifdef` case statements with options for all currently supported platforms begin to procreate like rabbits throughout the code.

# Intent

---



- Separate the construction of a complex object from its representation so that the same construction process can create different representations.
- Parse a complex representation, create one of several targets.



# Problem

---

## Design Patterns

### Creational Patterns

Abstract factory

**Builder**

Factory method

Prototype

Singleton

- An application needs to create the elements of a complex aggregate.
- The specification for the aggregate exists on secondary storage and one of many representations needs to be built in primary storage.

# Intent

---



- Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- Defining a “virtual” constructor.
- The `new` operator considered harmful.



# Problem

---



- A framework needs to standardize the architectural model for a range of applications, but allow for individual applications to define their own domain objects and provide for their instantiation.

# Intent

---



- Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
- Co-opt one instance of a class for use as a breeder of all future instances.
- The `new` operator considered harmful.

# Problem

---



Application “hard wires” the class of object to create in each “new” expression.

# Intent

---



- Ensure a class has only one instance, and provide a global point of access to it.
- Encapsulated “just-in-time initialization” or “initialization on first use”.



# Problem

---

## Design Patterns

### Creational Patterns

Abstract factory

Builder

Factory method

Prototype

**Singleton**

- Application needs one, and only one, instance of an object. Additionally, lazy initialization and global access are necessary.

# References

---



Deitel, P. (2016).

*C++: How to program.*

Pearson.



Gaddis, T. (2014).

*Starting Out with C++ from Control Structures to Objects.*

Addison-Wesley Professional, 8th edition.



Jones, B. (2014).

*Sams teach yourself C++ in one hour a day.*

Sams.