* Creational Pattern

Creational design patterns abstract the instantiation process

* Systems evolve to depend more on object composition than class inheritance
* 关键点: 1) 外界获得的是abstract class或interface, 可以用这些method
* 2）隐藏了具体哪些class被用到和这些class如何被初始化和合并在一起的
* Abstract Factory
* 1) Provide an interface for creating families of related or dependent objects without specifying their concrete classes. (只提供了构造一个object的interface)
* clients only have to commit to an interface defined by an abstract class, not a particular concrete class
* 2) Abstract factory define common instantiate interface, Concrete factory create concrete object
* a) AbstractFactory factory = new RectangularFactory();
* b) Shape shape = factory.createShape();
* 3) Applicability
* a system should be independent of how its products are created, composed, and represented.
* a system should be configured with one of multiple families of products. ( 有一系列的同样的产品, 都继承这个interface)
* a family of related product objects is designed to be used together, and you need  to enforce this constraint.  (里面有多个create的method)
* you want to provide a class library of products, and you want to reveal just their  interfaces, not their implementations.

4) Benefits

- Isolates concrete class

- It makes exchanging product families easy (只需要更改Concrete Factory)

* - It promotes consistency among products
* \*\*\* Supporting new kinds of products is difficult  and hard to use subclass features(因为都是用interface来实现, 同样的method)
* 5) Implementations
* Concrete Factory is singleton
* Abstract Factory define static factory method to create instance
* - The concrete factory is initialized with a prototypical instance of each product in the family, and it creates a new product by cloning its prototype
* Defining extensible factories
* - A more flexible but less safe design is to add a parameter to operations that create objects. Only one create method
* 6) Related Patterns
* AbstractFactory classes are often implemented with factory methods but they can also be implemented using Prototype
* A concrete factory is often a singleton
* Builder
* 1. Intent (for complex object)
* Separate the construction of a complex object from its representation so that the same construction process can create different representations.

2. Structure

* Directory use a builder(build parts of a object) to construct object, each concrete builder will create a concrete object

3. Applicability

* the algorithm for creating a complex object should be independent of the parts that make up the object and how they're assembled.
* the construction process must allow different representations for the object that's constructed.

4. Consequences

* It lets you vary a product's internal representation (don’t reveal internal class a builder access)
* It isolates code for construction and representation
* It gives you finer control over the construction process
* 5. Related Patterns
* Abstract Factory (87) is similar to Builder in that it too may construct complex objects. The primary difference is that the Builder pattern focuses on constructing a complex object step by step. Abstract Factory's emphasis is on families of product objects (either simple or complex). Builder returns the product as a final step, but as far as the Abstract Factory pattern is concerned, the product gets returned immediately.
* A Composite (163) is what the builder often builds.
* Factory Method(类似abstract factory, 都是subclass来实现create)
* 1. 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.

2. Applicability

* a class can't anticipate the class of objects it must create.
* a class wants its subclasses to specify the objects it creates
* classes delegate responsibility to one of several helper subclasses, and you want  to localize the knowledge of which helper subclass is the delegate

3. Structure

* Creator { public Product factoryMethod(); }

4. Consequences

* Factory methods eliminate the need to bind application-specific classes into your code
* Provides hooks for subclasses
* Connects parallel class hierarchies

5. Variations

* Parameterized factory methods 🡺 createDocument (DocumentType type)
* Factory methods could be interface or abstract method

6. Related Patterns

* Abstract Factory (87) is often implemented with factory methods. The Motivation example in the Abstract Factory pattern illustrates Factory Method as well.
* Factory methods are usually called within Template Methods (325). In the document example above, NewDocument is a template method.
* Prototypes (117) don't require subclassing Creator. However, they often require an Initialize operation on the Product class. Creator uses Initialize to initialize the object. Factory Method doesn't require such an operation.

Prototype Pattern

1. Intent (像汽车一样, 每个车型都有一个框架)

* Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

2.

Singleton

1. Intent

* Ensure a class only has one instance, and provide a global point of access to it

2. Applicability

* there must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
* when the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code

3. Structure

SingletonClass { static element; public SingletonClass instanceOf() {} }

4. Benefits

* Controlled access to sole instance
* Permits refinement of operations and representation
* Permits a variable number of instances

5. Implementations

* Ensuring a unique instance (private constructor, public getInstanceOf(), static instance)
* Public Singleton instance() {

if(instance == null) {

* + - Synchronized(this) {

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

* + - }

} return instance; }

* Subclass: registry of singletons

Have a dictionary to store singletons, and get by name