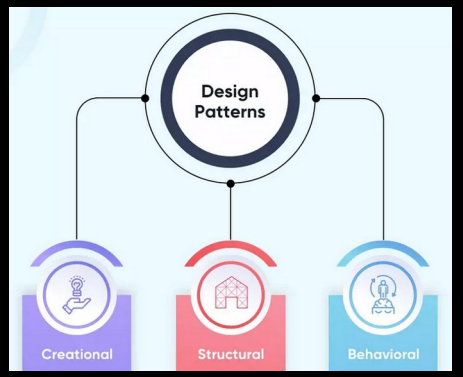


Creational Design Pattern << Factory >>

Ajith de Silva @epita.fr | 2025

Creational Design Patterns :: About

Concerned with Object-Oriented concept

Offer how to create instances form objects in:

… Flexible

… Efficient

… Maintainable

Provide INTERFACE for creating Instances in SUPPER CLASS

Allows SUBCLASSES to alter TYPE of instance to create

Great Decoupling with other code. HIGH Flexibility

Centralize creational logic

Creational Design Patterns

1. **Singleton**
   - *… One INSTANCE.*
   - *… Global access*
2. **Factory**
   - *… Interface to create instances in supper class.*
   - *… Subclass decides type of Instance`*
3. **Abstract Factory**
   - *… Group similar Factory based on Instances types*
4. **Builder**
   - *… Create complex instance step by step*
5. **Prototype**
   - *… Clone Existing instance*

Factory Design Pattern

1. **In Brief….**

   … provides interface for instance creation in supper class. (Separate constructions of instances)

   … subclass can decide what instance to create based on type

2. **HOW ??**

   Define Abstract/Interface class with required methods/properties

   … this will be the uniform for concrete classes

   … contains multiple abstract methods

   Implement concrete class by implementing the Define Abstract/Interface

   Create Supper Class with Static Method for instance creation

   … expecting the what instance type as parameter

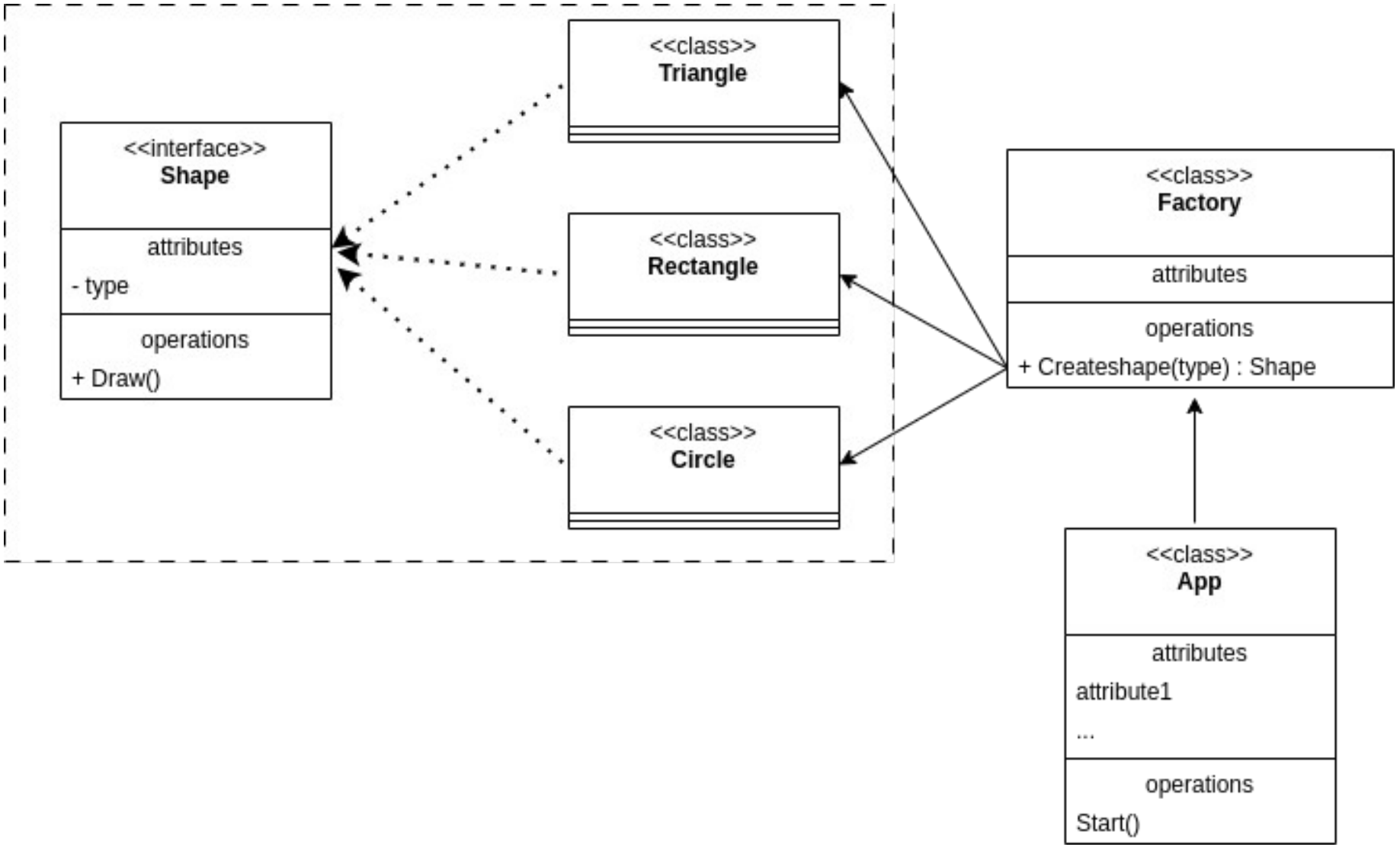
   … create requested instance by type and return it

Factory Design Pattern :: MORE

1. **High Lose Coupling**
   - … caller/client code does not have any binding with the type
   - … maintainability & expandability is HIGH
   - … single responsibility principle

2. **Complexity??**
   - *… Will be more complex when you have more subclasses*

Factory Design Pattern :: UML



Factory Design Pattern :: How to Implement

1. **Define the interface with required methods & properties**

2. **Implement the interface in concrete class**

3. **Create a Factory class**

   … must have a method to accept type parameter and return requested instance

4. **Client/caller will use the Factory class to retrieve required instance by passing type**

Factory Design Pattern :: Practical

1. Check/study the no factory sample
2. Check/study the factory sample
3. Run/Debug and check how it works
4. Extend concrete implementation
5. Introduce NEW cake and use it the caller code