Creational patterns

In software engineering, creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or added complexity to the design. Creational design patterns solve this problem by somehow controlling this object creation.

- Abstract Factory

  Creates an instance of several families of classes
- Builder

  Separates object construction from its representation
- Factory Method

  Creates an instance of several derived classes
- Object Pool

  Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
- Prototype

  A fully initialized instance to be copied or cloned
- Singleton

  A class of which only a single instance can exist

Rules of thumb

1. Sometimes creational patterns are competitors: there are cases when either Prototype orAbstract Factory could be used profitably. At other times they are complementory: Abstract Factory might store a set of Prototypes from which to clone and return product objects,Builder can use one of the other patterns to implement which components get built. Abstract Factory, Builder, and Prototype can use Singleton in their implementation.
2. Abstract Factory, Builder, and Prototype define a factory object that's responsible for knowing and creating the class of product objects, and make it a parameter of the system. Abstract Factory has the factory object producing objects of several classes. Builder has the factory object building a complex product incrementally using a correspondingly complex protocol.Prototype has the factory object (aka prototype) building a product by copying a prototype object.
3. Abstract Factory classes are often implemented with Factory Methods, but they can also be implemented using Prototype.
4. Abstract Factory can be used as an alternative to Facade to hide platform-specific classes.
5. Builder focuses on constructing a complex object step by step. Abstract Factory emphasizes a family of product objects (either simple or complex). Builder returns the product as a final step, but as far as the Abstract Factory is concerned, the product gets returned immediately.

6. Builder is to creation as Strategy is to algorithm.
7. Builder often builds a Composite.
8. Factory Methods are usually called within Template methods.
9. Factory Method: creation through inheritance. Prototype: creation through delegation.
10. Often, designs start out using Factory Method (less complicated, more customizable, subclasses proliferate) and evolve toward Abstract Factory, Prototype, or Builder (more flexible, more complex) as the designer discovers where more flexibility is needed.
11. Prototype doesn't require subclassing, but it does require an Initialize operation. Factory Method requires subclassing, but doesn't require Initialize.
12. Designs that make heavy use of the Composite and Decorator patterns often can benefit fromPrototype as well.