Factory Pattern

The **Factory Method Pattern** defines an interface for creating an object but lets the subclasses decide which class instantiate. Factory method lets a class defer instantiation to subclasses.

Abstract Factory Pattern

The **Abstract Factory Pattern** provides an interface for creating families of related or dependent objects without specifying their concrete classes.

Factory Patterns

- *Creational patterns* abstract the object instantiation process:
  - Hide how objects are created
  - Make system independent of how its objects are created and composed.
- Factory patterns are examples of creational patterns.
- *Class creational patterns* focus on the use of inheritance to decide on the object to be instantiated:
  - Factory method
- *Object creational patterns* focus on the delegation of the instantiation to another object:
  - Abstract Factory

Factory Patterns

- All OO languages have an idiom for object creation
  - Java = new
  - C++ = malloc, new
- Creational patterns allow us to write methods that create new objects *without explicitly using the* ***new*** *operator*.
- All factory patterns encapsulate “object creation”.
- Allows writing of methods that can instantiate different objects and can be extended to instantiate newly-developed objects, all without modifying the method’s code!

Factory Method vs. Abstract Factory

- Both patterns are good at decoupling applications from specific implementations
- Both patterns create objects – that’s their job
- Factory Method uses *inheritance* to decouple applications form specific implementations
- Abstract Factory uses *object composition* to decouple applications form specific implementations

Factory Method

- Uses a class to create objects
- To create objects, extend a class and override a factory method
- Factory method creates objects
- Clients are decoupled from concrete types because subclasses handle object creation

Abstract Factory

- Provides and abstract type for creating a family of products (Advantage: creates a group of related products)
- Subclasses of this type determine how products (i.e., objects in the family) are produced
- To use, instantiate one and pass it into some code that is written to the abstract type
- Clients are decoupled from concrete products
- Abstract Factory provides more flexibility than Factory Method by delegating instantiation to another object instead of changing subclass to be instantiated, but it is more complex than Factory Method as well.

Use Factory Method When

- You need to decouple your client code from the concrete classes you need to instantiate
- You don’t know ahead of time all the concrete classes you are going to need
- To use, just subclass and implement the factory method.

Use Abstract Factory When

- You have families of products you need to create
- You want to make sure your clients create products that belong together

How important is it to use Factories?

- Heuristics for use:
  - Strict interpretation: use factories for every volatile class
  - But don’t use factories for everything: too extreme
  - Initially - don’t start out using factories, add them in as you see the need for them .
- Remember: factories are a complexity that can often be avoided in the early phases.
  - They unnecessarily complicate designs!