Abstract Factory

- Belongs in the Creational Patterns category.

Creational Patterns

- They are patterns that abstract out the object creation. So your code doesn’t have to depend on or know about other concrete classes of whose instances it needs to use.
- We’re attacking OCP and LSP head-on here.
- Abstracts instantiation process.
  - Creational Patterns abstract out the instantiation of objects.
  - We don’t have to rely on the concrete details of the classes we want to use — we abstract the details out.
- Makes system independent of how its objects are created, composed, and represented.
- Encapsulated knowledge about which concrete classes the system uses.
- Hides how instances of these classes are created and put together.
- Examples of Creational Patterns →
  1. Abstract Factory
  2. Factory Method
  3. Singleton
  4. Prototype
  5. Builder

Abstract Factory

> Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

- Abstract Factory → you are *abstracting* the object creation using a *factory*.
- Abstract Factory is an object scope pattern. That is, we rely upon association as the predominant idea, even though we will be using inheritance along the way.
- Create an abstract factory which is going to tell you what objects you are allowed to use for your class.
- In Java, instead of implementing factories, you can use reflection.

Use Abstract Factory when

- system should be independent of how its products are created, composed, and represented.
- system should be configured with one of multiple families of products.
- a family of related product objects must be used together and this constraint need to be enforced.
- you want to reveal only the interface and not the implementation of a class library of products.

Consequences of using Abstract Factory

- Isolates concrete classes.
- Makes exchanging product families easy.
- Promotes consistency among products.
- Supporting new kinds of products is difficult.