Prototype Pattern

> Specify the kind of objects to create using prototypical instance, and create new object by copying this prototype.

- In languages like C++, Java, and C#, `new` is *not* polymorphic, so it's difficult to dynamically create an object using `new`.
- Prototype is often a workaround since the operation of new is not polymorphic.
- If we check the instance type in the copy constructor we'd be violating OCP.
- While factory method has class scope, prototype has object scope (because it uses association).

When to use Prototype Pattern?

- System should be independent of how its products are created, composed, and represented and the classes to be instantiated are specified at runtime.
- You want to avoid creating class hierarchy of factories that parallel the class hierarchy of products.

Consequences of using Prototype

- Adding & removing products at run-time is very easy.
- Specifying new objects by varying values is easy.
- Specifying new objects by varying structure is easy.
- Reducing subclassing.
- Configuring application with classes dynamically.
- Disadvantage → each subclass must implement *clone* operation. This is intrusive. One may argue that this is a bit of a violation of SRP, as each object has to a little bit more responsibility, but that's a compromise as you reduce class hierarchies.

Prototype vs. Other Patterns

- Abstract Factory is a competing Pattern, however, may work together as well.
- Composite & Decorator benefit from Prototype.