**Factory (*Creational Design Pattern*)**

## **Basic Concept**

* Use a specialized object solely to create other objects, like a real-world factory.
* Object creation abstracted away from client so it can focus on it's specific role in the application (seperation of concerns).

## **Actors**

* **Client**
  + Object that requires an instance of another class (view model, code-behind, etc).
  + Rather than creating it itself, delegates this to the factory.
* **Factory**
  + Once invoked, creates a new instance of the product and passes it back to client.
  + Normally contains a Build() method that returns the request product.
* **Product**
  + Object that the client requires which is produced by the factory.

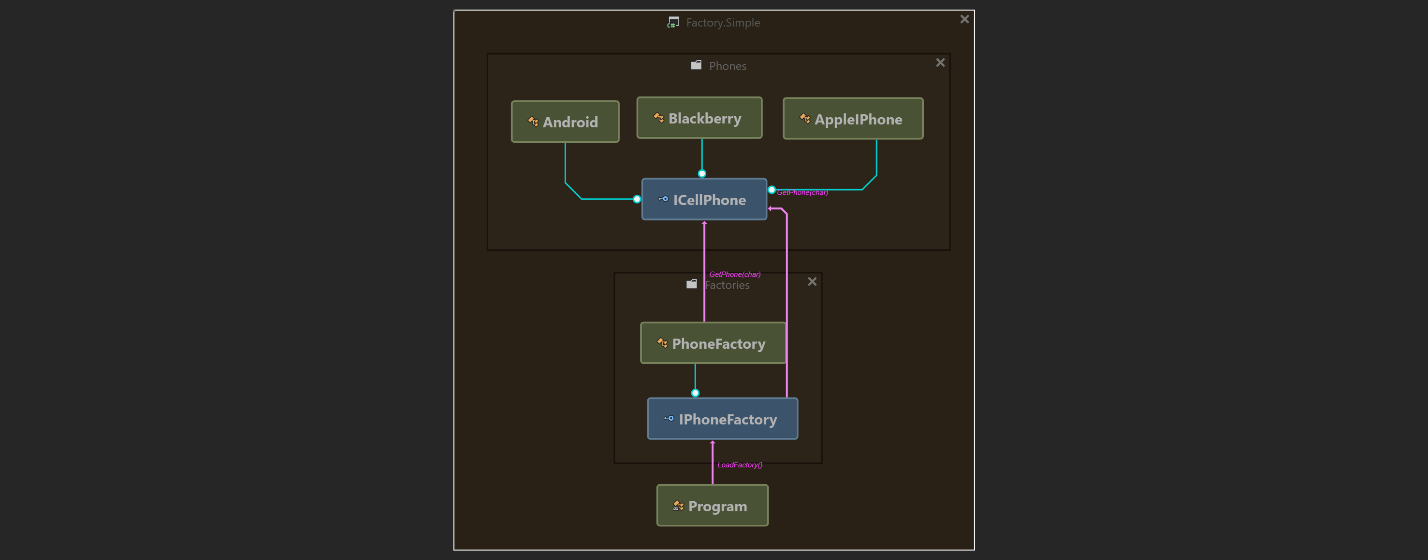


## **Why Interfaces?**

* coding to a contract (interface) rather than a concrete class allows for more resilient, generic code
* can loosen coupling between layers (**ex.** IEnumerable instead Array/List)
* client code (whoever gets an instance of the interface) only has access to the exposed methods/properties in the interface

## **Simple Factory Example**

* single factory concrete-type (PhoneFactory) decides which product to create
* 3 different products (Android, Blackberry, Apple iPhone)
* client simply creates a new PhoneFactory instance, then passes in a variable telling Factory which product to build



## **Complex Factory Example**

* multiple concrete-types for factory, each creates it's own phone type.
* 3 different products, but 2 now implement their own interface (IAndroid, IAppleIPhone)
* client uses reflection to create an instance of the needed factory (using the Fully-Qualified Name of the class)
* client doesn't decide which product gets built, ***configuration does***

