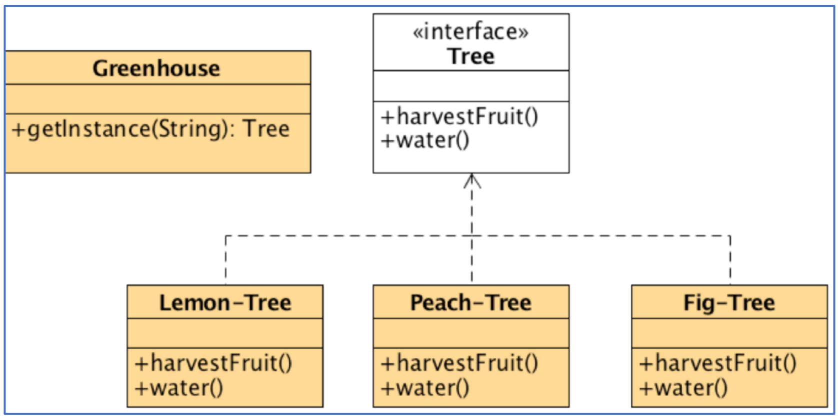
Class

1. **Factory Method: Define an interface for creating an object, but let subclasses decide which class to instantiate**

A static method of a class that returns an object of that class' type.



>>> Polymorphism.

Problem

– A framework needs to standardize the architectural model for a range of applications, but allow for individual applications to define their own domain objects and provide for their instantiation.

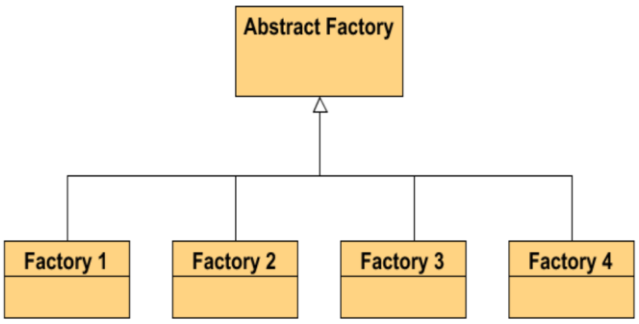
- If you have an inheritance hierarchy that exercises polymorphism, consider adding a polymorphic creation capability by defining a static factory method in the base class.
- Design the arguments to the factory method. What qualities or characteristics are necessary and sufficient to identify the correct derived class to instantiate?
- Consider designing an internal "object pool" that will allow objects to be reused instead of created from scratch.
- Consider making all constructors private or protected.

Object

1. **Abstract Factory: Provide an interface for creating families of related or dependent objects without specifying their concrete classes.**

A hierarchy that encapsulates many possible "platforms", and the construction of a suite of "products".

The Abstract Factory defines a Factory Method per product



Problem

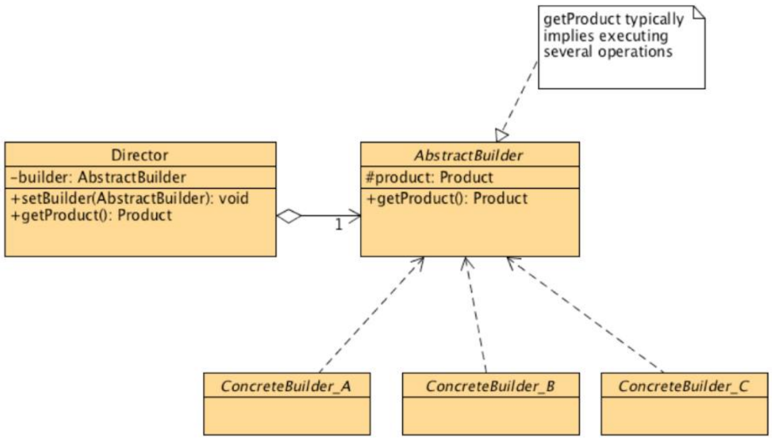
– If an application is to be portable, it needs to encapsulate platform dependencies.

– These "platforms" might include: windowing system, operating system, database, etc.

- Decide if "platform independence" and creation services are the current source of pain.
- Map out a matrix of "platforms" versus "products".
- Define a factory interface that consists of a factory method per product.
- Define a factory derived class for each platform that encapsulates all references to the new operator.
- The client should retire all references to new, and use the factory methods to create the product objects.

2. **Builder: Separate the construction of a complex object from its representation so that the same construction process can create different representations.**

Parse a complex representation, create one of several targets.



Problem

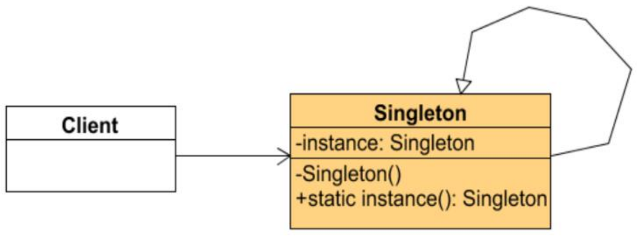
– An application needs to create the elements of a complex aggregate. The specification for the aggregate exists on secondary storage and one of many representations needs to be built in primary storage.

- Decide if a common input and many possible representations (or outputs) is the problem at hand.
- Encapsulate the parsing of the common input in a Reader class (the Director).
- Design a standard protocol for creating all possible output representations. Capture the steps of this protocol in a Builder interface.
- Define a Builder derived class for each target representation.
- The client creates a Reader object and a Builder object, and registers the latter with the former.
- The client asks the Reader to "construct".
- The client asks the Builder to return the result.

3. **Singleton: Ensure a class has only one instance, and provide a global point of access to it.**

- Define the constructor as private (or protected))
  - private Singleton(String name)
- Define a private static reference to the single class object
  - static private Singleton instance
- Define a acessor method to that instance
  - static public Singleton getInstance ()

  – Customers can access only the singleton object through this method

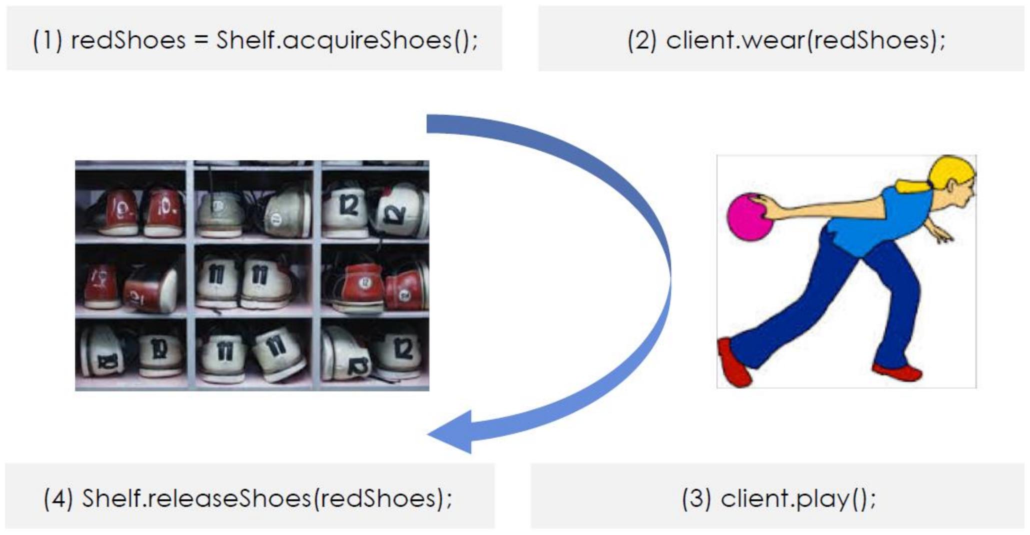


Problem

– Application needs one, and only one, instance of na object. Additionally, lazy initialization and global access are necessary.

- Define a private static attribute in the "single instance" class.
- Define a public static accessor function in the class.
- Do "lazy initialization" (creation on first use) in the accessor function.
- Define all constructors to be protected or private.
- Clients may only use the accessor function to manipulate the Singleton.

4. Object Pool: Object pooling can offer a significant performance boost



```
(1) redShoes = Shelf.acquireShoes();
(2) client.wear(redShoes);
(3) client.play();
(4) Shelf.releaseShoes(redShoes);
```

- Create the Pool class with a collection of PooledObjects
- Create acquire and release methods in Pool class

Problem

– Object are used to manage the object caching. A cliente with access to a Object pool can avoid creating a new Object by simply asking the pool for one that has already been instantiated instead.

– It is desirable to keep all Reusable objects that are not currently in use in the same object pool so that they can be managed by one coherent policy.