**Abstract Factory**

* **Introduction:**
* Abstract factory is used when we have two or more objects which work together forming a kit or set and there can be multiple sets or kits that can be created by client code.
* So we separate client code from concrete objects forming such a set and also form the code which creates these sets.
* **UML:**

**A screenshot of a cell phone

Description automatically generated**

**A picture containing photo, grass, different, sign

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

Role – Abstract Product:

Interface for a type of product.

Role – Concrete Product:

Implements the product interface or class.

Role – Abstract Factory:

Interface defining operations to create products.

Role – Concrete Factory:

Implements factory and creates products from a particular family.

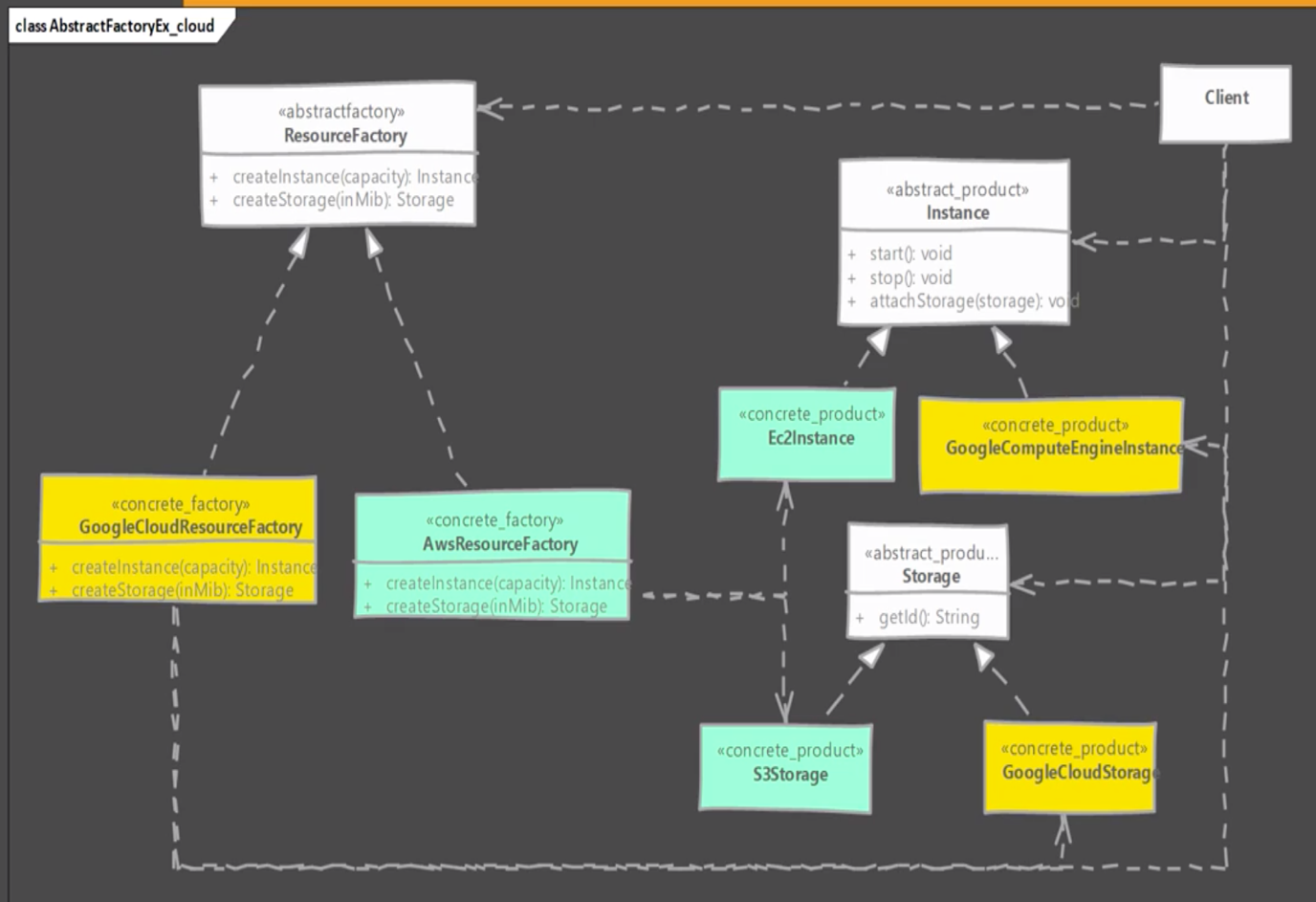
Role – Client:

Uses abstract factory and abstract product.

A screenshot of a cell phone

Description automatically generated

* **Implementation Steps:**
* We start by studying the product “sets”.
  + Create abstract factory as an abstract class or an interface.
  + Abstract factory defines abstract methods for creating products.
  + Provide concrete implementation of factory for each set of products.
* Abstract factory makes use of factory method pattern. You can think of abstract factory as an object with multiple factory methods.
* **Example – UML:**

****

* **Implementation & Design Consideration:**
* Factories can be implemented as singleton, we typically ever need only one instance of it anyway. But make sure to familiarize yourself with drawbacks of singleton.
* Adding a new product type requires changes to the base factory as well as all implementations of factory.
* We provide the client code with concrete factory so that it can create objects.

**Design Consideration:**

* When you want to constrain object creations so that they all work together then abstract factory is good design pattern.
* Abstract factory uses factory method pattern.
* If objects are expensive to create then you can transparently switch factory implementations to use prototype design pattern to create objects.
* **Example:**
* The java.xml.parsers.DocumentBuilderFactory is a good example of abstract factory pattern.
* However this implementation doesn’t match 100% with the UML of abstract factory from GoF. The class has a static newInstance() method which returns actual factory object class.
* The newInstance() method however uses classpath scanning, system properties, an external property file as ways to find the factory class & creates the factory object. So we can change the factory class being used, even if this is a static method.
* **Comparison with Factory Method:**

|  |  |
| --- | --- |
| **Abstract Factory** | **Factory Method** |
| Hides factories as well as concrete objects used from the client code. | Hides the concrete objects which are used from the client code |
| Suitable when multiple objects are designed to work together & client must use products from single family at a time. | Concerned with one product & it’s subclasses. Collaboration of product itself with other object is irrelevant. |

* **Pitfalls:**
* A lot more complex to implement than factory method.
* Adding a new product requires changes to base factories as well as ALL implementations of factory.
* Difficult to visualize the need at start of development and usually starts out as a factory method.
* Abstract Factory design pattern is very specific to the problem of “product families”.
* **Quiz:**

**A screenshot of a social media post

Description automatically generated**

**A screenshot of a social media post

Description automatically generated**

**A screenshot of a social media post

Description automatically generated**