Study of Design Pattern

a **design pattern** is a general repeatable solution to a commonly occurring problem in software **design**.
A **design pattern** isn't a finished **design** that can be transformed directly into code

There are mainly three types of design patterns
1)**Creational**: These design patterns are all about class instantiation or object creation. ...

2)**Structural**: These design patterns are about organizing different classes and objects to form larger structures and provide new functionality. ...

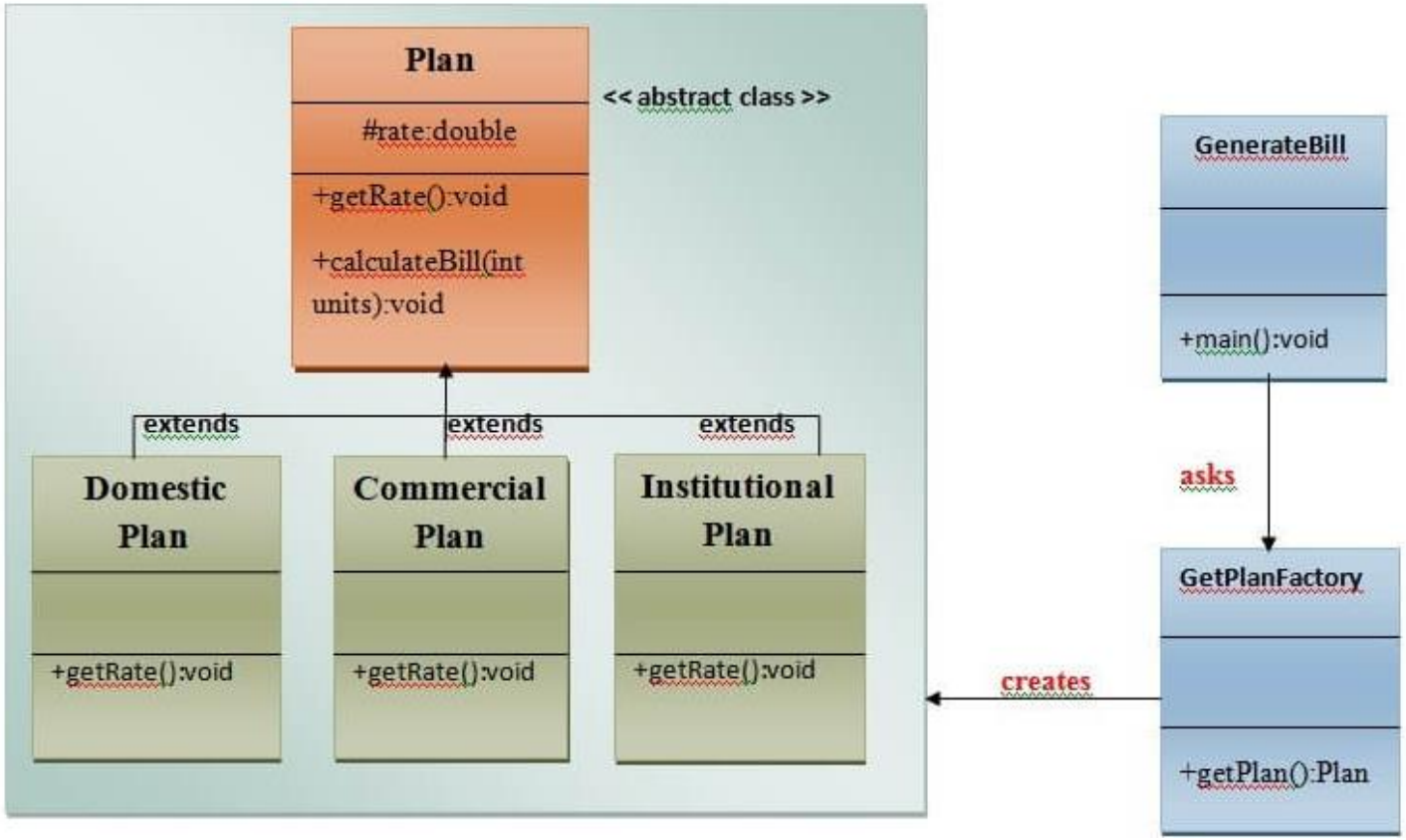
3)**Behavioral**:

Creational Pattern

- Creational patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code. In software engineering.
- Types of creational design patterns
- There are following 6 types of creational design patterns.

1. Factory Method Pattern
2. Abstract Factory Pattern
3. Singleton Pattern
4. Prototype Pattern
5. Builder Pattern
6. Object Pool Pattern

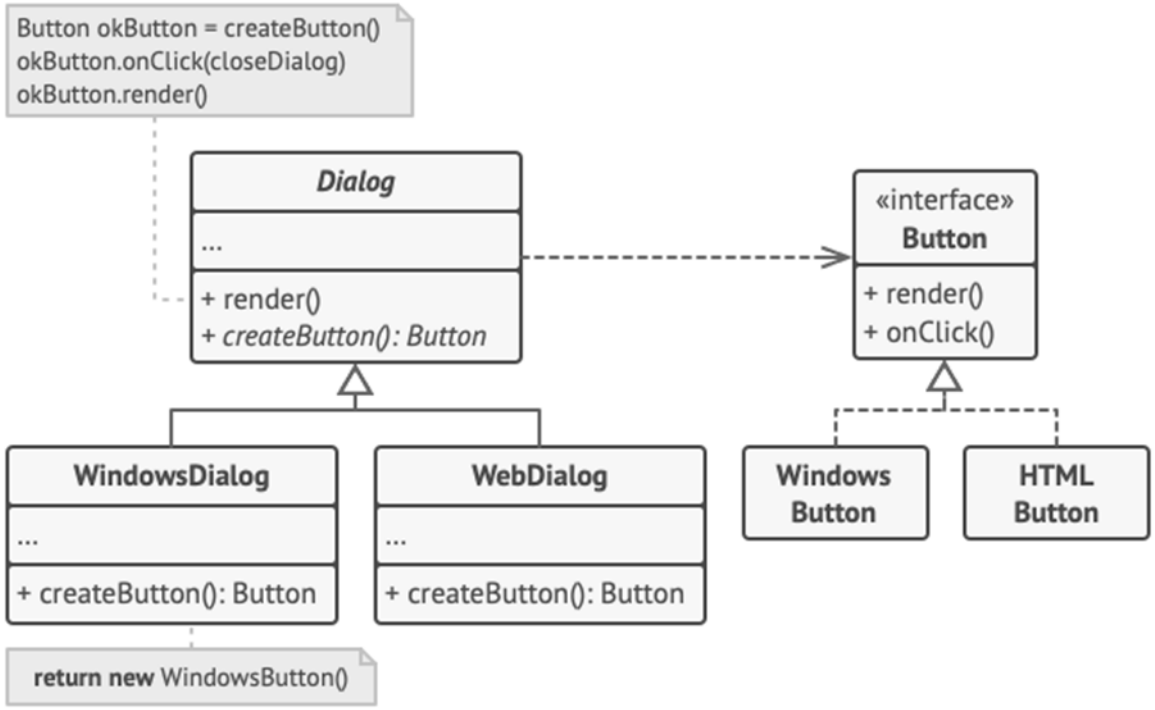
UML for Factory Method Pattern



When to Use Factory Method Design Pattern

- When the implementation of an interface or an abstract class is expected to change frequently
- When the current implementation cannot comfortably accommodate new change
- When the initialization process is relatively simple, and the constructor only requires a handful of parameters

Factory Method Design Pattern



The cross-platform dialog example.

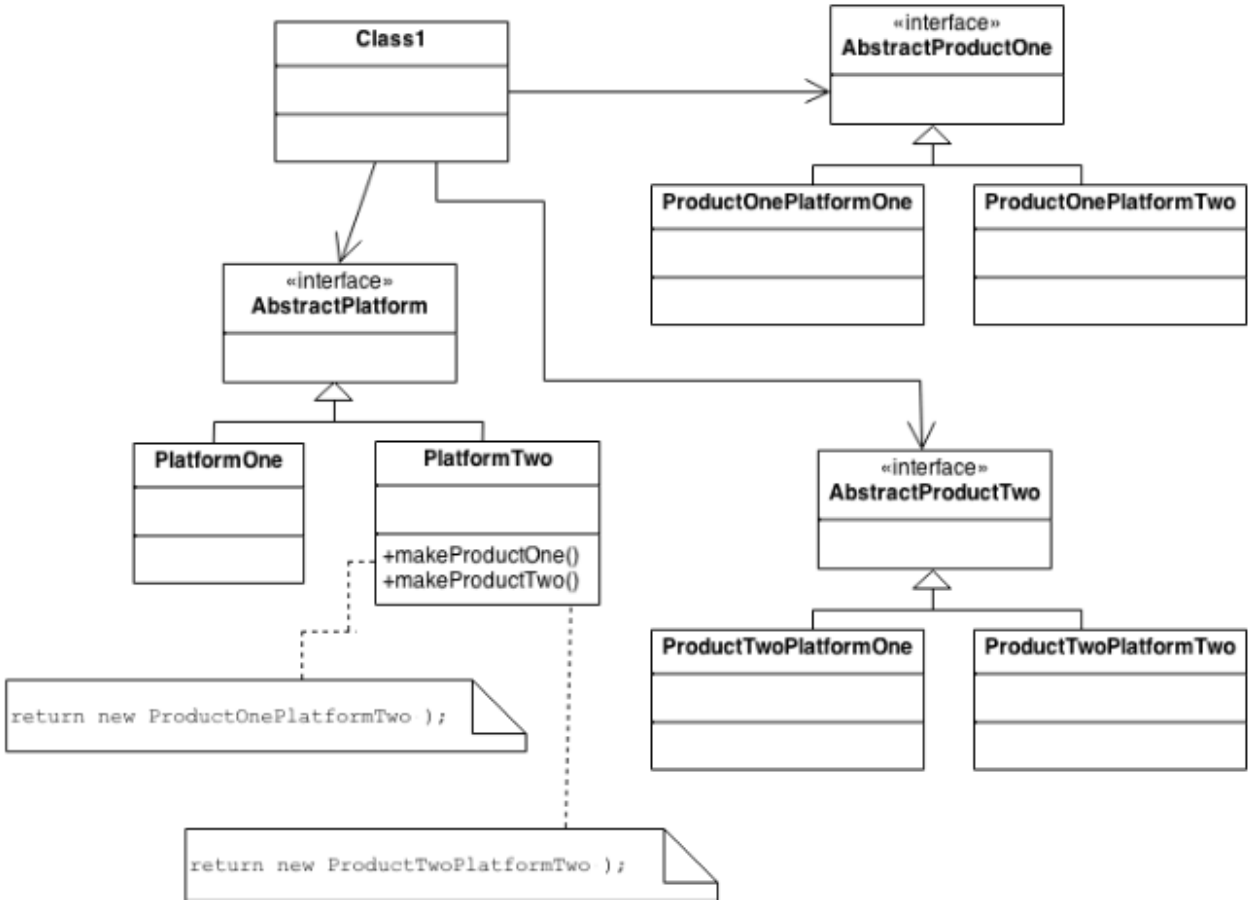
Abstract Factory Design Pattern

- 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 `new` operator considered harmful

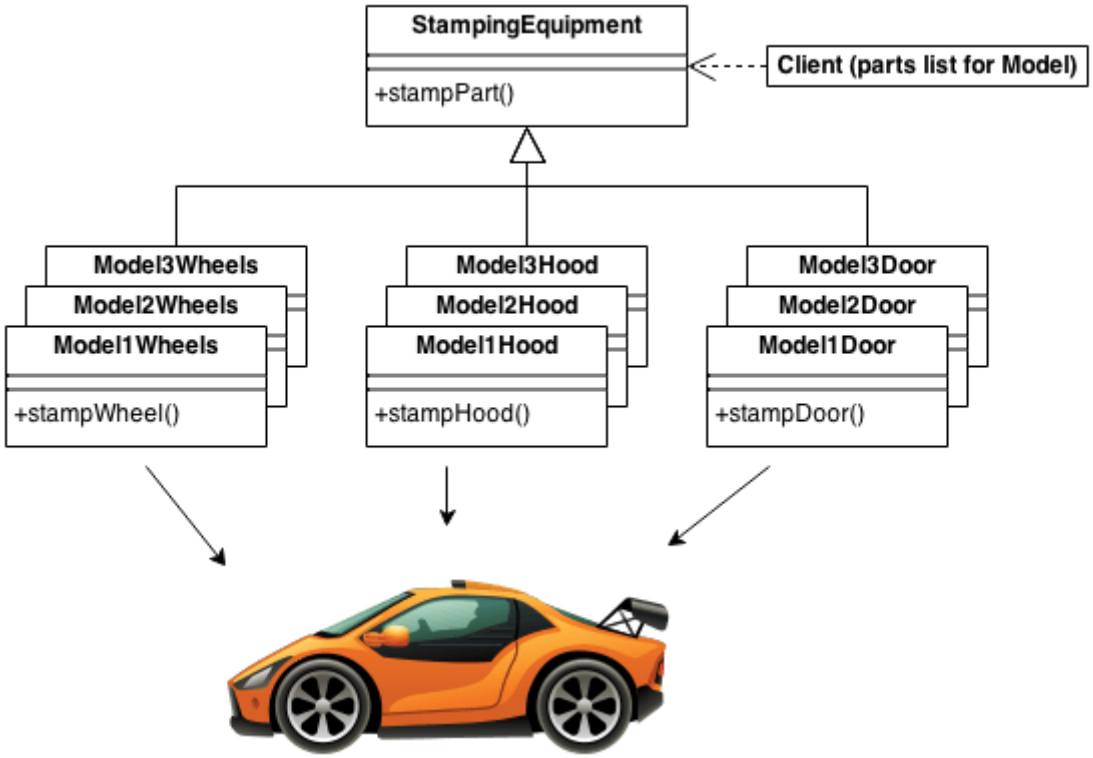
Structure

- `AbstractFactory` — declares an interface for operations that create abstract products.
- `ConcreteFactory` — implements operations to create concrete products.
- `AbstractProduct` — declares an interface for a type of product objects.
- `Product` — defines a product to be created by the corresponding ConcreteFactory; it implements the AbstractProduct interface.
- `Client` — uses the interfaces declared by the AbstractFactory and AbstractProduct classes.

Abstract Factory



Abstract Factory classes



Implementation

- We are going to create a Shape interface and a concrete class implementing it. We create an abstract factory class AbstractFactory as next step. Factory class ShapeFactory is defined, which extends AbstractFactory. A factory creator/generator class FactoryProducer is created.
- AbstractFactoryPatternDemo, our demo class uses FactoryProducer to get a AbstractFactory object. It will pass information (CIRCLE / RECTANGLE / SQUARE for Shape) to AbstractFactory to get the type of object it needs.