# Abstract Factory Pattern

**Jiaxin Wang**

**CSPP 51023**

**Winter 2010**

# Intent

- “Provide an interface for creating families of related or dependent objects without specifying their concrete classes.”
  - provide a simple creational interface for a complex family of classes
    - Client does not have to know any of those details.
  - avoid naming concrete classes
    - Clients use abstract creational interfaces and abstract product interfaces. Concrete classes can be changed without affecting clients.

# Applicability

- Use the Abstract Factory Pattern if:
  - clients need to be ignorant of how servers are created, composed, and represented.
  - clients need to operate with one of several families of products
  - a family of products must be used together, not mixed with products of other families
  - provide a library and want to show just the interface, not implementation of the library components.

# Collaborators

- Usually only one ConcreteFactory instance is used for an activation, matched to a specific application context. It builds a specific product family for client use -- the client doesn’t care which family is used -- it simply needs the services appropriate for the current context.
- The client may use the AbstractFactory interface to initiate creation, or some other agent may use the AbstractFactory on the client’s behalf.
- The factory returns, to its clients, specific product instances bound to the product interface. This is what clients use for all access to the instances.

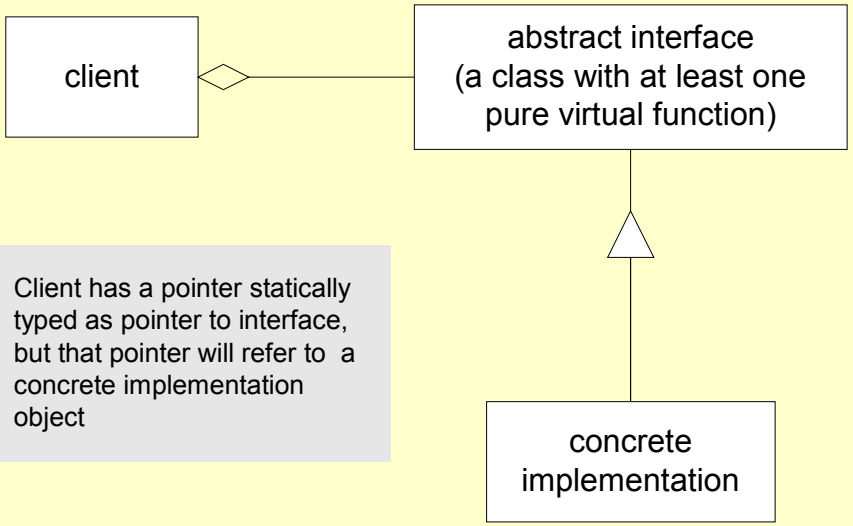
# Consequences

- The Abstract Factory Pattern has the following benefits:
  - It isolates concrete classes from the client.
    - You use the Abstract Factory to control the classes of objects the client creates.
    - Product names are isolated in the implementation of the Concrete Factory, clients use the instances through their abstract interfaces.
  - Exchanging defined product families is easy.
    - None of the client code breaks because the abstract interfaces don't change.
    - Because the abstract factory creates a complete family of products, the whole product family changes when the concrete factory is changed.
  - It promotes consistency among products.
    - It is the concrete factory's job to make sure that the right products are used together.

    It is also easy to replace any implementation of the product interfaces. Just rebuild the library and copy into the client's directory.

# Abstract Interface

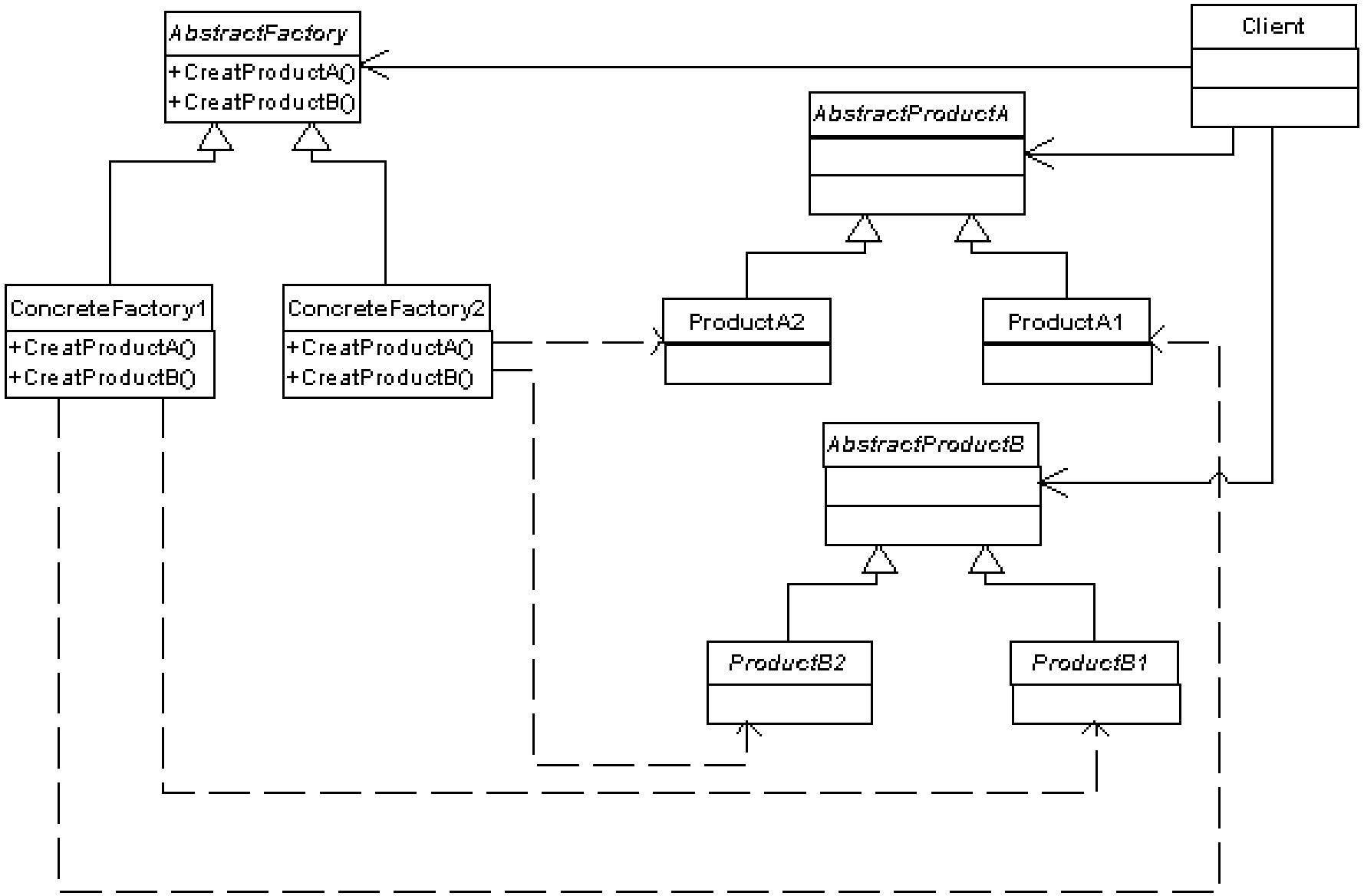
Fact:
This client will be compile-time independent of the concrete implementation if, and only if, it does not directly create an instance of the concrete class

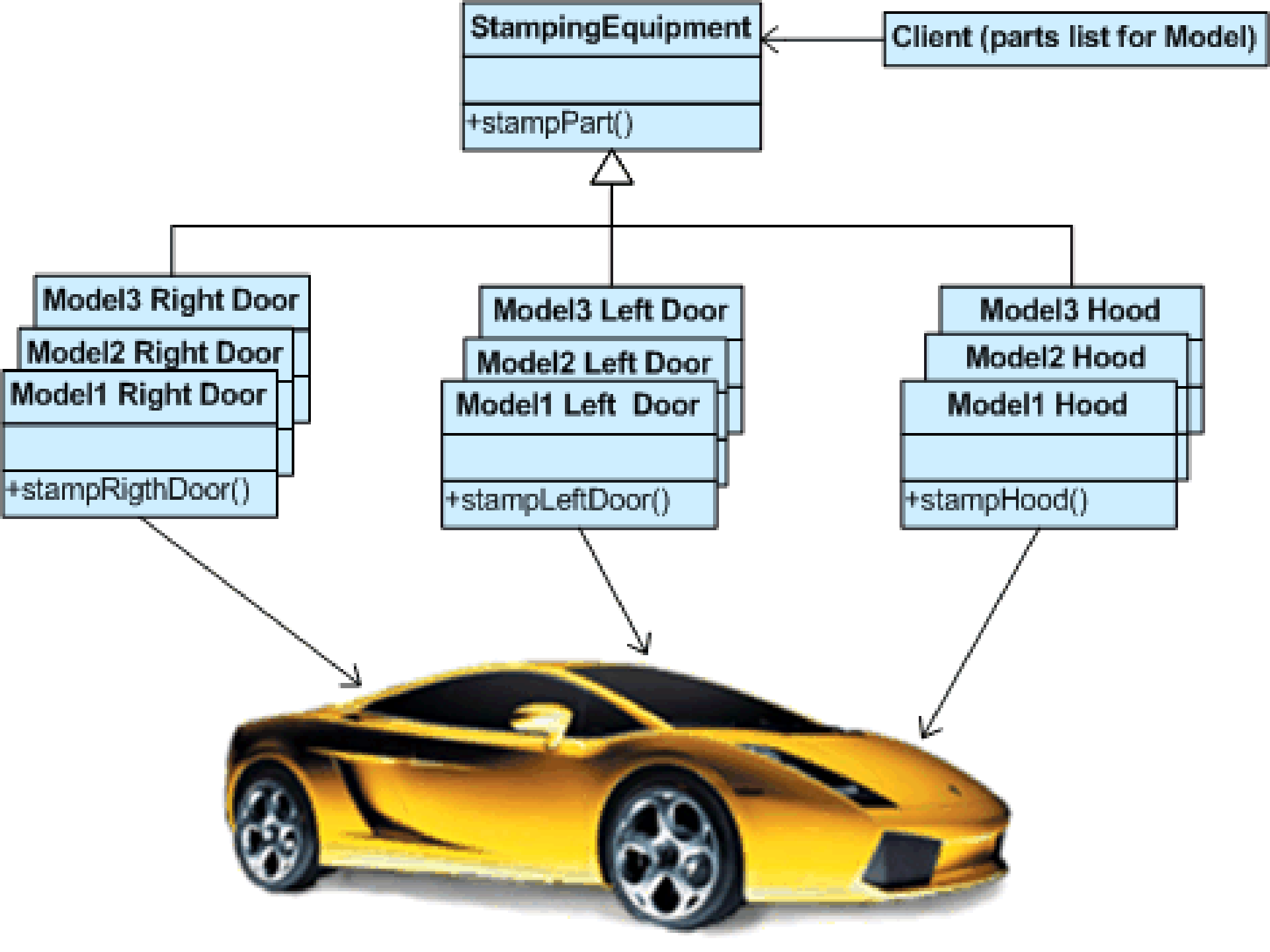



The purpose of an abstract interface is to provide a protocol for clients to use to request service from concrete objects without coupling to their implementations

Client has a pointer statically typed as pointer to interface, but that pointer will refer to a concrete implementation object

# Abstract Factory Structure




StampingEquipment
+stampPart()
Client (parts list for Model)
Model3 Right Door
Model2 Right Door
Model1 Right Door
+stampRigthDoor()
Model3 Left Door
Model2 Left Door
Model1 Left Door
+stampLeftDoor()
Model3 Hood
Model2 Hood
Model1 Hood
+stampHood()

# Implementation example

- Sample sample=new Sample();
- Sample mysample=new MySample();
- Sample hissample=new HisSample();

```
Public class Factory
{
Public static Sample creator(int which)
 {
    if (which==1)
      return new SampleA();
    else if (which==2)
      return new SampleB();
 }
}

Sample sampleA=Factory.creator(1);
```

# Implementation example

```
public abstract class Factory{
        public abstract Sample creator();
        public abstract Sample2 creator(String name); }
        public class SimpleFactory extends Factory{
        public Sample creator(){
        .........
        return new SampleA
        }
        public Sample2 creator(String name){
        .........
        return new Sample2A
        } }
        public class BombFactory extends Factory{
        public Sample creator(){
        ......
        return new SampleB
        }
        public Sample2 creator(String name){
        ......
        return new Sample2B
        } }
```