Design Patterns

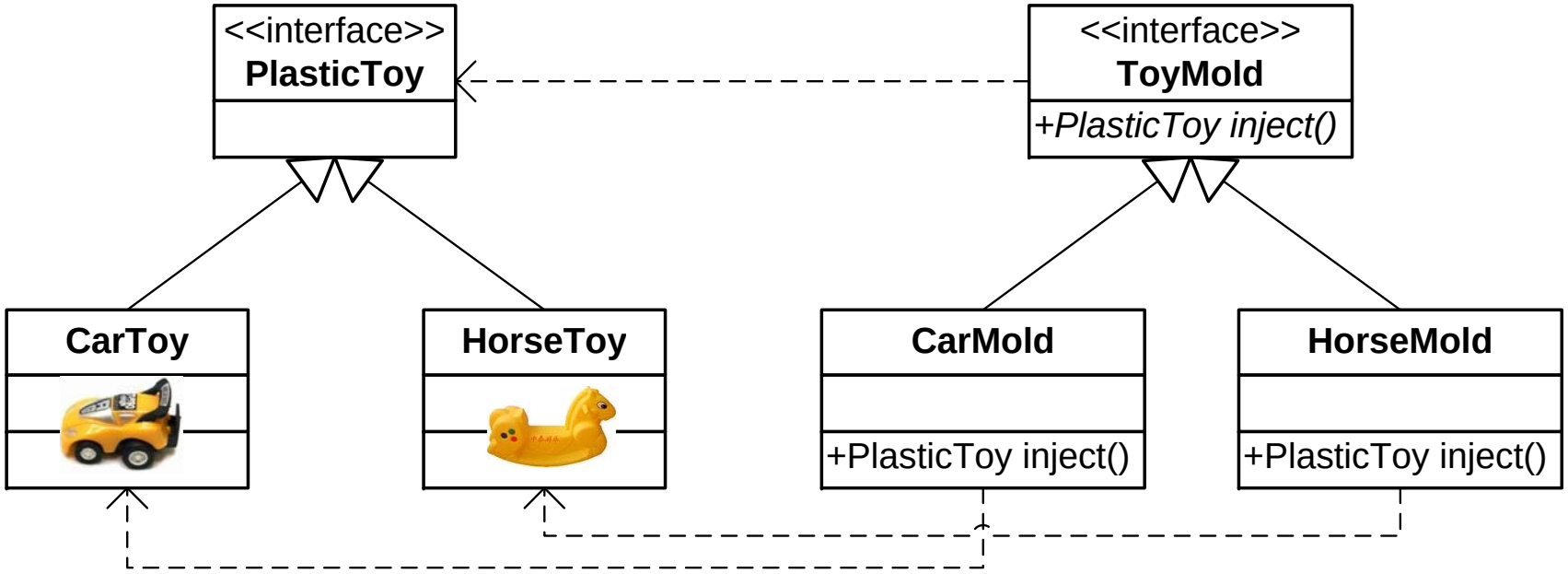
宋 杰
Song Jie
东北大学 软件学院
Software College, Northeastern University

1. Factory Method Pattern

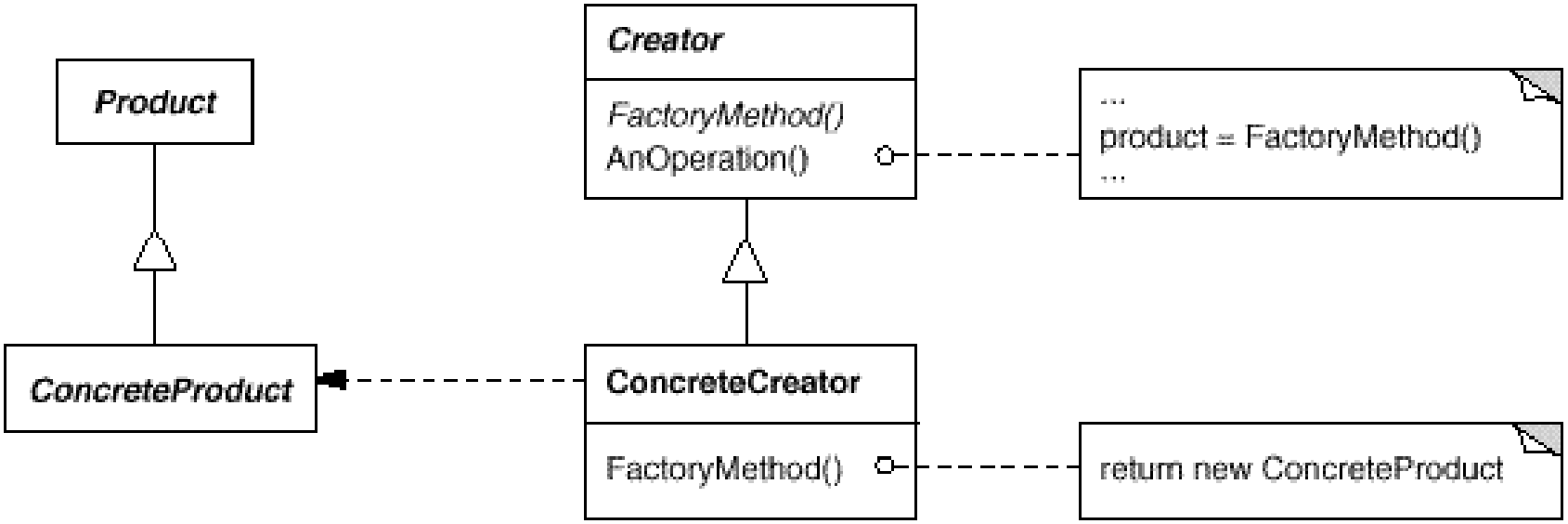
Intent

- Define an interface for creating an object, but let subclasses decide which class to instantiate. [in'stænʃieit]
  - Why need a method to create an object;
  - Why use interface?
  - The factory method return what kind of object?
  - Why let subclasses to decide the concrete class?

Intent


<<interface>>
PlasticToy
<<interface>>
ToyMold
+PlasticToy inject()
CarToy
HorseToy
CarMold
+PlasticToy inject()
HorseMold
+PlasticToy inject()

Structure


Product
Creator
FactoryMethod()
AnOperation()
...
product = FactoryMethod()
...
ConcreteProduct
ConcreteCreator
FactoryMethod()
return new ConcreteProduct

Participants

- Product: defines the interface of objects the factory creates.
- ConcreteProduct: implements the Product interface.
- Creator: declares the factory method, which returns an object of type Product.
  - Creator may also define a default implementation of the factory method that returns a default ConcreteProduct object.
- ConcreteCreator: overrides the factory method to return an instance of a ConcreteProduct, referred by Product.

Codes – Product

```
interface Product {
}

class DefaultProduct implements Product {
}

class ProductA implements Product {
}

class ProductB implements Product {
}
```

Codes – Factory Method (version 1)

```
interface Factory {

    public Product createProduct(String type);
}

class ConcreteFactory implements Factory {
    public Product createProduct(String type){
        if (type.equals("A")) {
            return new ProductA();
        } else if (type.equals("B")){
            return new ProductB();
        }else{
            return new DefaultProduct();
        }
    }
}
```

Codes – Factory Method (version 2)

```
interface Factory {

    public Product createProductA();
    public Product createProductB();
    public Product createProduct();

}

class ConcreteFactory implements Factory {

    public Product createProductA() {
        return new ProductA();
    }
    public Product createProductB() {
        return new ProductB();
    }
    public Product createProduct() {
        return new DefaultProduct();
    }
}
```

Codes - Client

```
class Client{
    public static void main(String[] args) {
        Factory factory = new ConcreteFactory();
        Product productA = factory.createProductA();
        Product productB = factory.createProductB();
    }
}
```

Consequences

- Factory methods eliminate the need to bind application-specific classes into your code.
- The code only deals with the Product interface; therefore it can work with any user-defined ConcreteProduct classes.
- Creating objects inside a class with a factory method is always more flexible than creating an object directly.

Applicability

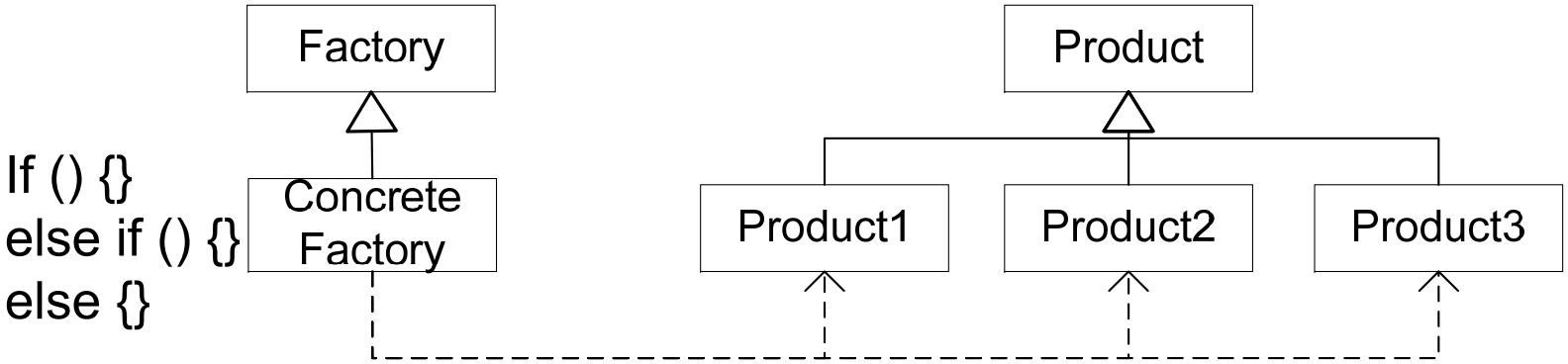
- The concrete products are required not to exposed to clients;
- Creator can't decide the concrete product it must create;
- Creator wants its subclasses to specify the product it creates.
- Creator delegate the responsibility of creating instance to one of several subclasses.

Implementation 1:
Factory methods in pure factory class or not

- Factory method can be contained in a business class which performed some business logic.
  - It always including the logic that using the products.
- Factory method can be contained in a pure factory class which is responsible for nothing but creating products.

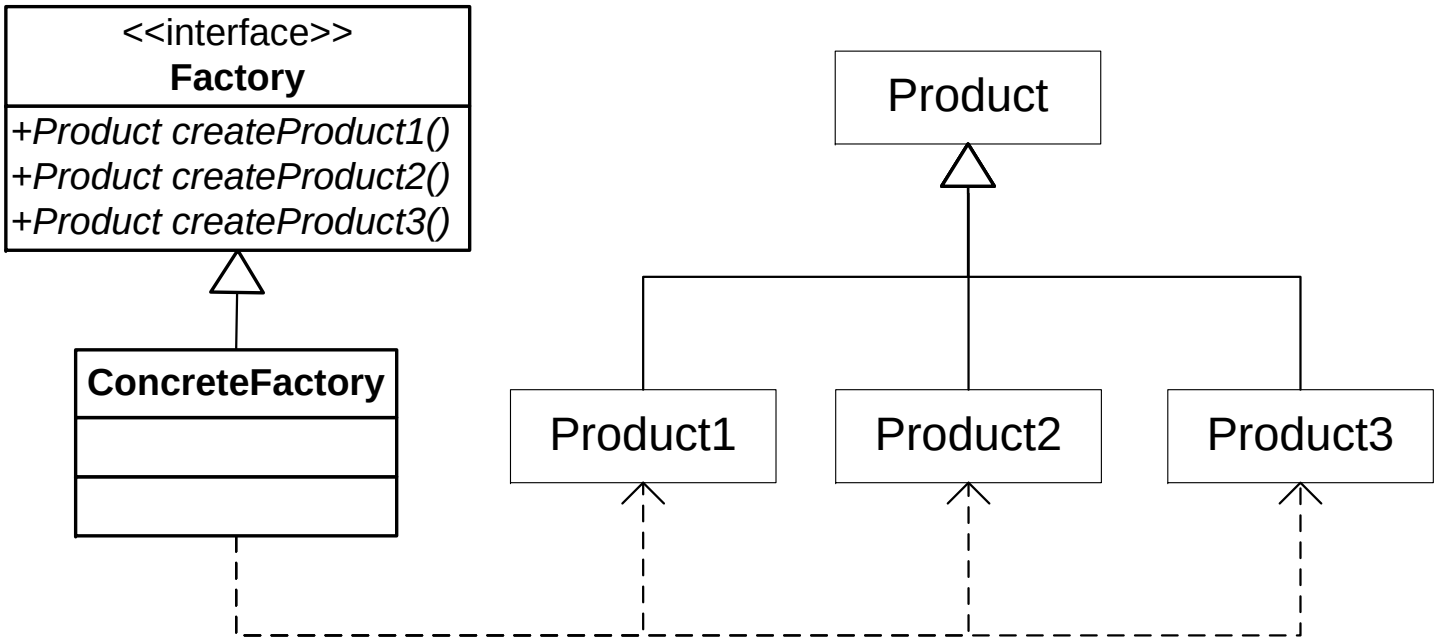
Implementation 2: Parameterized factory methods

- Factory method create multiple kinds of products by taking a parameter that identifies the kind of object to create.



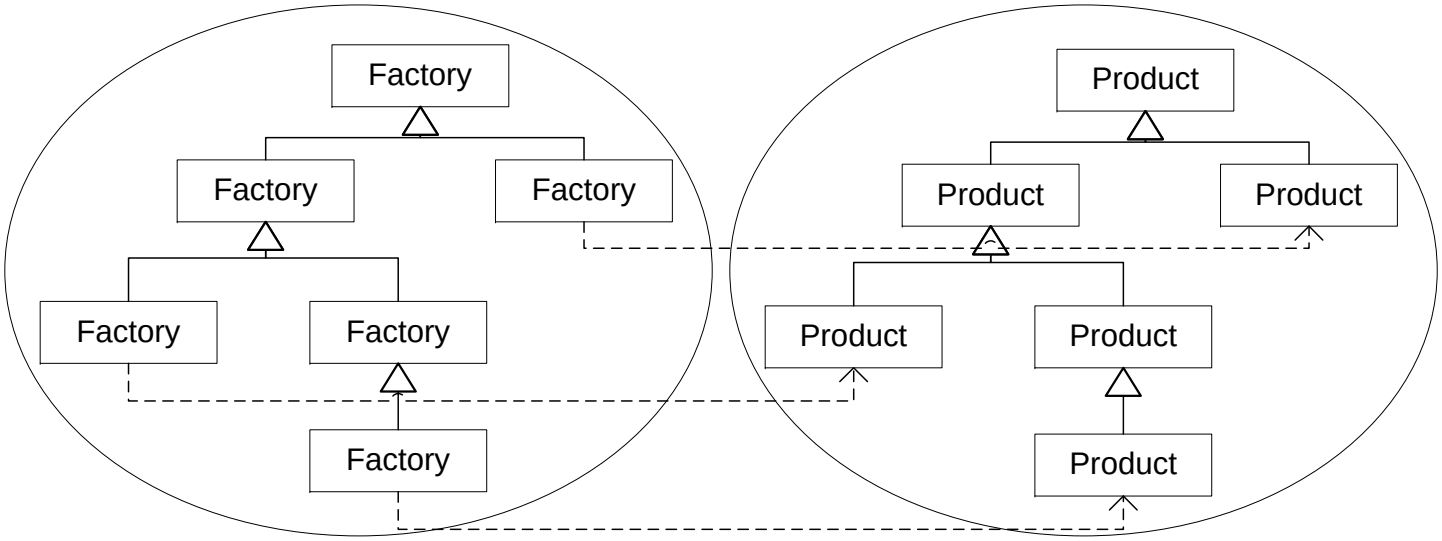
Implementation 3: Parallel factory methods

- A group of factory methods create multiple kinds of products by different methods signatures.



Implementation 4: Parallel class in inherited hierarchies

- Generally, there are many hierarchical products to be created by hierarchical factory. The factory and corresponded product a in same level of inherited hierarchy.

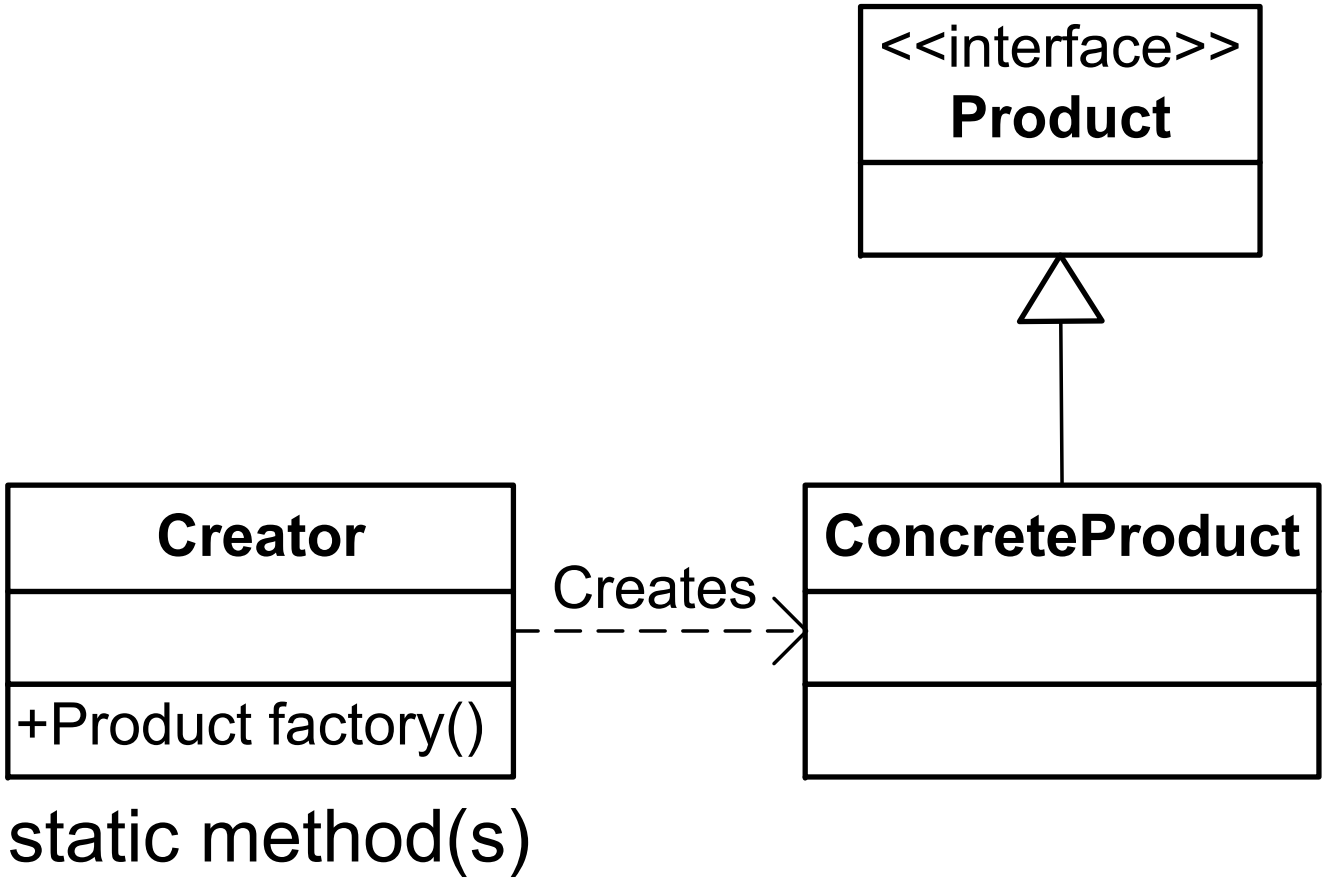


Implementation 5:

Default product providing by default factory

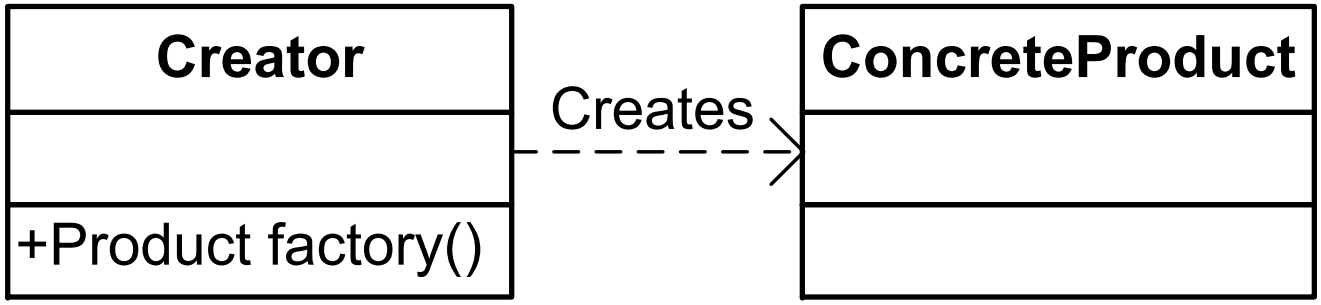
- Creator is an abstract class or interface and does not provide an implementation for the factory method it declares, **OR**
- Creator is a concrete class and provides a default implementation for the factory method. **OR**
- An abstract creator that defines a default implementation providing default product .

Variation 1: Abstract Factory is omitted (Simple Factory Pattern)



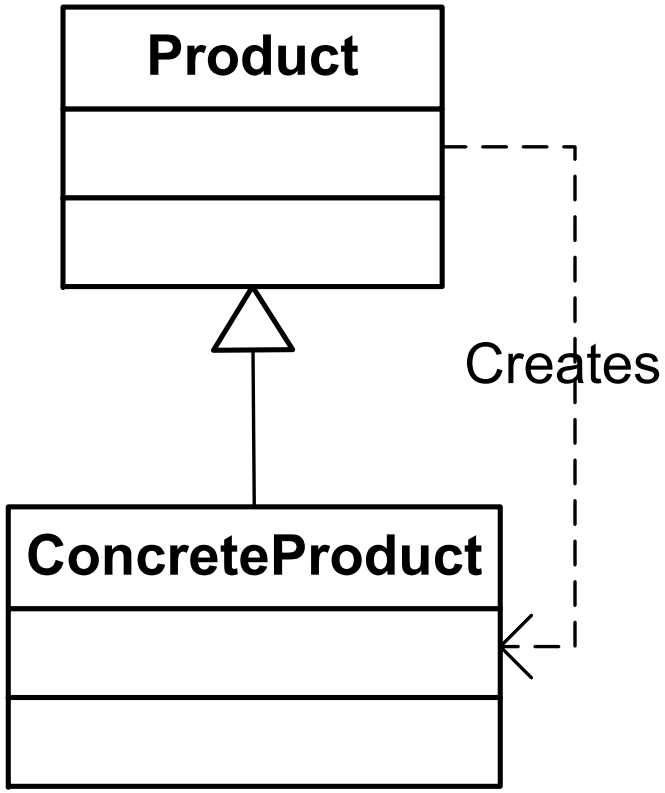
Variation 2: Abstract Factory and Product are omitted

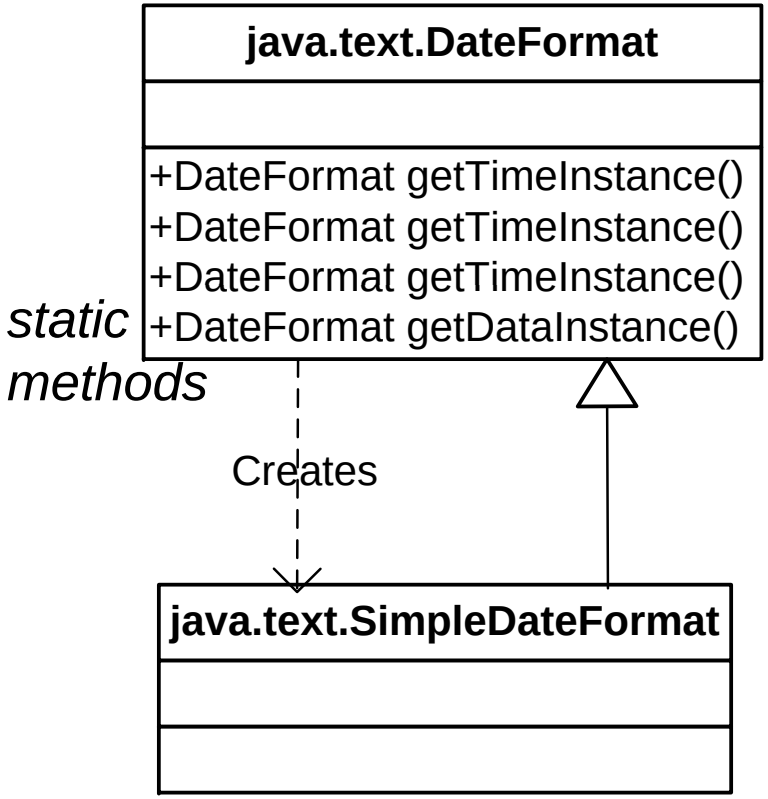
- If products are “unrelated”



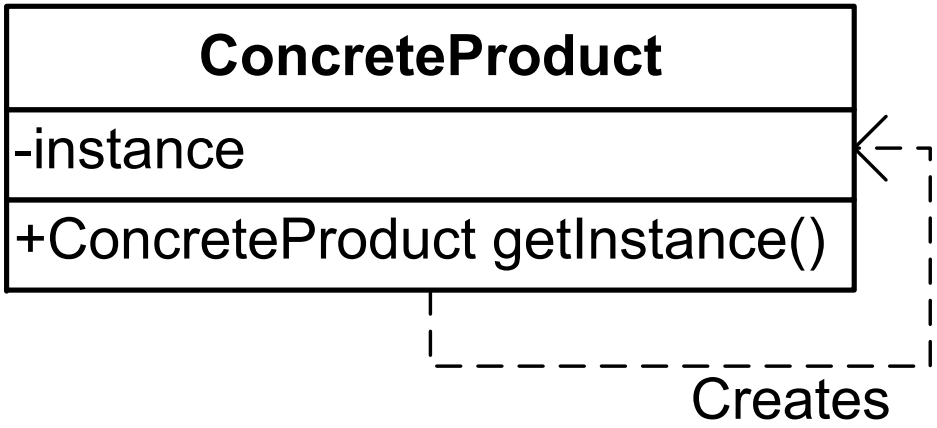
static method(s)

Variation 3: Products contains Factory method to create itself





Variation 4: Concrete Product creates itself



Variation 5: Factory with registered instances (products) for instance reusing

- Factory store the created instances in an registered pool;
- Reusing the registered instance when required;
- Further more, the instance can not only be create by “new”, but also initialized from other resources, for example from database.

Example: JDBC

```
Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
String url="jdbc:oracle:thin:@localhost:1521:orcl";
//orcl为数据库的SID
String user="test";
String password="test";
Connection conn= DriverManager.getConnection(url,user,password);
Statement stmt = conn.createStatement();
```

Example: JNDI and EJB

```
//JNDI naming context
Context ctx = new InitalContext();
//find EJB instance,factory method pattern
EmplyeeHome home = (EmplyeeHome)ctx.lookup("Emplyee");
//factory method pattern
Emplyee emp = home.create(1001,"Song","Jie");
emp.setTel("13940348888");
```

Let’s go to next…