Simple Factory

Design Patterns in Java

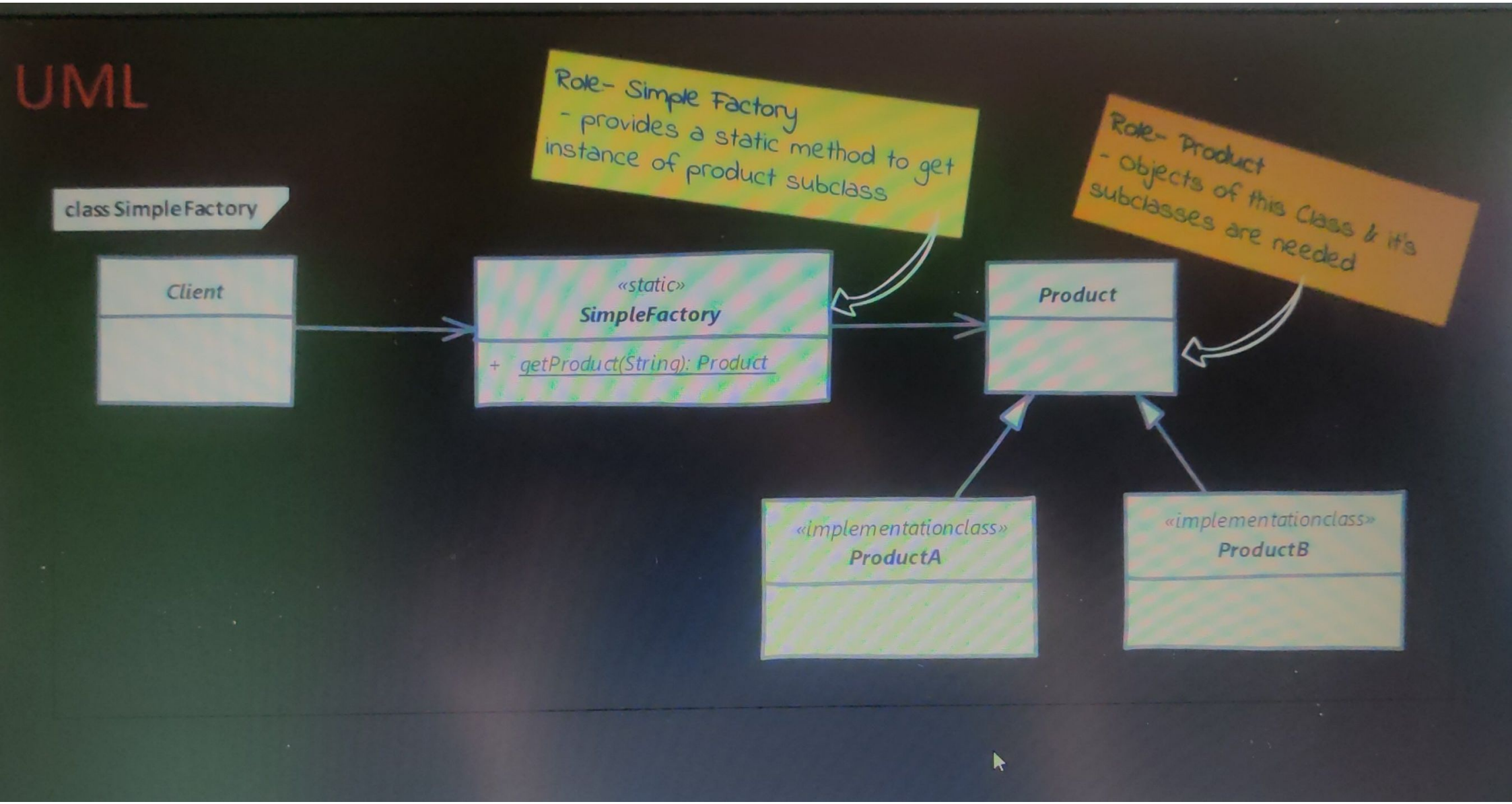
What problem simply factory solves?

Multiple types can be instantiated and the choice is based on some simple criteria

```
if (key.equalsIgnoreCase("pudding")) {

        //create pudding object

} else if (key.equalsIgnoreCase("cake")){

        //create cake object

}
```

What is a Simple Factory?

- Here we simply move the instantiation logic to a separate class and most commonly to a static method of this class.
- Some do not consider simple factory to be a "design pattern", as its simply a method that encapsulates object instantiation. Nothing complex goes on in that method.
  - ❖ *We are studying simple factory as it is often confused with "factory method" pattern*
- Typically we want to do this if we have more than one option when instantiating object and a simple logic is used to choose correct class.


UML
Role- Simple Factory
- provides a static method to get instance of product subclass
Role- Product
- objects of this Class & it's subclasses are needed
class SimpleFactory
Client
«static»
SimpleFactory
+ getProduct(String): Product
Product
«implementationclass»
ProductA
«implementationclass»
ProductB

Implement a Simple Factory

- We start by creating a separate class for our simple factory
  - Add a method which returns desired object instance.
    - This method is typically static and will accept some argument to decide which class to instantiate
    - You can also provide additional arguments which will be used to instantiate objects

Implementation Considerations

- Simple factory can be just a method in existing class. Adding a separate class however allows other parts of your code to use simple factory more easily.
- Simple factory itself doesn't need any state tracking so it's best to keep this as a static method.

Design Considerations

- Simple factory will in turn may use other design pattern like builder to construct objects.
- In case you want to specialize your simple factory in sub classes, you need factory method design pattern instead.

Example of a Simple Factory

- The java.text.NumberFormat class has getInstance method, which is an example of simple factory.

```
private static NumberFormat getInstance(LocaleProviderAdapter adapter,
                                        Locale locale, int choice) {
    NumberFormatProvider provider = adapter.getNumberF          Provider();
    NumberFormat numberFormat = null;
    switch (choice) {
    case NUMBERSTYLE:
        numberFormat = provider.getNumberInstance(locale);
        break;
    case PERCENTSTYLE:
        numberFormat = provider.getPercentInstance(locale);
        break;
    case CURRENCYSTYLE:
        numberFormat = provider.getCurrencyInstance(locale);
        break;
    case INTEGERSTYLE:
        numberFormat = provider.getIntegerInstance(locale);
        break;
    }
    return numberFormat;
}
```

Code from NumberFormat.class in rt.jar

Compare & Contrast with Factory Method Pattern

Simple Factory

- We simply move our instantiation logic away from client code. Typically in a static method.
- Simple factory knows about all classes whose objects it can create.

Factory Method

- Factory method is more useful when you want to delegate object creation to subclasses.
- In Factory method we don't know in advance about all product subclasses.

Pitfalls

- The criteria used by simple factory to decide which object to instantiate can get more convoluted/complex over time. If you find yourself in such situation then use factory method design pattern.

In-A-Hurry Summary

- Simple factory encapsulates away the object instantiation in a separate method.
- We can pass an argument to this method to indicate product type and/or additional arguments to help create objects