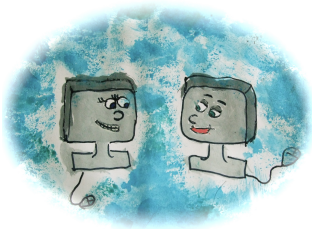
Design Patterns & Refactoring

Factory Methods

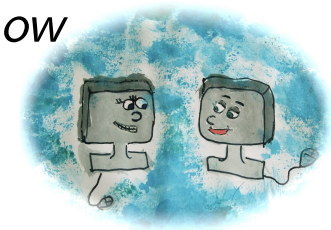
Oliver Haase

HTWG Konstanz

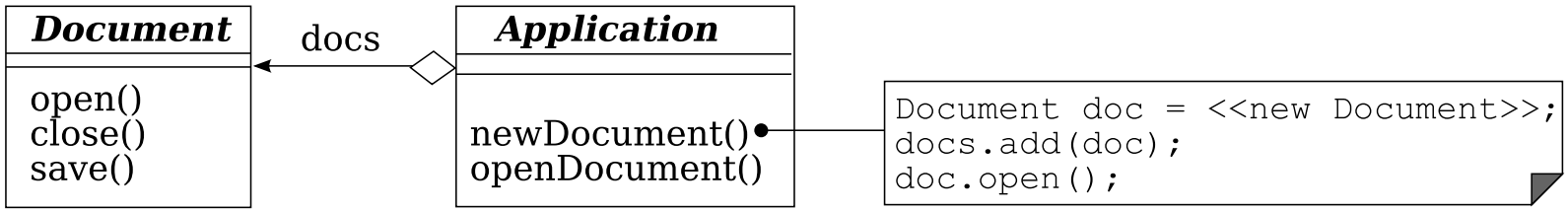


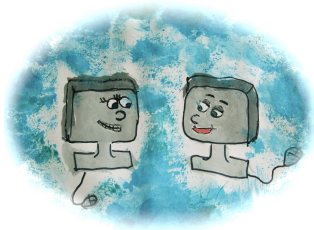
Description

- **Classification**: class based creational pattern
- **Purpose**: Define method for object creation in abstract class, leave actual creation to concrete subclasses.
- **Also known as**: virtual constructor
- **Application Example**: Framework for document management system
  - contains abstract classes `Application` and `Document`
  - `Application` implements business logic
  - `Document` implements common properties and functionality of all document types
  - `Application` knows, *when* to create documents but not *how*

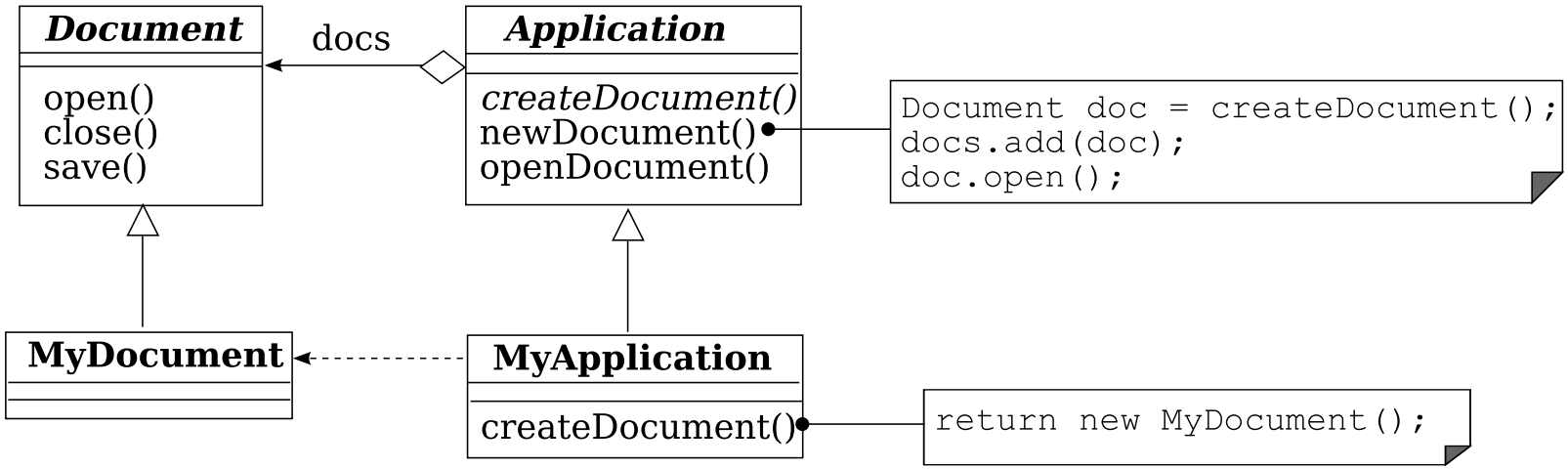


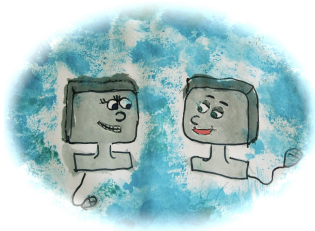
Structure — using an Example



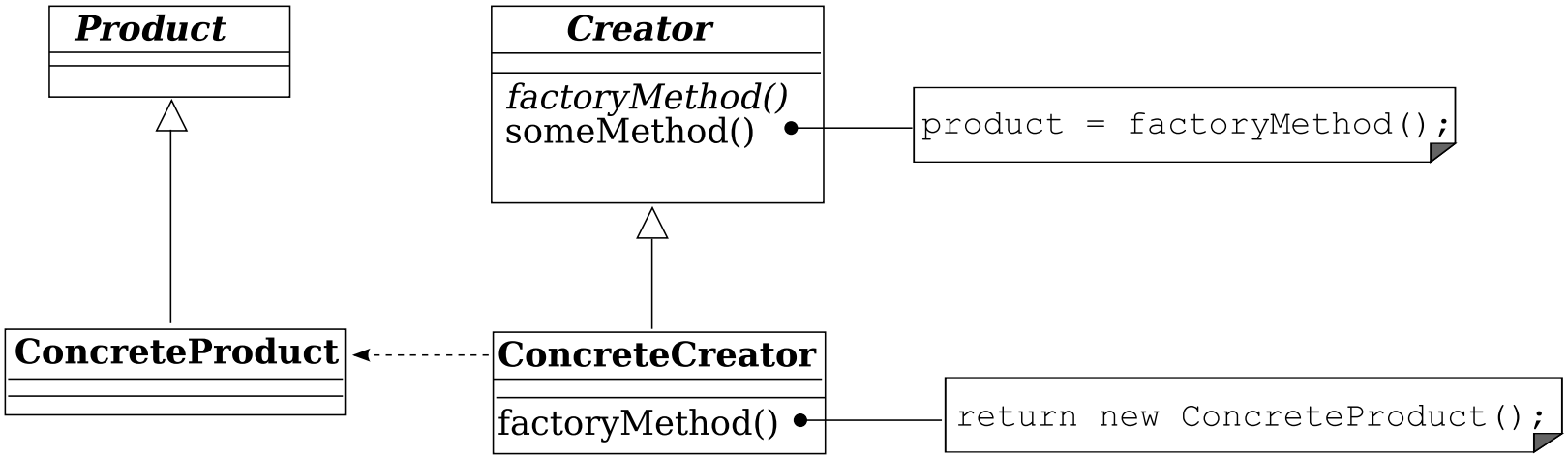


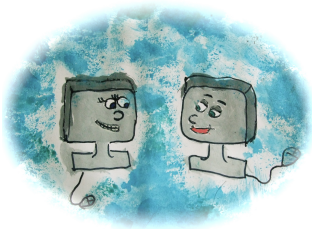
Structure — using an Example





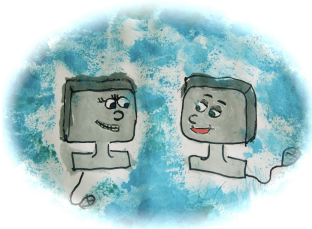
General Structure





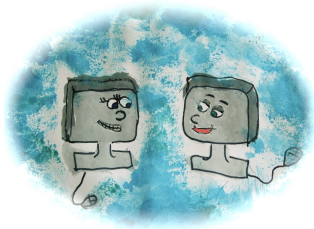
Description II

- **Members**:
  - *Product*: base class / interface of the objects to be created
  - *ConcreteProduct*: concrete products
  - *Creator*:
    - defines factory method
    - may provide default implementation that creates certain concrete product
    - uses factory method, in order to create product objects
  - *ConcreteCreator*: overrides / implements factory method, such that concrete product is created.



Description III

- **Potential Drawback**: Clients (users of the framework) might have to extend *Creator* class only to define factory method
- **Relationship with other patterns**: Abstract factories often use factory methods that are overriden by the concrete factories.

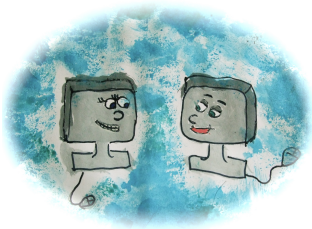


Additional Considerations

In Java, generic types and reflection can be used to build generic factory methods

$\longrightarrow$ *Creator* gets parameterized with concrete product class

$\longrightarrow$ *Creator* needs not be extended.



Additonal Considerations

```
public class GenericCreator <T extends Product> {
  private Class<T> type;
  public GenericCreator(Class<T> type) {
    this.type = type;
  }
  private T createProduct() throws Exception {
    return type.newInstance();
  }
  public void doProduct() throws Exception {
    Product product = createProduct();
    product.use();
  }
  public static void main(String[] args)
      throws Exception {
    GenericCreator<ConcreteProduct> creator =
        new GenericCreator<ConcreteProduct>
            (ConcreteProduct.class);
    creator.doProduct();
  }
}
```