Abstract Factory

A creational pattern

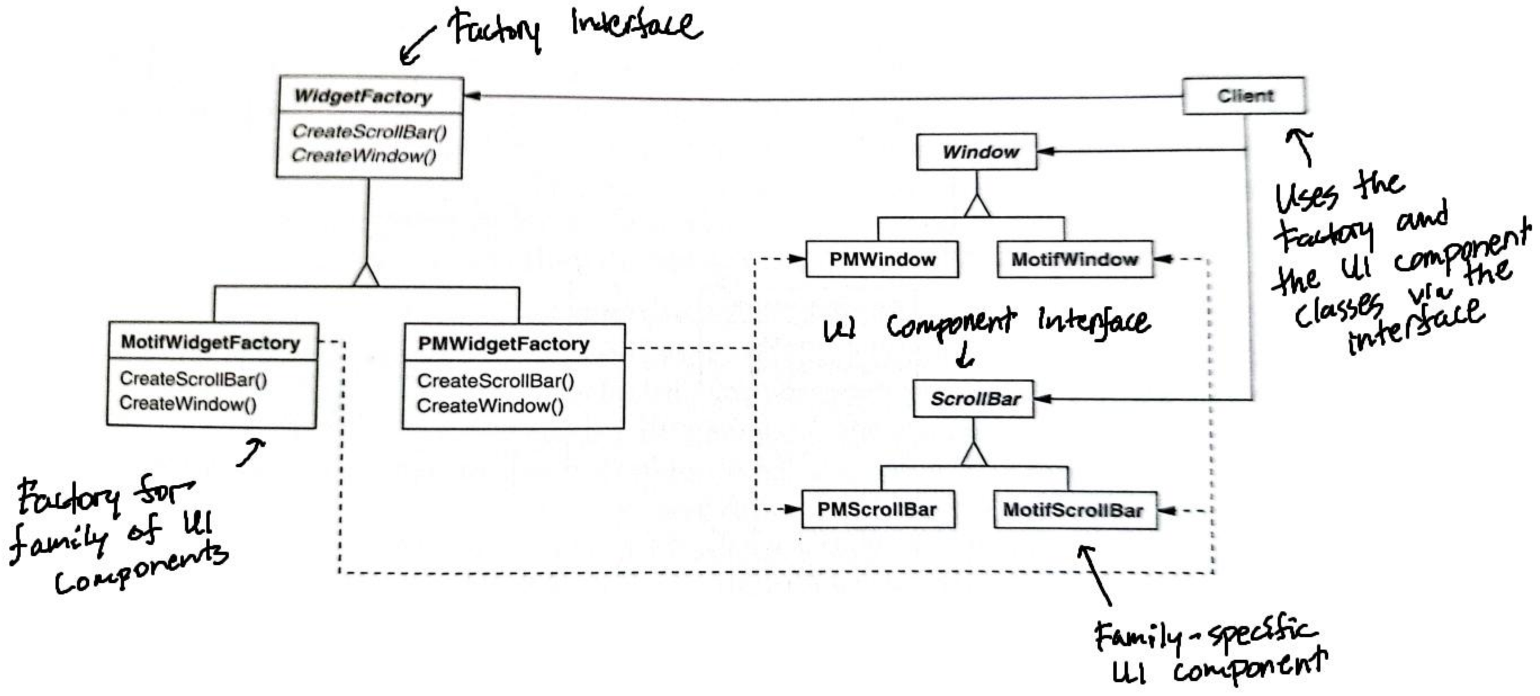
Learning goals

1. Learn the idea, structure, and Java implementation of the Abstract Factory design pattern.
2. Learn to apply the Abstract Factory DP in your own programming.

Idea of Abstract Factory

- The Abstract Factory design patterns helps create **families of related products**.
- Examples of product families:
  - The UI components must reflect the chosen ”look and feel” and style guidelines of the environment.
  - A game may have different themes. The charaters and other displayed objects follow the chosen theme.
- The DP allows the application to choose the product family in one statement.
- It becomes straightforward to add new product families.
- The compatibility of created products is guaranteed.

Example explained



- The application (client) can use UI components of ’Motif’ or ’PM’ family.
- All components (e.g. Window and ScrollBar) must be of the same family.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 87

General structure

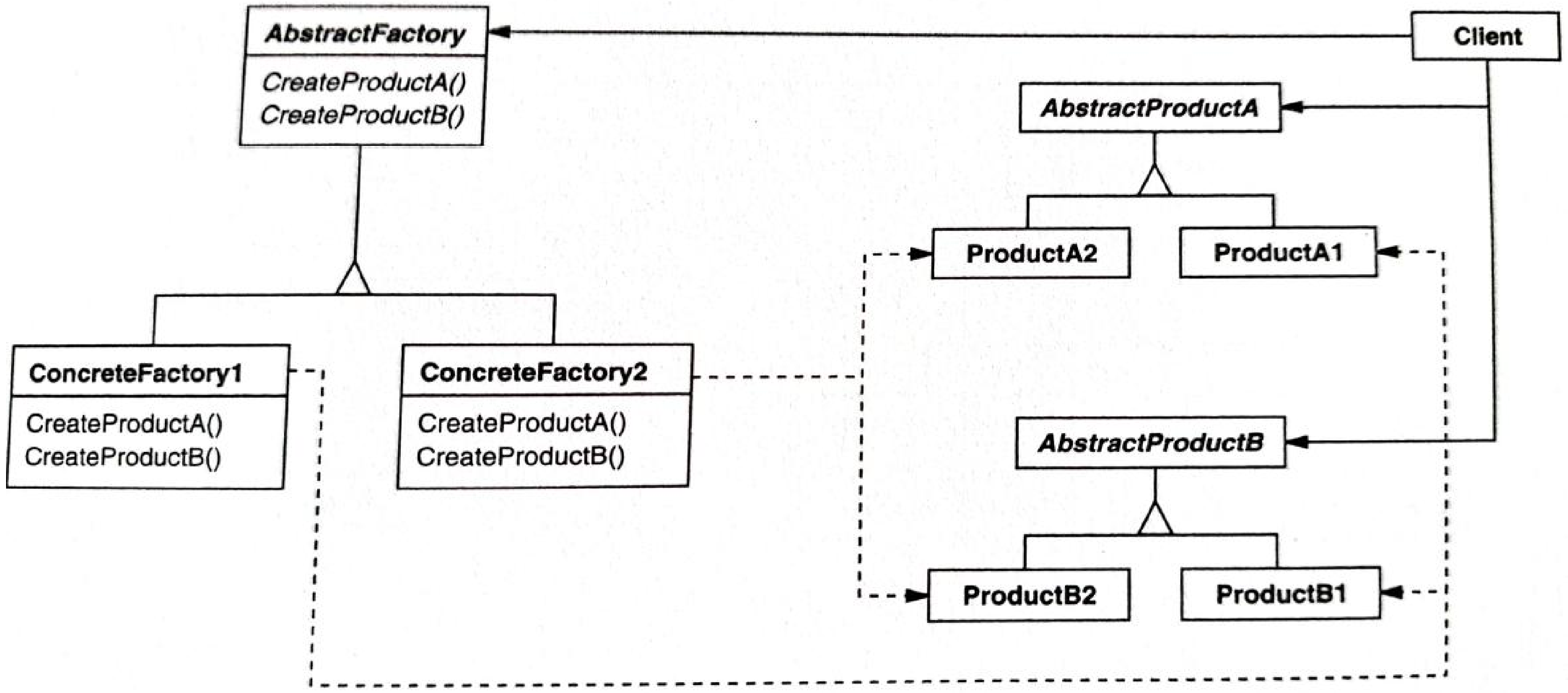


Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 88

Roles

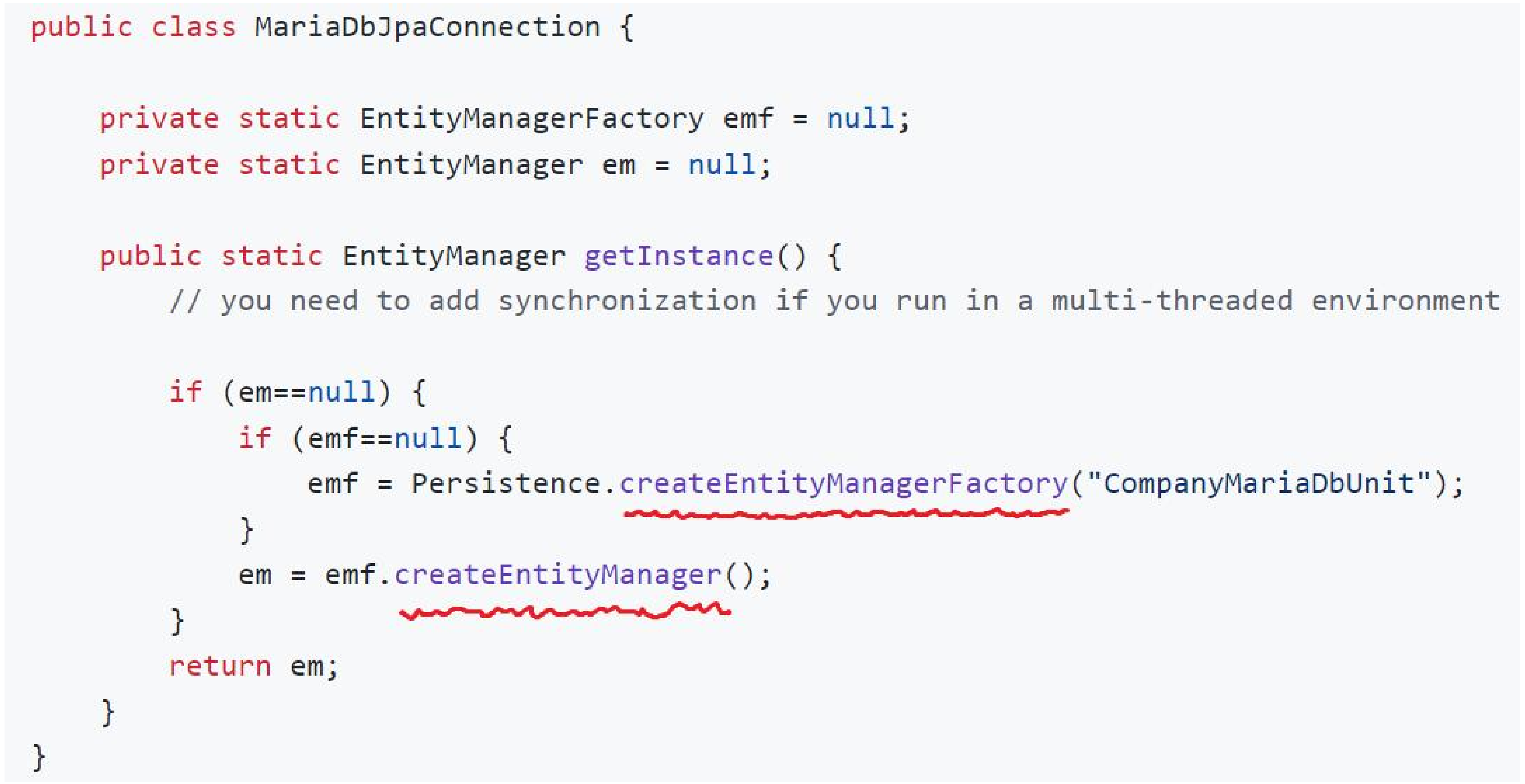
- **Abstract factory**: declares the interface for operations that create abstract products.
  - Can be an interface or an abstract class
- **Concrete factory**: implements the creation of the concrete products in the chosen family.
- **Abstract product**: declares the interface for operations that the concrete products must provide.
  - Can be an interface of an abstract class
- **Concrete product**: defines the implementation of the abstract product. That is, implements the family-specific product.
- **Client** creates products via Abstract factory interface and uses them via Abstract product interface.

Case: JPA Entity Manager

From earlier studies, you may be familiar with JPA (Jakarta Persistence API).
  - It is an ORM framework for persisting Java objects.

- In Java SE context, both the EntityManagerFactory and the EntityManager objects are obtained from a factory method.
  - The concrete class of EntityManager can be different fro different DBMS (such as MariaDB or PostgresSQL)
  - Note that both EntityManager and EntityManagerFactory are interfaces, not concrete classes.

An example use case



```java
public class MariaDbJpaConnection {

    private static EntityManagerFactory emf = null;
    private static EntityManager em = null;

    public static EntityManager getInstance() {
        // you need to add synchronization if you run in a multi-threaded environment

        if (em==null) {
            if (emf==null) {
                emf = Persistence.createEntityManagerFactory("CompanyMariaDbUnit");
            }
            em = emf.createEntityManager();
        }
        return em;
    }
}
```