Factory Method

A creational pattern

Learning goals

1. Understand the idea of creational patterns.
2. Learn the idea, structure, and Java implementation of the Factory Method design pattern.
3. Learn to apply the Factory Method DP in your own programming.

Creational patterns

- **Creational patterns** are one of the three groups of design patterns.
  - The other groups are structural and behavioral.
- The purpose is to **outsource the creation** of objects.
  - The construction responsibility is taken away from the classes that utilize the objects.
  - The creation can be delegated to subclasses or other objects.
- Figuratively speaking, one tries to get rid of sloppy use of **`new`** statements here and there.

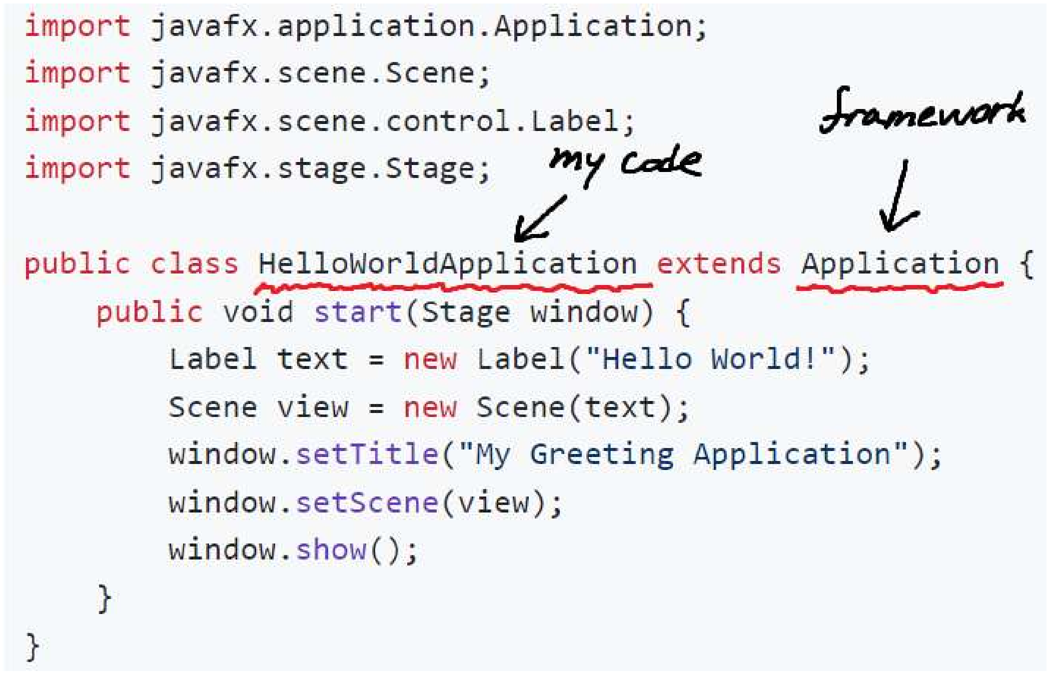
Framework approach

- The Factory Method DP helps design and use **frameworks**.
- The definition of a framework according to Wikipedia:

  "In computer programming, a software framework is an abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software."

Framework approach

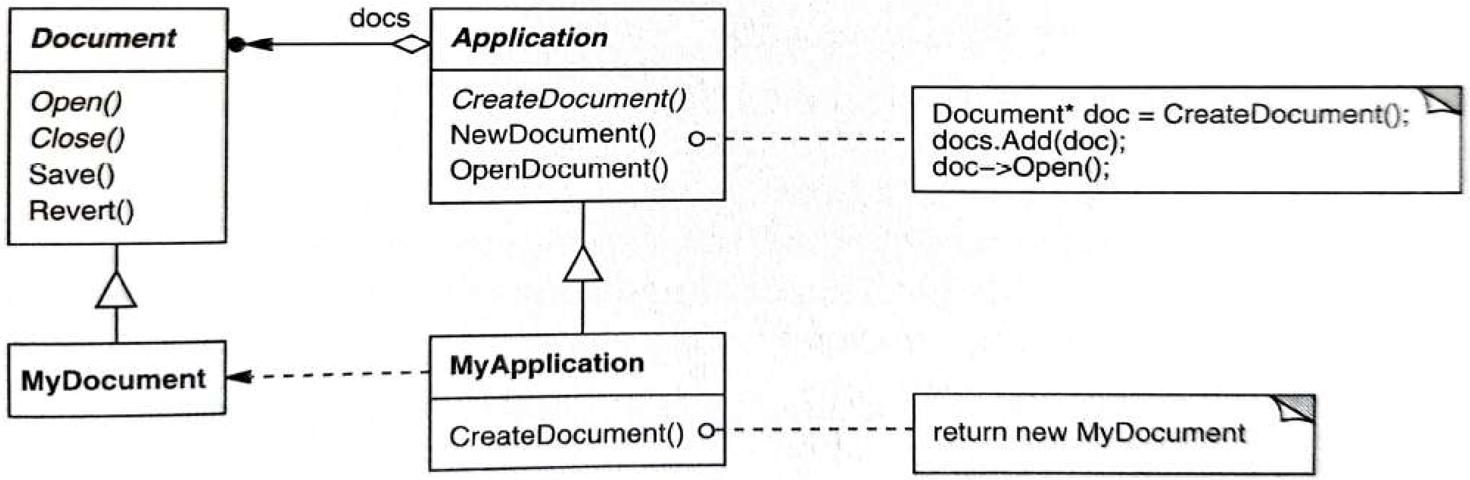
- The JavaFX is a framework. It provides the basis for building a GUI.
- The user then tailors the concrete implementation to their needs.



```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;

public class HelloWorldApplication extends Application {
    public void start(Stage window) {
        Label text = new Label("Hello World!");
        Scene view = new Scene(text);
        window.setTitle("My Greeting Application");
        window.setScene(view);
        window.show();
    }
}
```

Framework approach



```
Document* doc = CreateDocument();
docs.Add(doc);
doc->Open();
```

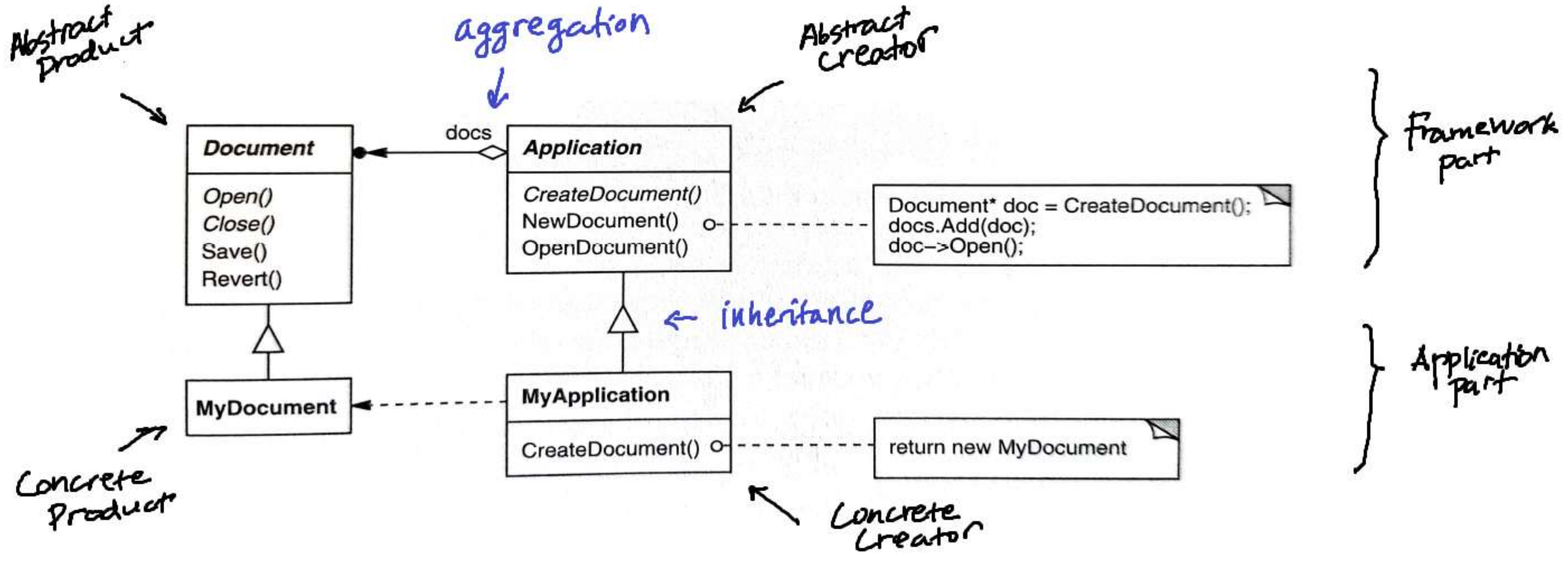
- A framework in the image provides the functionality to deal with any applications and documents.
- The framework is not interested in the actual concrete application nor document

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 107

Idea of Factory Method

- Define abstractions (i.e. interfaces or abstract classes) for objects and their creators.
- The concrete implementations of the creator then define which objects to create.
- Thus, the idea is to **defer** object instantiation.
- The application just deals with the interfaces, not the concrete classes.

Example explained



- The framework part acts provides generalization.
  - The framework is able to deal with *any* applications (such as MyApplication) as well as their corresponding documents (such as MyDocument).

General structure

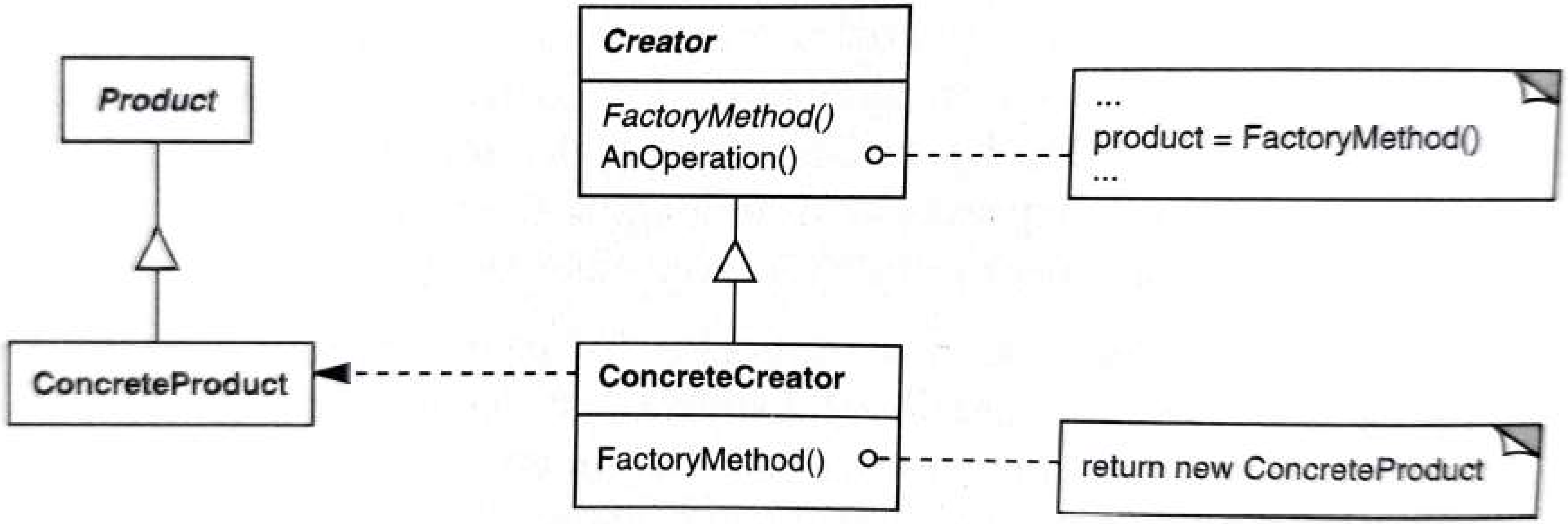
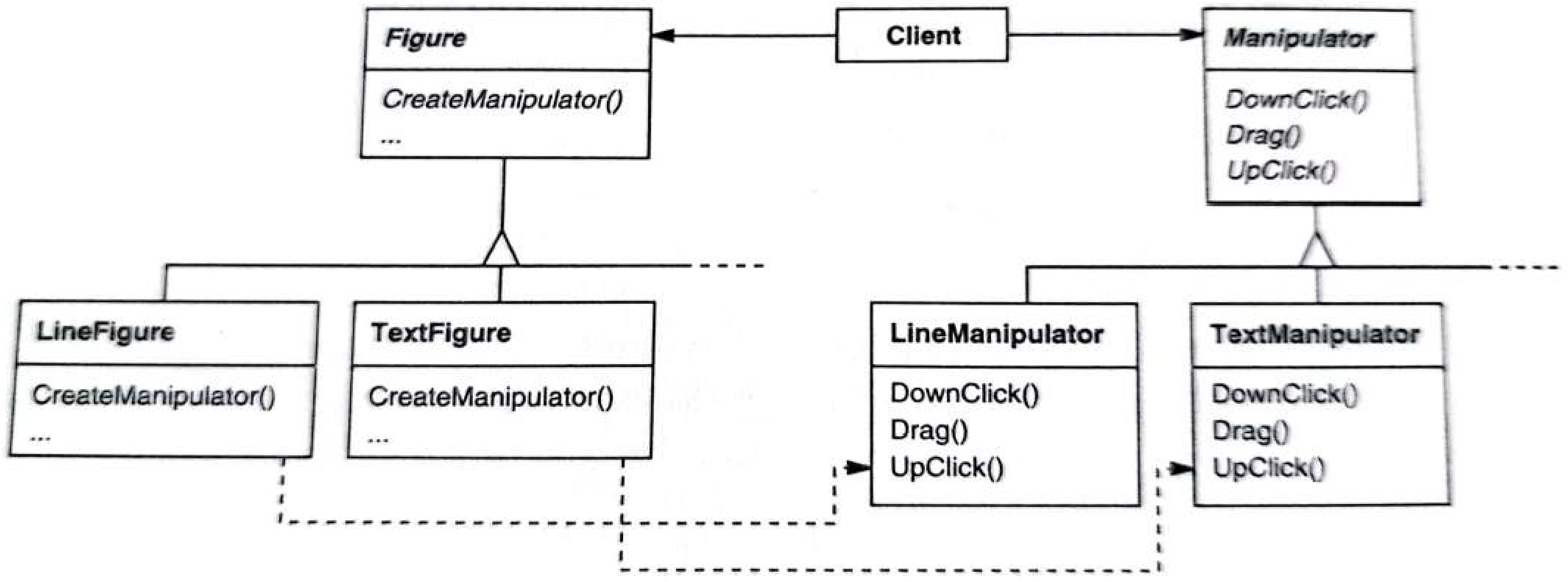


Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 107

Roles

- **Abstract product**: declares the operations that the concrete products must implement.
  - Can be an interface or an abstract class
- **Concrete product**: implements the concrete object to be created.
- **(Abstract) Creator**: declares the interface that contains the factory method for creating an abstract product.
  - Can be an interface of an abstract class
- **Concrete Creator**: declares the actual creation of the concrete products.

Connecting class hierarchies



- In the image, there are parallel class hierarchies.
- The factory method in Figure is responsible for creating a correct manipulator.
- The Client creates a Figure of the chosen type and is guaranteed to get a correct Manipulator as well.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 107

Connecting class hierarchies

- Each figure type has its own manipulator class.
  - For example, **`LineFigure`** represents a line, whereas **`LineManipulator`** provides tools and operations for lines such as dragging, scaling, etc.
  - These operations may be complex and figure-specific with temporary information related to operations, so dedicated classes are warranted.
  - Here, Factory Method DP helps:
    - Each concrete figure has its own implementation of **`createManipulator`**() method.
    - Thus, a **`LineFigure`** is able to create a **`LineManipulator`** etc.

Collaborative Task (10 mins)

- Let's think about the **ArrayList** class in the **java.util** package.
- It has an **iterator()** method that provides an **Iterator** object for iterating over the **ArrayList** elements.
- Browse the API documentation for the following classes and/or interfaces:
  - **ArrayList**
  - **AbtractList**
  - **Collection**
  - **Iterator**
- Sketch a class diagram of the involved classes and interfaces.
- How is the Factory Method design pattern applied here?

Interfaces vs. abstract classes

- In Java, either interfaces or abstract classes can be used as abstract products and abstract creators.
- Benefits of using an interface:
  - Can be used to circumvent the need for multiple inheritance.
  - Provides a clean API.
  - Loose coupling. Implementing classes only have a minimal connection to the interface definition.
- Benefits of using an abstract class:
  - Easy to provide a partial implementation.
  - More flexible evolution: adding a method to an interface requires updating all implementing classes.
  - Can store instance variables.