8. Шаблоните фабрики (Factory)

ЛЕКЦИОНЕН КУРС: ШАБЛОНИ ЗА ПРОЕКТИРАНЕ

ДОЦ. Д-Р ЕМИЛ ДОЙЧЕВ

Шаблоните Фабрики

- ✓ Шаблоните фабрики са пример за създаващи шаблони.
- ✓ *Създаващите шаблони* отделят процеса на инстанция на обектите. Те скриват детайлите по създаването на обектите и помагат като цяло системата да е независима от това как се създават и как се композират нейните обекти.
- ✓ *Създаващите шаблони за класове* се фокусират върху използването на наследяването за да определят кой обект трябва да бъде инстанциран.
  - *Метод Фабрика (Factory Method)*
- ✓ Създаващите шаблони за обекти преотстъпват правото за инстанциране на друг обект.
  - *Абстрактна Фабрика (Abstract Factory)*

Шаблоните Фабрики

- ✓ Всички ОО езици за програмиране имат идиом за създаване на обекти. В Java този идиом е операторът **`new`**.

- ✓ Създаващите шаблони ни позволяват да направим методи, които създават нови обекти без изрично да използват оператора `new`.

- ✓ Това ни позволява да направим методи, които могат да инстанцират различни обекти, и които могат да бъдат разширени да инстанцират обекти – разработени в последствие.

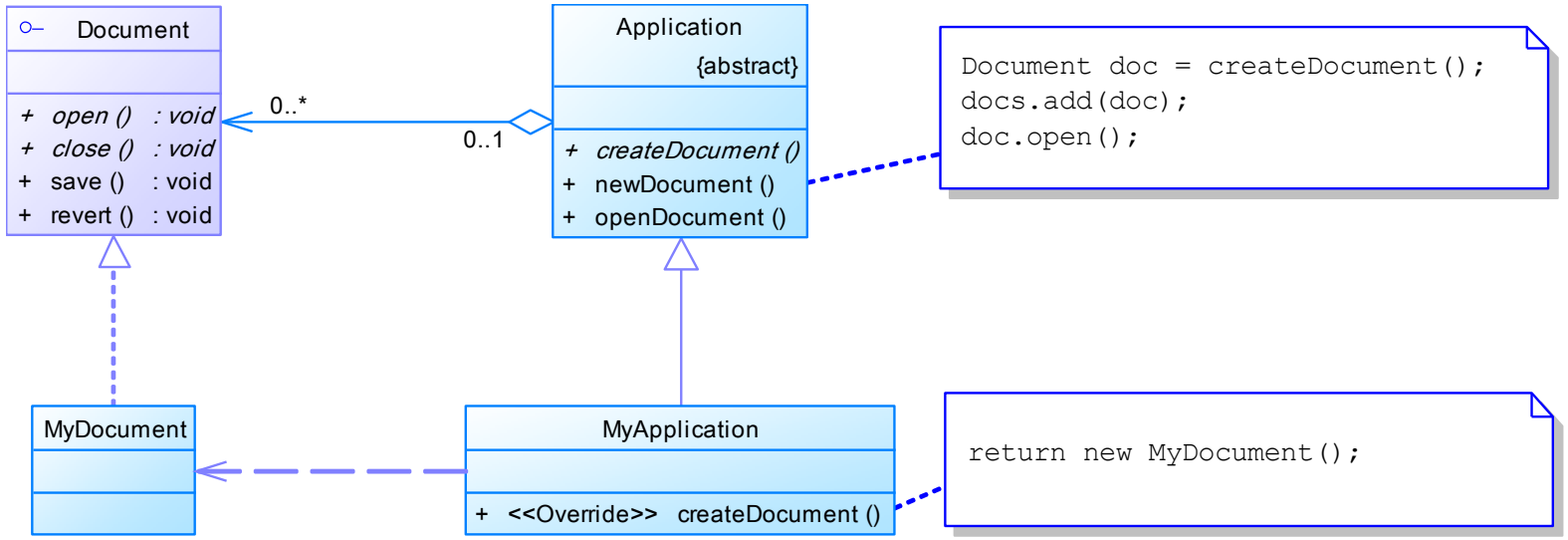
- ✓ И всичко това без да се модифицира кода на метода.

Метод Фабрика (Factory Method)

- ✓ **Цел:** Дефинира интерфейс за създаване на обект, но позволява на подкласовете да решат кой клас да инстанциират. Метод фабрика дава възможност на класа да преотстъпи процеса на създаване на подкласове.
- ✓ **Известен и като:** Виртуален конструктор (Virtual Constructor)

Метод Фабрика (Factory Method)

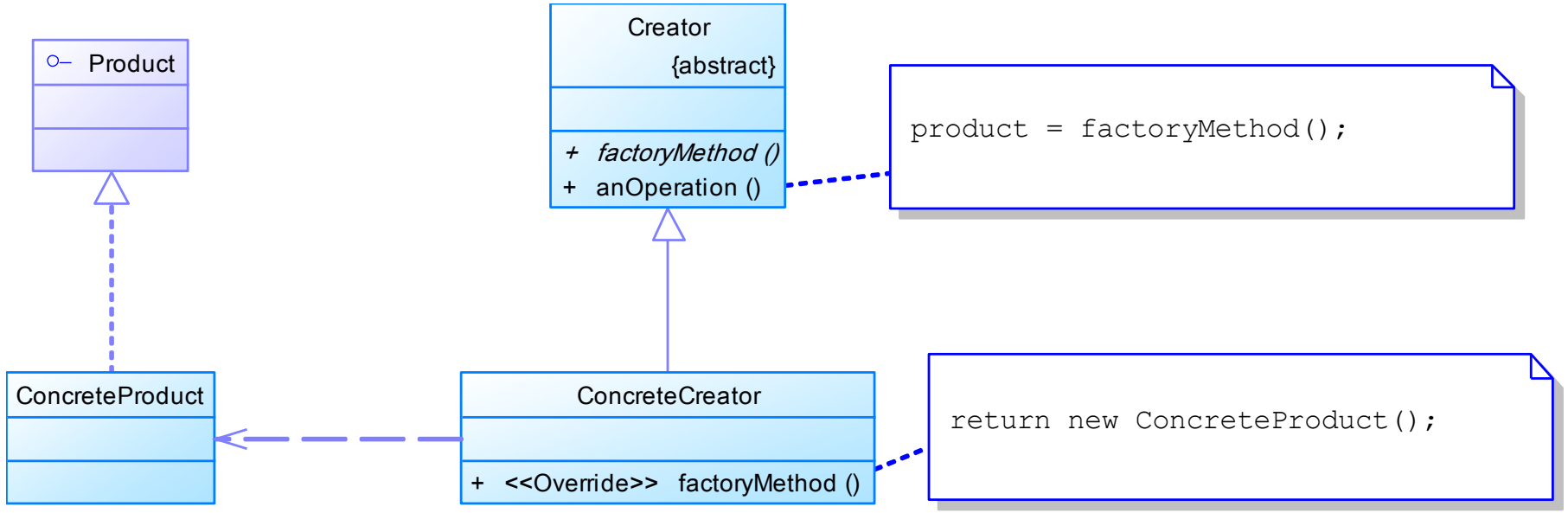
- **Мотивация:** Рамките за приложения използват абстрактни класове, за да дефинират взаимовръзки между обектите. Често рамката е отговорна и за създаването на тези обекти.



- Например – рамка за приложения, които могат да представят множество документи на потребителя.
- Ключови абстракции са класовете *Application* и *Document*. И двата класа са абстрактни и клиентите трябва да създадат техни подкласове, за да реализират специфичните си имплементации – напр. *DrawingApplication* и *DrawingDocument*.
- Т.е. рамката трябва да инстанцира класове, но тя познава само абстрактните класове, които не може да инстанцира.

Метод Фабрика (Factory Method)

- ✓ **Приложимост:** Шаблонът Метод Фабрика се използва в следните случаи:
  - Някой клас не може да предвиди класа на обектите, които трябва да създаде.
  - Някой клас иска неговите наследници (подкласове) да определят обектите, които създава.
- ✓ **Структура:**



Метод Фабрика (Factory Method)

✓ **Участници**

- **Product** (Document) – дефинира интерфейса на обектите, създавани от метода фабрика.
- **ConcreteProduct** (MyDocument) – имплементира интерфейса Product.
- **Creator** (Application) – декларира метода фабрика, който връща обект от типа Product.
  - Възможно е да дефинира и имплементация по подразбиране, която да връща подразбиращ се обект ConcreteProduct.
  - Може да извиква метода фабрика за да създаде Product обект.
- **ConcreteCreator** (MyApplication) – предефинира метода фабрика, така че да връща инстанция на ConcreteProduct.

✓ **Взаимодействия.** Creator разчита на своите подкласове да имплементират метода фабрика, така че той да може да върне инстанцията на съответния ConcreteProduct.

Метод Фабрика (Factory Method)

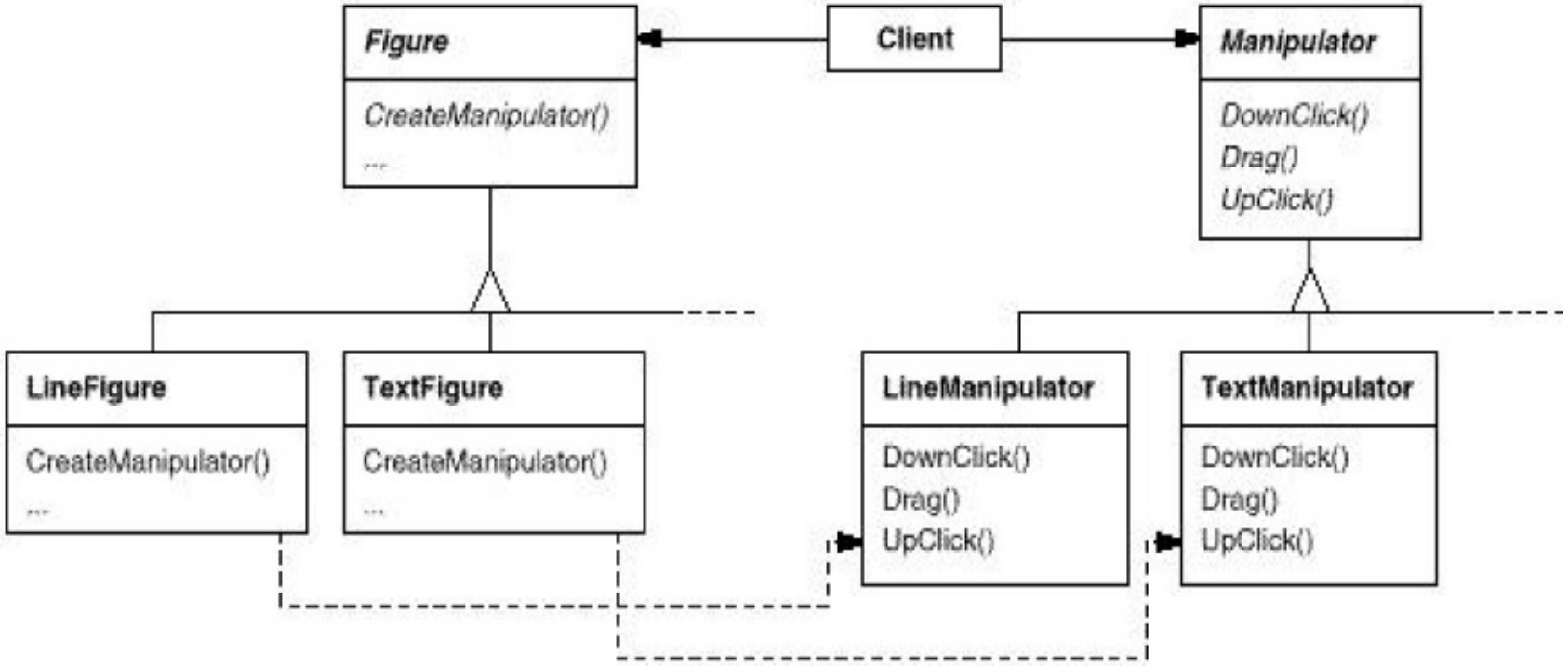
✓ Какво точно означава, че шаблонът Метод фабрика „позволява на подкласовете да решат кой клас да инстанциират“?

- Означава, че класа Creator е създаден без да е ясно кой точно конкретен обект (ConcreteProduct) ще бъде създаван. Това се определя единствено от това, кой подклас ConcreteCreator ще бъде инстанциран от приложението.

- Не означава, че по някакъв начин подкласовете определят в runtime кой ConcreteProduct клас да създадат.

Метод Фабрика (Factory Method)

- ✓ **Пример** (от рамката Unidraw, за създаване на собствени графични редактори, http://c2.com/cgi/wiki?UnidrawFramework)
  - Методът фабрика в този случай е *createManipulator()*



Метод Фабрика (Factory Method)

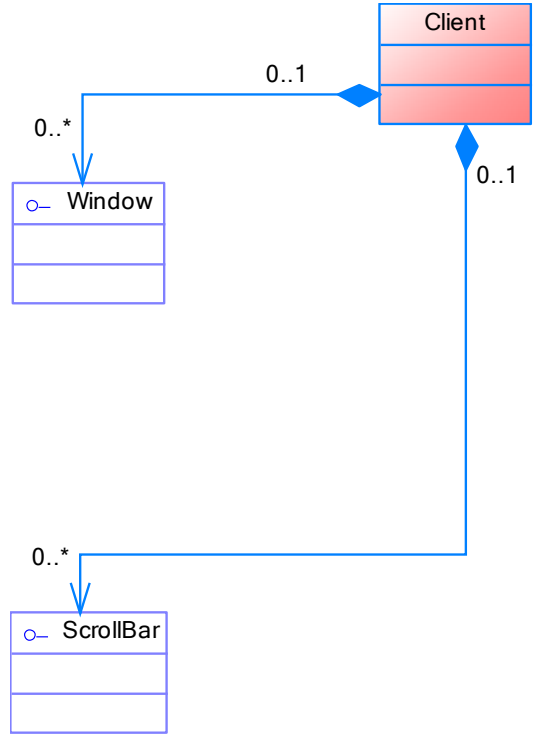
- ✓ **Следствия**
  - Предимства
    - Кода е гъвкав с висока повторна употреба чрез елиминиране на инстанцирането на класове – специфични за приложението.
    - Кода работи единствено с интерфейса Product и може да работи с всеки ConcreteProduct, който имплементира този интерфейс.
  - Недостатъци
    - Клиентите може да трябва да наследят Creator класа само за да инстанцират някакъв ConcreteProduct.
- ✓ **Имплементационни забележки**
  - Класа Creator може да бъде абстрактен или конкретен
  - Ако е необходимо метода фабрика да създава обекти от различен тип – тогава метода трябва да има параметър (вероятно използван в условна конструкция), чрез който да определи кой обект да създаде.

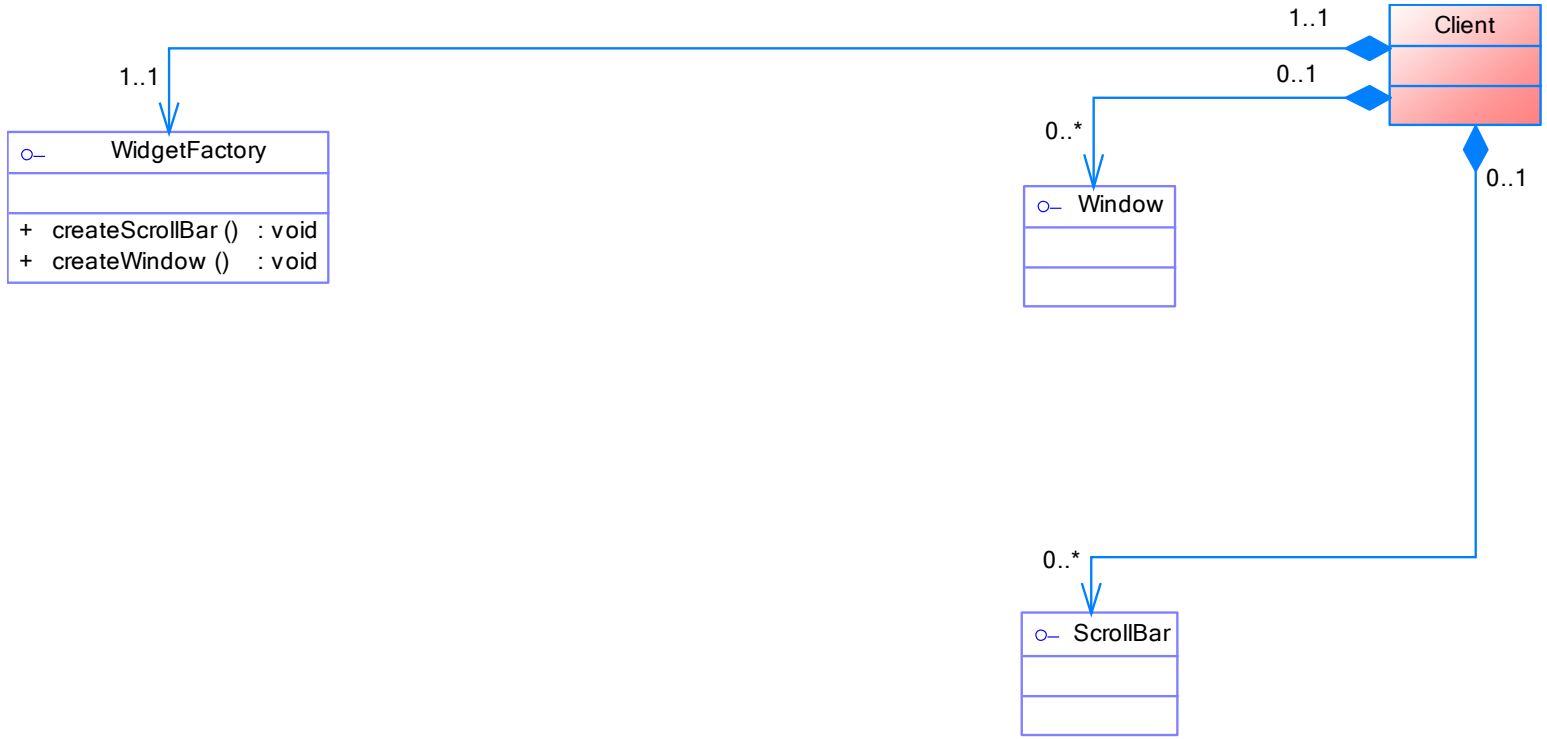
Абстрактна фабрика (Abstract Factory)

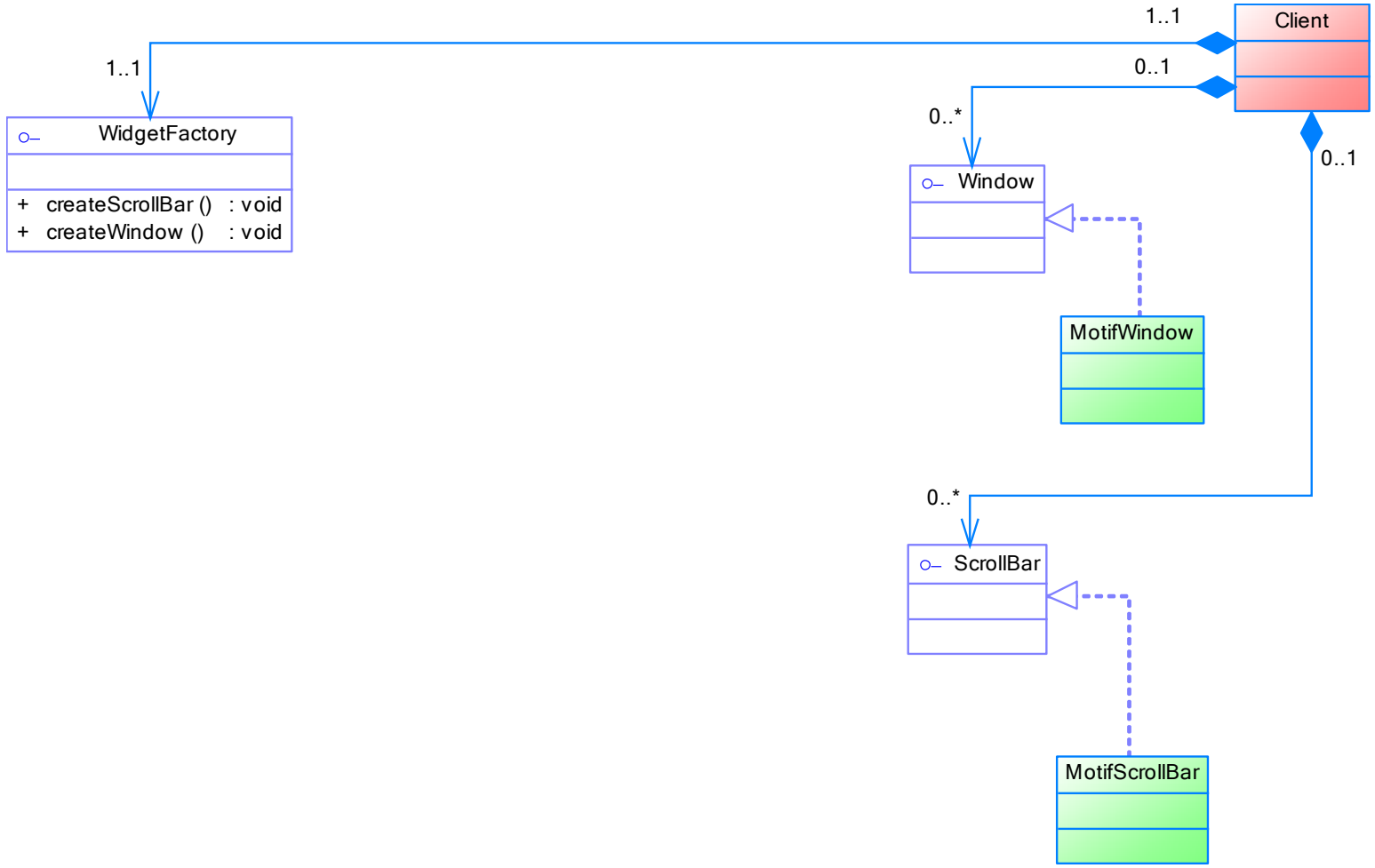
- ✓ **Цел:** Предоставя интерфейс за създаване на семейства взаимосвързани или зависими обекти, без да се задават конкретните им класове.
  - Подобен на шаблона Метод фабрика
  - Една от разликите е, че с шаблона Абстрактна фабрика клас може да делегира правата за инстанциране на обекти на друг обект, докато с Метод фабрика това става само с наследяване и отговорността за инстанциране на обектите се поема от клас наследник.
- ✓ **Известен и като:** Комплект (Kit)

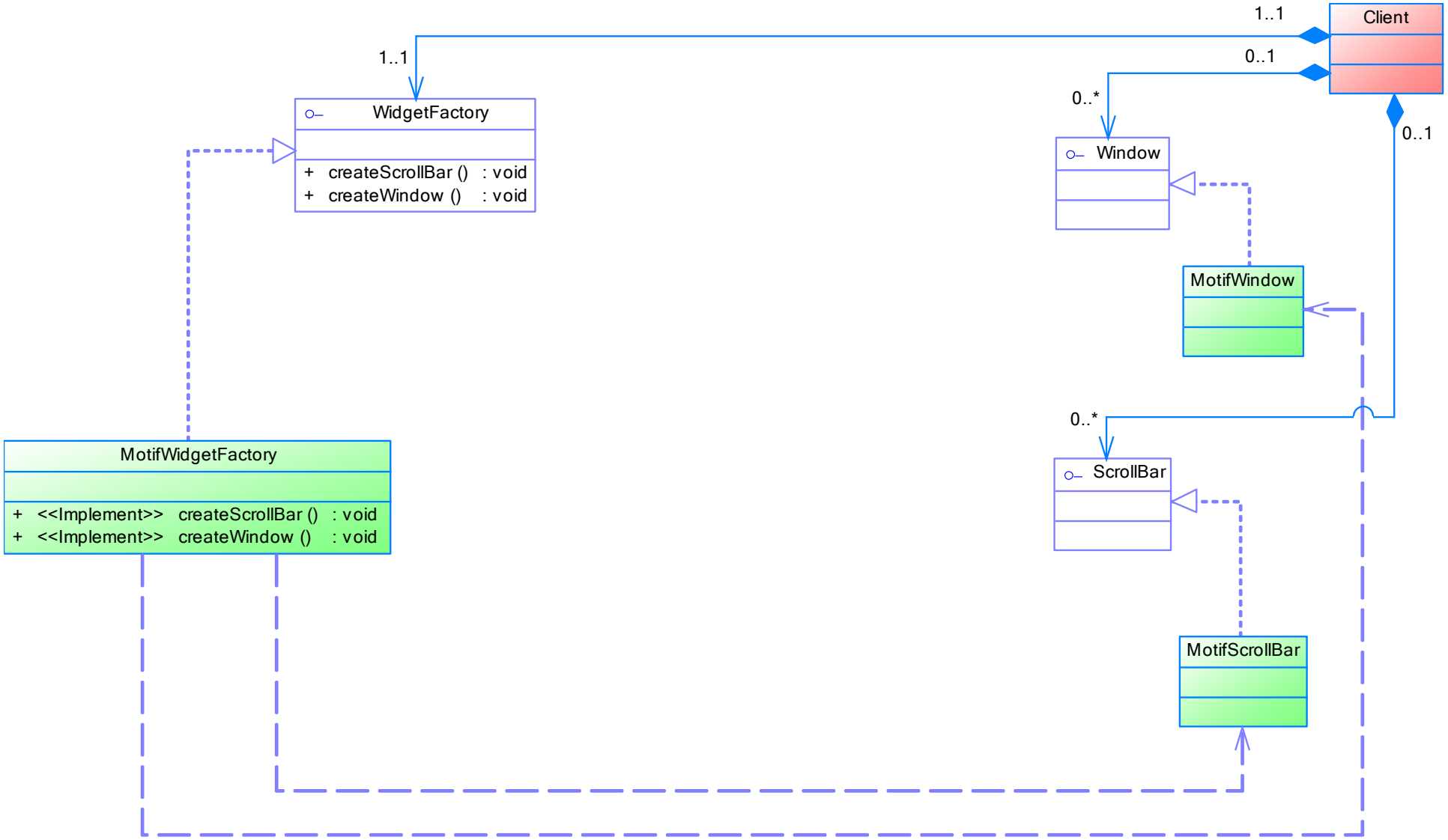
Абстрактна фабрика (Abstract Factory)

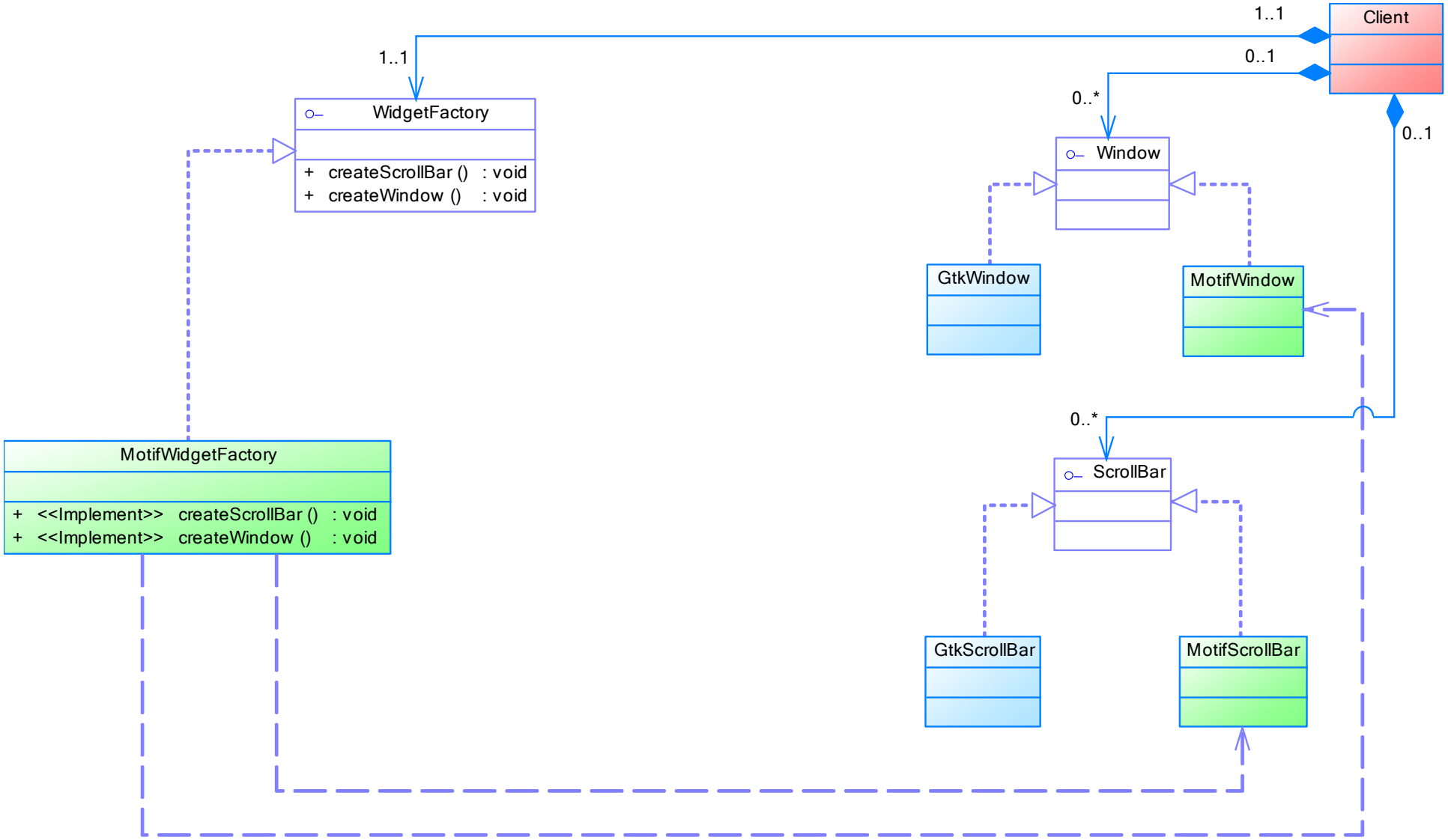
- ✓ **Мотивация:** пример - набор от инструменти за потребителски интерфейс, които поддържат множество стандарти за външен вид (look & feel) – напр. Motif, Presentation Manager, и др.
  - Дефиниране на абстрактен клас WidgetFactory, с интерфейс за създаване на всички основни видове графични елементи.
  - Деф. на абстрактен клас за всеки вид управляващ елемент.
  - Конкретните подкласове имплементират графични елементи за конкретните стандарти за външен вид.
  - Клиентите нямат представа за конкретните класове, които използват и остават независими от избрания външен вид.

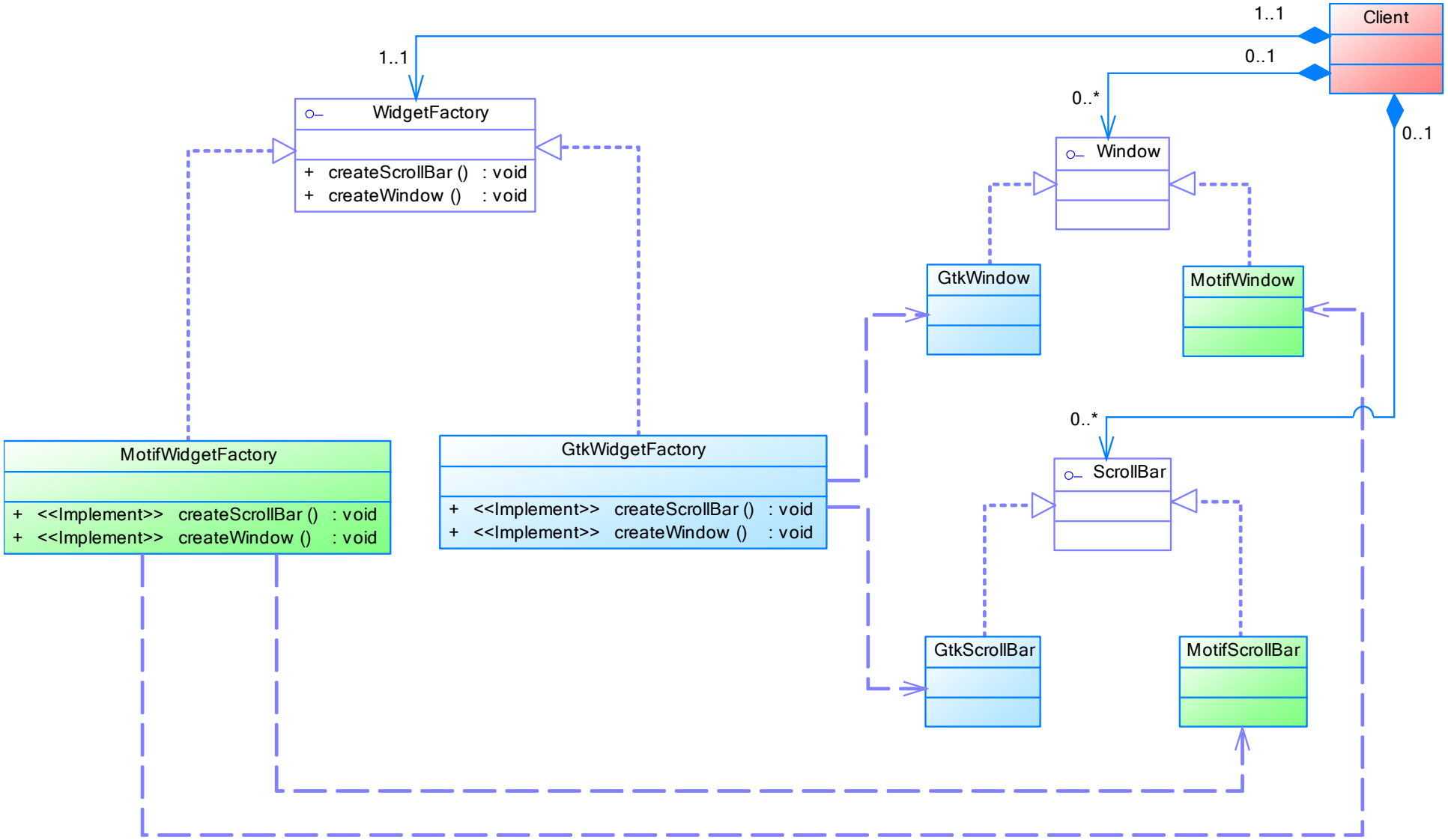

Client
0..1
0..*
Window
0..1
0..*
ScrollBar


1..1
Client
1..1
0..1
0..*
WidgetFactory
+ createScrollBar () : void
+ createWindow () : void
Window
0..1
0..*
ScrollBar


1..1
Client
1..1
0..1
0..*
0..1
WidgetFactory
+ createScrollBar () : void
+ createWindow () : void
Window
MotifWindow
0..*
ScrollBar
MotifScrollBar


1..1
Client
1..1
0..1
0..*
0..1
WidgetFactory
+ createScrollBar () : void
+ createWindow () : void
Window
MotifWindow
0..*
ScrollBar
MotifWidgetFactory
+ <<Implement>> createScrollBar () : void
+ <<Implement>> createWindow () : void
MotifScrollBar


1..1
Client
1..1
0..1
0..*
0..1
WidgetFactory
+ createScrollBar () : void
+ createWindow () : void
Window
GtkWindow
MotifWindow
0..*
ScrollBar
MotifWidgetFactory
+ <<Implement>> createScrollBar () : void
+ <<Implement>> createWindow () : void
GtkScrollBar
MotifScrollBar

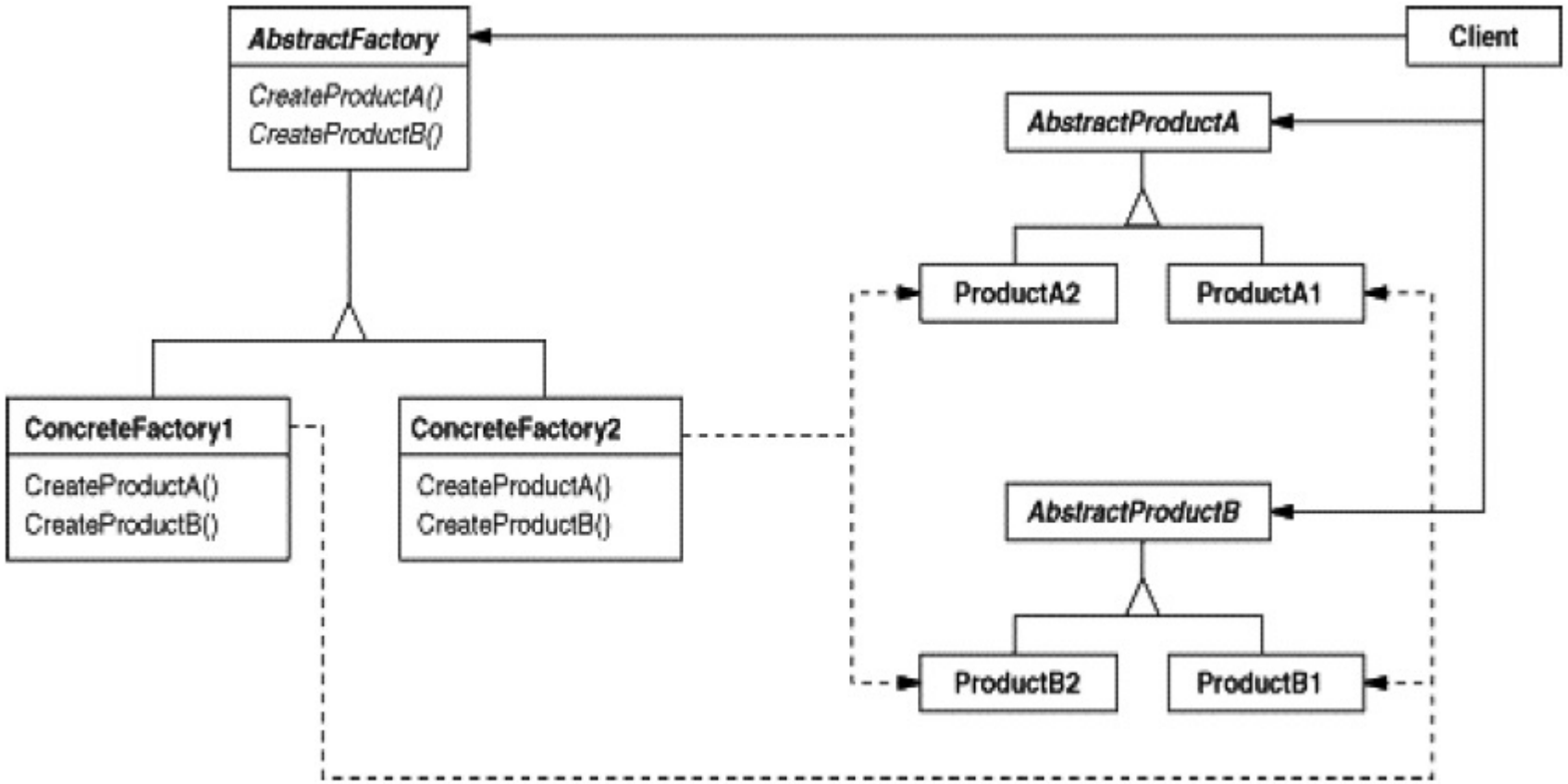

1..1
Client
1..1
0..1
0..*
0..1
WidgetFactory
+ createScrollBar () : void
+ createWindow () : void
Window
GtkWindow
MotifWindow
MotifWidgetFactory
+ <<Implement>> createScrollBar () : void
+ <<Implement>> createWindow () : void
GtkWidgetFactory
+ <<Implement>> createScrollBar () : void
+ <<Implement>> createWindow () : void
0..*
ScrollBar
GtkScrollBar
MotifScrollBar

Абстрактна фабрика (Abstract Factory)

- ✓ За всеки стандарт за външен вид се създава по един конкретен подклас на WidgetFactory.
- ✓ Всеки подклас имплементира операциите за създаване на съответните графични елементи за дадения външен вид.
- ✓ WidgetFactory налага зависимости между конкретните класове за графични елементи. Всяка лента за скролиране за Motif трябва да се използва с бутон за Motif и текстов редактор за Motif.

Абстрактна фабрика (Abstract Factory)

- ✓ **Структура**



Абстрактна фабрика (Abstract Factory)

- ✓ **Участници**
  - **AbstractFactory** (WidgetFactory) - декларира интерфейс за операции, създаващи абстрактни обекти продукти.
  - **ConcreteFactory** (MotifWidgetFactory, PMWidgetFactory) – имплементира операциите за създаване на конкретни обекти продукти.
  - **AbstractProduct** (Window, ScrollBar) – декларира интерфейс за типа на даден обект продукт.
  - **ConcreteProduct** (MotifWindow, MotifScrollBar)
    - дефинира обект продукт, който да се създаде от съответната фабрика;
    - имплементира интерфейса AbstractProduct.
  - **Client** – използва само интерфейси, декларирани от класовете AbstractFactory и AbstractProduct.

Абстрактна фабрика (Abstract Factory)

✓ **Взаимодействия**

- Обикновено по време на изпълнението се създава само по една инстанция на класа ConcreteFactory. Конкретната фабрика създава обекти продукти с определена имплементация.
- За да създадат други обекти продукти, клиентите трябва да използват друга конкретна фабрика.
- AbstractFactory делегира създаването на обекти продукти на своя подклас ConcreteFactory.

Абстрактна фабрика (Abstract Factory)

- ✓ **Пример:** Java 1.1 Abstract Window Toolkit (AWT) е предназначен да предостави GUI (graphical user interface/графичен потребителски интерфейс) в хетерогенна среда.
- ✓ AWT използва Абстрактна фабрика за да генерира всички необходими визуални компоненти за специфичната платформа, на която работи приложението.
- ✓ Например: част от кода на класа java.awt.List

```
public class List extends Component implements ItemSelectable {
...
  peer = getToolkit().createList(this);
...
}
```

- ✓ Метода getToolkit() е наследен от класа Component и връща указател към обекта фабрика, използван за създаване на всички AWT компоненти.

Абстрактна фабрика (Abstract Factory)

- Ето как изглежда getToolkit() метода в класа Component

```
public Toolkit getToolkit() {
  // If we already have a peer, return its Toolkit.
  ComponentPeer peer = this.peer;
  if ((peer != null) && ! (peer instanceof
     java.awt.peer.LightweightPeer)){
      return peer.getToolkit();
  }
  // If we are already in a container, return its Toolkit.
  Container parent = this.parent;
  if (parent != null) {
    return parent.getToolkit();
  }
  // Else return the default Toolkit.
  return Toolkit.getDefaultToolkit();
}
```

Абстрактна фабрика (Abstract Factory)

- ✓ И метода getDefaultToolkit() в Toolkit класа

```
public static synchronized Toolkit getDefaultToolkit() {
  if (toolkit == null)
    String nm = System.getProperty("awt.toolkit",
                          "sun.awt.motif.MToolkit");
    toolkit = (Toolkit)Class.forName(nm).newInstance();
  }
  return toolkit;
}
```

Абстрактна фабрика (Abstract Factory)

- ✓ **Следствия**
  - Предимства
    - Изолира клиентите от конкретните класове.
    - Улеснява смяната на семействата продукти, тъй като една конкретна фабрика може да поддържа пълния набор от продукти.
    - Допринася за съвместимостта между продуктите – използват се само продукти от едно семейство
  - Недостатъци
    - Поддържането на нови видове продукти е трудно – необходимо е разширяване на интерфейса AbstractFactory.

Абстрактна фабрика (Abstract Factory)

- ✓ **Имплементационни забележки**
  - Колко инстанции на конкретна фабрика са необходими?
    - Едно приложение обикновено се нуждае само от една инстанция на конкретна фабрика.
    - Може да се използва Singleton (сек) шаблона.
  - Как фабриките могат да създават продуктите?
    - Factory Methods
    - Factories
  - Как могат да бъдат добавяни нови продукти към AbstractFactory интерфейса?
    - AbstractFactory дефинира различен метод за създаването на всеки отделен тип продукт.
    - Може да се промени интерфейса да съдържа само метода

```
make(String kindOfProduct)
```

Как фабриките създават продукти?

- **Метод 1:** Чрез използване на методи фабрики

```
/**
 * WidgetFactory.
 * This WidgetFactory is an abstract class.
 * Concrete Products are created using the factory methods
 *   implemented by sublcasses.
 */
public abstract class WidgetFactory {
  public abstract Window createWindow();
  public abstract Menu createScrollBar();
  public abstract Button createButton();
}
```

Как фабриките създават продукти? (продължение)

```
/**
 * MotifWidgetFactory.
 * Implements the factory methods of its abstract superclass.
 */
public class MotifWidgetFactory
  extends WidgetFactory {

  public Window createWindow() {return new MotifWindow();}
  public ScrollBar createScrollBar() {
    return new MotifScrollBar();}
  public Button createButton() {return new MotifButton();}

}
```

Как фабриките създават продукти? (продължение)

- ✓ Типичен код за клиент. Кода е един и същ независимо как фабриката създава продуктите.

```
...
// Create new factory.
WidgetFactory wf = new MotifWidgetFactory();

// Create a button.
Button b = wf.createButton();

// Create a window.
Window w = wf.createWindow();
...
```

Как фабриките създават продукти? (продължение)

- **Метод 2.** Чрез използване на фабрики.

```
/**
 * WidgetFactory.
 * This WidgetFactory contains references to factories
 *   (composition!) used to create the Concrete Products.
 *   But it relies on a subclass constructor to create the
 *   appropriate factories.
*/
public abstract class WidgetFactory {
  protected WindowFactory windowFactory;
  protected ScrollBarFactory scrollBarFactory;
  protected ButtonFactory buttonFactory;

  public Window createWindow() {return
    windowFactory.createWindow();}
```

Как фабриките създават продукти? (продължение)

```
  public ScrollBar createScrollBar() {return
    scrollBarFactory.createScrollBar();}
  public Button createButton() {return
    buttonFactory.createButton();}
}

/**
 * MotifWidgetFactory.
 * Instantiates the factories used by its superclass.
 */
public class MotifWidgetFactory
  extends WidgetFactory {
  public MotifWidgetFactory() {
    windowFactory = new MotifWindowFactory();
    scrollBarFactory = new MotifScrollBarFactory();
    buttonFactory = new MotifButtonFactory();
  }
}
```

Как фабриките създават продукти? (продължение)

- ✓ **Метод 3.** Чрез използване на фабрики без необходимост от създаване на подкласове (чиста композиция)

```
/**
 * WidgetFactory.
 * This WidgetFactory contains reference to factories used
 *   to create Concrete Products. But it does not need to be
 *   subclassed.  It has an appropriate constructor to set
 *   these factories at creation time and mutators to change
 *   them during execution.
 */
public class WidgetFactory {

  private WindowFactory windowFactory;
  private ScrollBarFactory scrollBarFactory;
  private ButtonFactory buttonFactory;
```

Как фабриките създават продукти? (продължение)

```
public WidgetFactory(WindowFactory wf,
                     ScrollBarFactory sbf,
                     ButtonFactory bf) {
  windowFactory = wf;
  scrollBarFactory = sbf;
  buttonFactory = bf;
}

ublic void setWindowFactory(WindowFactory wf) {
  windowFactory = wf;
}

public void setScrollBarFactory(ScrollBarFactory sbf) {
  scrollBarFactory =sbf;
}

public void setButtonFactory(ButtonFactory bf) {
  buttonFactory = bf;
}
```

Как фабриките създават продукти? (продължение)

```
  public Window createWindow() {return
    windowFactory.createWindow();}
  public ScrollBar createScrollBar() {return
    scrollBarFactory.createScrollBar();}
  public Button createButton() {return
    buttonFactory.createButton();}
}
```

Край: Шаблоните фабрики

ЛЕКЦИОНЕН КУРС: ШАБЛОНИ ЗА ПРОЕКТИРАНЕ