# Creational Patterns: Abstract Factory и Builder

Доц. д-р Станка Хаджиколева

# Creational Patterns (Градивни шаблони)

- **Abstract Factory**
- **Builder**
- Factory Method
- Prototype
- Singleton

# Abstract Factory (Абстрактна фабрика)

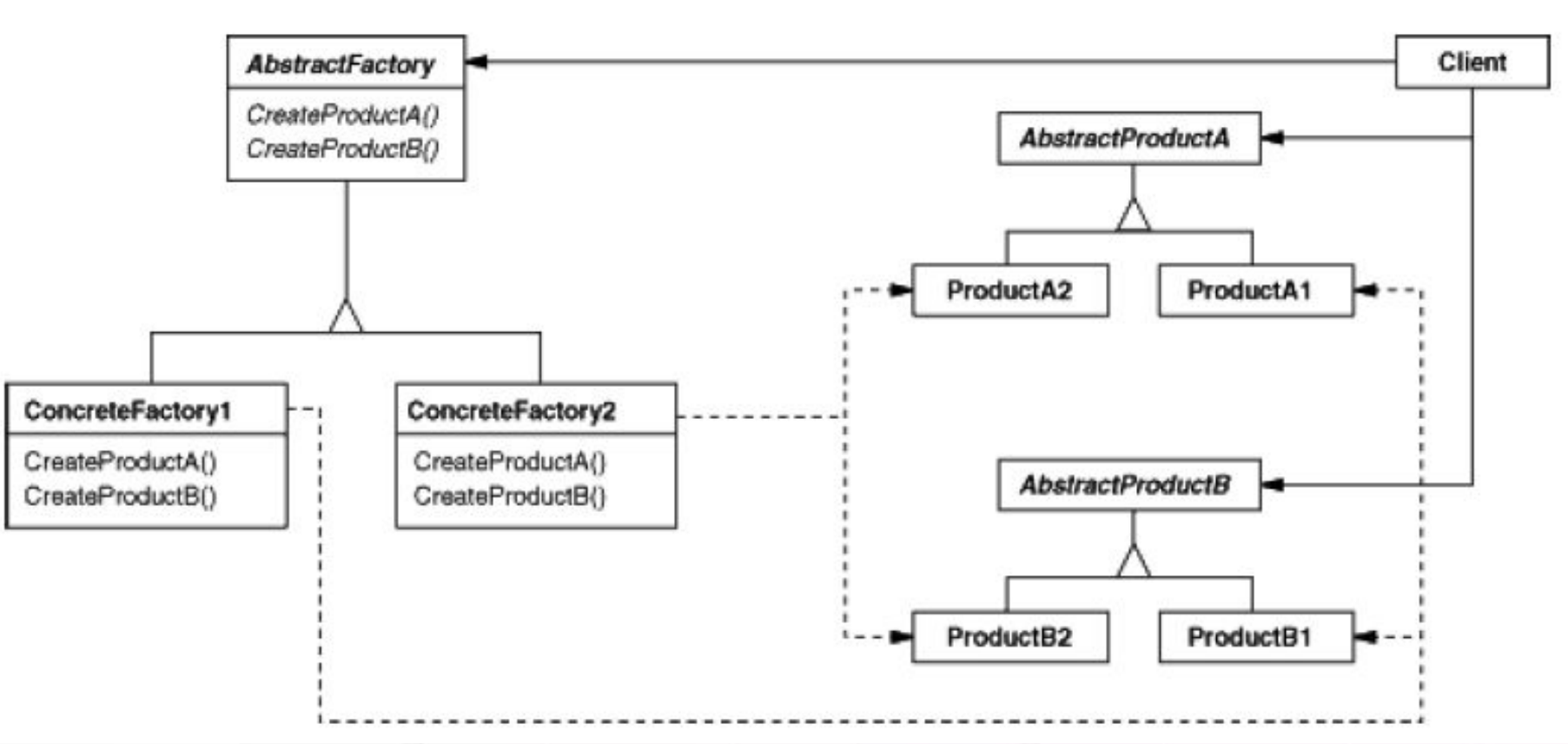
- **Цел:** Доставя интерфейс за създаване на семейства от свързани или зависими обекти, без да се специфицират техните конкретни класове.

# Употреба

Подходящо е да се използва в следните случаи:

- една система трябва да бъде независима от това, как се създават, композират и представят нейните продукти;
- една система трябва да бъде конфигурирана с една от няколко групи продукти;
- проектирано е група от свързани продукти да бъдат използвани заедно, и това ограничение трябва да се наложи стриктно;
- искате да предоставите библиотеки от класове за продукти, като разкриете само техните интерфейси, без имплементацията им.

# Структура на шаблона



# Структура на шаблона

- **AbstractFactory** - декларира интерфейс с методи за създаване на абстрактни продукти;
- **ConcreteFactory** - имплементира операциите за създаване на конкретни продукти;
- **AbstractProduct** - декларира интерфейс за клас продукти;
- **ConcreteProduct** - дефинира продукт, който да бъде създаден от конкретна фабрика; имплементира интерфейса AbstractProduct;
- **Client** - използва само интерфейси, декларирани чрез AbstractFactory и AbstractProduct класове.

# Взаимодействия

- Обикновено в рънтайма се създава една единствена инстанция на класа ConcreteFactory. Тази конкретна фабрика създава продукти, които имат специфична имплементация. За да създадат различни групи продукти, клиентите трябва да използват различни конкретни фабрики.

- AbstractFactory отлага създаването на продукти, то се извършва от подкласа ConcreteFactory.

# Предимства и недостатъци

- **Изолира конкретните класове.** Една фабрика капсулира дейностите по създаване на продукти, и по този начин изолира клиентите от имплементацията на класовете. Клиентите манипулират инстанциите чрез техните абстрактни интерфейси. Имената на класовете на продуктите са изолирани в имплеметацията на конкретната фабриката; те не се появяват никъде в клиентския код.
- **Лесна замяна на групи продукти.** Класът на конкретна фабрика се появява само веднъж в кода на приложението - където се създава нейна инстанция. Това позволява лесна промяна на фабриката, съответно на цяла група продукти. Тъй като една абстрактна фабрика създава цяла група от продукти, цялата група продукти се

# Предимства и недостатъци

- **Добавянето на нови видове продукти не е лесно.** AbstractFactory интерфейса фиксира множество от продукти, които могат да бъдат създавани. Добавянето на нови видове продукти изисква разширяване на интерфейса AbstractFactory, което води след себе си до промяна на всички класове, които го наследяват.
- **Налага съгласуваност между продукти.** Когато е проектирано група продукти да работят заедно, е важно приложението да използва обектите от само една група в даден момент. AbstractFactory позволява лесно да се наложи това ограничение.

# Връзка с други шаблони

- AbstractFactory класовете често се имплементират с Factory Method, може да се използва и Prototype.
- Конкретна фабрика често е удачно да се реализира с шаблона Singleton.

# Builder (Строител)

**Цел:**

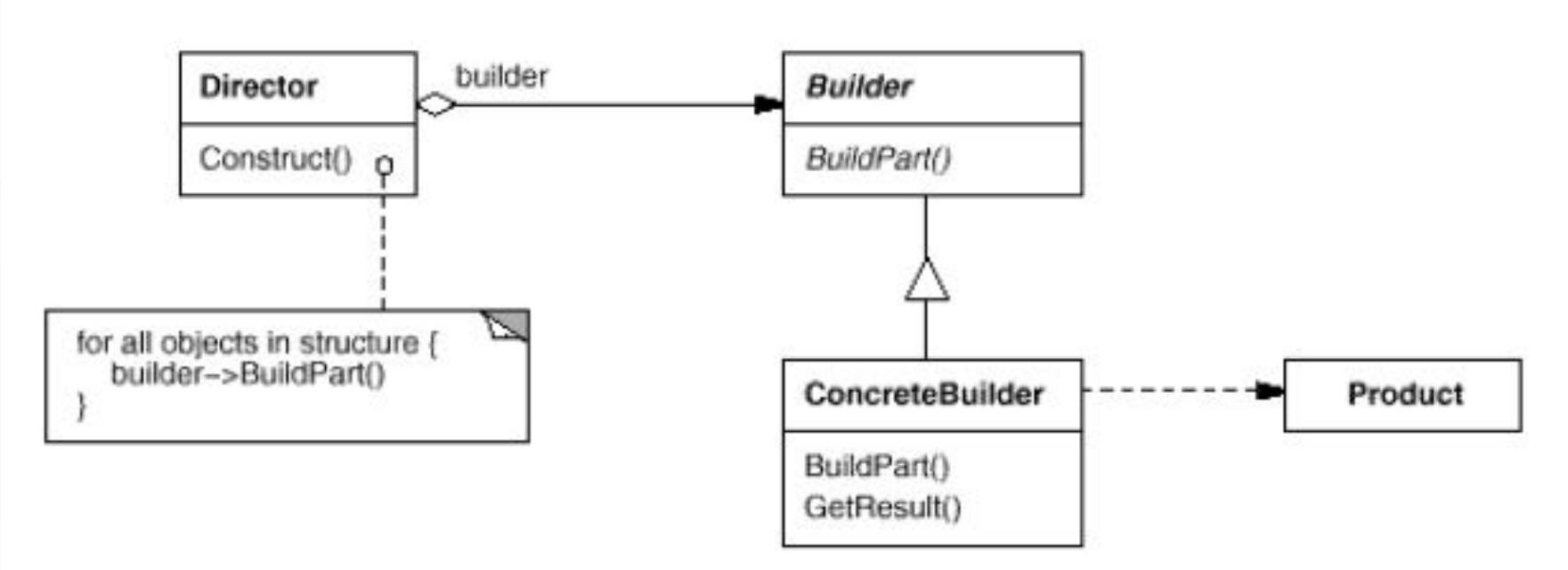
- Разделя процеса на създаването на сложен продукт от неговото представяне.
- Един и същи процес за създаване на продукт може да създаде продукт с различни представяния.

# Употреба

Подходящо е да се използва в следните случаи:

- алгоритъмът за създаване на сложен обект трябва да бъде независим от частите, от които е съставен обекта и начина, по който е асемблиран.
- процесът на конструиране трябва да позволи различни представяния на обекта.

# Структура на шаблона



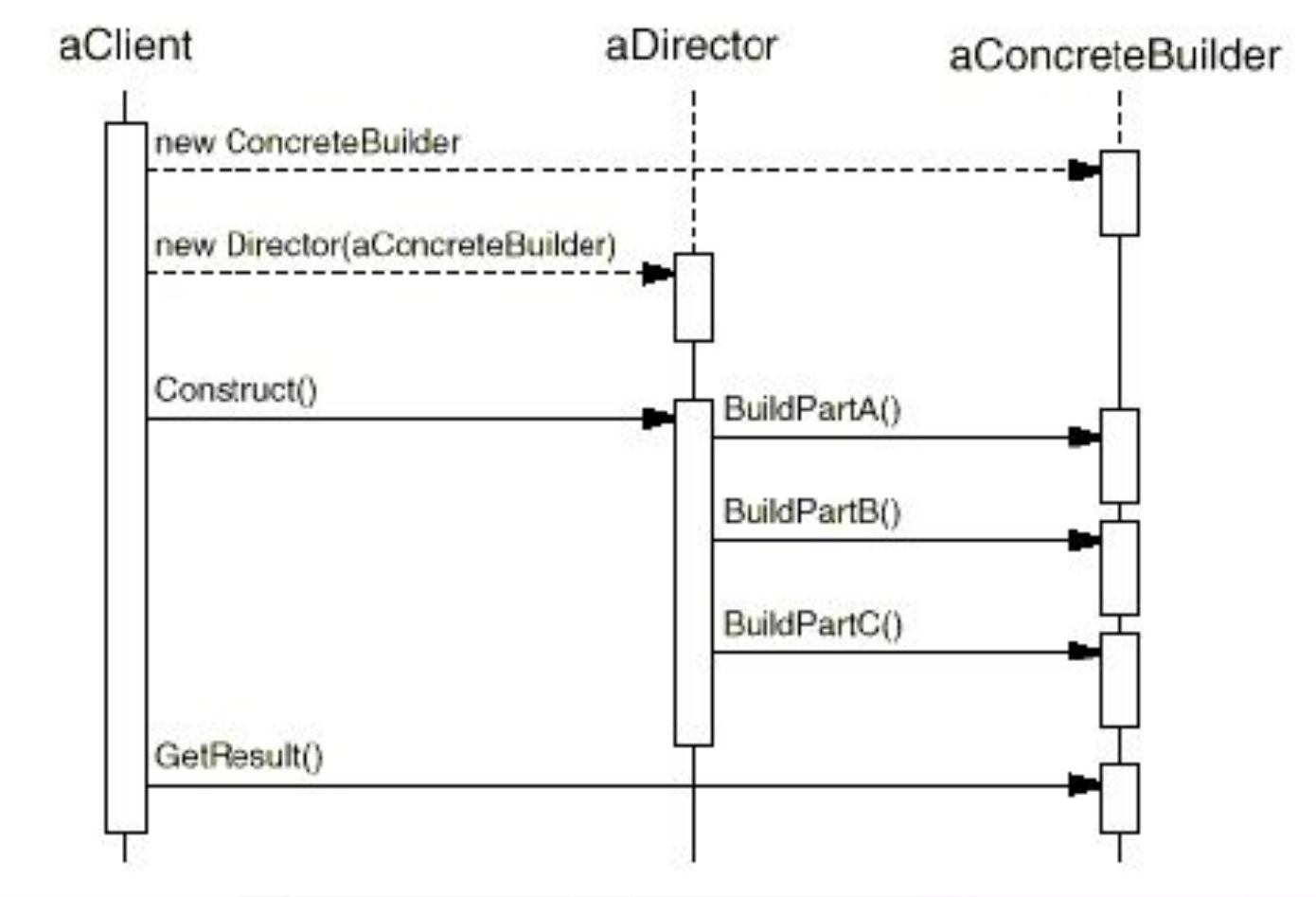
# Структура на шаблона

- **Builder** - абстрактен интерфейс за създаване на части от продукт;
- **ConcreteBuilder**
  - конструира и асмеблира части на продукта чрез имплементиране на интерфейса Builder;
  - дефинира и следи за създаваните от него представяния;
  - предоставя интерфейс за извличане на продукта.
- **Director** - конструира обект, като използва интерфейса Builder;
- **Product**
  - представлява сложния обект, който се изгражда. ConcreteBuilder създава вътрешното представяне на продукта и дефинира процеса на неговото асемблиране;
  - включва класове, дефиниращи съставните части, включително интерфейсите за асемблиране на частите до крайния резултат.

# Взаимодействия

- Клиентът създава обект Director и го конфигурира с желания Builder обект.
- Director уведомява builder, когато трябва да се изгради някоя част от продукта.
- Builder обработва заявките от director-а и добавя частите към продукта.
- Клиентът извлича продукта от builder-а.

# Диаграма на взаимодействията на Builder и Director с клиента



# Предимства и недостатъци

- **Позволява да се променя вътрешното представяне на продукт.** Обектът Builder доставя на director-а абстрактен интерфейс за изграждане на продукт. Интерфейсът позволява на builder-а да скрие представянето и вътрешната структурата на продукта. Той скрива и как е асемблиран продукта. Тъй като продуктът се конструира чрез абстрактен интерфейс, всичко, което трябва да се направи, за да се промени вътрешното представяне на продукта, е да се дефинира нов вид builder.

# Предимства и недостатъци

- **Изолира кода за конструиране и представяне.** Шаблонът Builder подобрява модулността чрез капсулиране на начина, по който се конструира и представя сложен обект. Клиентите не трябва да знаят нищо за класовете, които дефинират вътрешната структура на продукта (такива класове не са включени в интерфейса на Builder-а).

Всеки ConcreteBuilder съдържа всичкия код за създаване и асемблиране на конкретен вид продукт. Кодът е написан веднъж; след това различни Director обекти могат да го използват многократно, за да изградят различни варианти на продукта от един и същи набор от части.

# Предимства и недостатъци

- **Има силен контрол върху процеса на изграждане на обект.** За разлика от градивните паблони, които изграждат продукти “наведнъж”, шаблонът Builder конструира продукт стъпка по стъпка под контрола на director-а. Чак след като продуктът е завършен, director го взема от builder-а. По този начин интерфейса Builder рефлектира върху процеса на конструиране на продукта повече отколкото другите градивни шаблони. Това дава по-прецизен контрол върху процеса на изграждане и следователно на вътрешната структура на изграждания продукт.

# Връзка с други шаблони

- Abstract Factory прилича на Builder по това, че също може да изгради сложни обекти. Основната разлика е, че шаблона Builder е фокусиран върху изграждане на сложен обект стъпка по стъпка. Акцента на Abstract Factory е върху групи от продукти (прости или сложни). Builder връща продукта като крайна стъпка, за разлика от шаблона Abstract Factory, където продукта се връща веднага.
- Често обекта, изграден от Builder е Composite.

# Благодаря за вниманието!