

Programação Avançada

7

Padrões de Desenho Factory: Simple factory, Factory method e Abstract factory

Bruno Silva, Patrícia Macedo

Sumário



- Enquadramento
 - Métodos de criação vs. fábricas
 - Simple Factory (exemplo aplicável à aula anterior)
- Problema geral e aplicações de:
 - Simple Factory
 - Method factory
 - Abstract factory
- Exercícios
- Prós e contras

Enquadramento

- Os padrões **factory** enquadram-se nos padrões de desenho *criacionais* (ou *de criação*).



Padrões criacionais

Estes padrões fornecem vários mecanismos de criação de objetos, por forma a aumentar a flexibilidade e reutilização de código.

Enquadramento

- (Mas) o termo **fábrica** [*factory*] é ambíguo e, intuitivamente, poderá representar um método ou classe que produz *objetos* (mais concretamente instâncias).
- Vamos desambiguar o termo nas duas seguintes situações:
 - i. método (possivelmente estático) que chama um construtor de uma forma particular ➡ **método de criação** [*creation method*];
 - ii. padrão de desenho criacional ➡ **fábrica** [*factory*].

Enquadramento

- Filosoficamente:
 - Todas as **fábricas** conterão **métodos de criação**;
 - Mas nem todos os métodos de criação consistirão em fábricas (i.e., o inverso não é necessariamente verdade).

Enquadramento

- De seguida ilustramos exemplos de **métodos de criação** aos quais não vamos designar por fábricas.
 - [Exemplo 1] Criar uma nova instância a partir do estado de outra, ou;
 - [Exemplo 2] Verbalizar o estado inicial da instância criada.
- Basicamente consistem em *wrappers* à volta de construtores.

Enquadramento

- [Exemplo 1] de métodos de criação:

```
public class Number {  
    private int value;  
  
    public Number(int value) { this.value = value; }  
  
    public Number sucessor() {  
        return new Number(this.value + 1);  
    }  
  
    public Number predecessor() {  
        return new Number(this.value - 1);  
    }  
  
    // ...  
}
```



`sucessor` e `predecessor` são métodos de criação.

Enquadramento

- [Exemplo 1] de métodos de criação (uso):

```
public class Main {  
    public static void main(String[] args) {  
        Number numberA = new Number(0);  
        Number numberB = numberA.sucessor(); //1  
        Number numberC = numberA.predecessor(); //-1  
        //...  
    }  
}
```

💡 Permite criar uma nova instância a partir do estado de outra.

Enquadramento

- [Exemplo 2] de métodos de criação:

```
public class Invoice {  
    private String vatNumber;  
    //...  
  
    private Invoice(String vatNumber) { /* private! */  
        this.vatNumber = vatNumber;  
        //...  
    }  
  
    public static withVAT(String vatNumber) {  
        return new Invoice(vatNumber);  
    }  
  
    public static withoutVAT() {  
        return new Invoice("999999999");  
    }  
}
```



`withVAT` e `withoutVAT` são métodos de criação.

Enquadramento

- [Exemplo 2] de métodos de criação (uso):

```
public class Main {  
    public static void main(String[] args) {  
        Invoice invoice1 = Invoice.withVat("123456789");  
        //...  
        Invoice invoice2 = Invoice.withoutVat();  
        //...  
    }  
}
```

 Verbaliza que tipo de instância está a ser criada.

Enquadramento

- Relativamente a fábricas existem três padrões principais:
 - Simple Factory
 - Factory Method
 - Abstract Factory
- Cada um é uma "evolução" do anterior !

Simple Factory

- Proposto em *Head First Design Patterns* (2004)

Motivação

- Centralizar a criação de *variantes* de objetos em vez de termos `new` e lógica condicional associada espalhados pelo código-fonte;

Simple Factory

Solução Proposta

- Uma classe que tem um método de criação contendo uma estrutura condicional que escolhe que tipo (variante) de objeto (produto) retorna mediante parâmetros do método.


Simple Factory

Exemplo de aplicação

```
public class StrategyFactory {  
    /**  
     * Documentation is critical. Say what `types` are supported.  
     */  
    public static Strategy create(String type) {  
        switch(type.toLowerCase()) {  
            case "diversity": return new StrategyDiversity();  
            case "multiskill": return new StrategyMultiSkill();  
            case "senior": return new StrategySenior();  
            case "specialized": return new StrategySpecialized();  
            default: return null; /* or throw exception */  
        }  
    }  
}
```

💡 Este exemplo incide sobre o código da aula acerca do padrão *strategy*; pode aplicá-lo diretamente (exercício autónomo)

Simple Factory

 Participantes do padrão neste exemplo:

- **Factory:** `StrategyFactory`
- **Product:** `Strategy`
- **Concrete Product:** `StrategyDiversity`, `StrategyMultiSkill`, `StrategySenior` e `StrategySpecialized`

! Observações gerais:

- O tipo do *produto* retornado pelo método é **sempre** um *super-tipo* (interface ou classe abstrata) dos *produtos concretos*.
- Note que o valor de `type` pode vir de algum *user input*, e.g., comando terminal ou *combobox*.

Problema real

- As publicações científicas contêm uma secção de bibliografia, a qual reporta a outros trabalhos citados no corpo do documento.
- Para publicar, o autor segue um estilo de formatação determinado pela editora; as citações, consequentemente, também seguem um formato específico.
- Existem diferentes formatos de citação, e.g., IEEE, APA, Chicago, etc.:
 - <https://pitt.libguides.com/citationhelp>

Problema real

- Exemplo de citações IEEE:

Type	In-text citation	Formatted Citation
Book	[1]	D. Sarunyagate, Ed., Lasers. New York: McGraw-Hill, 1996.
Chapter in book	[2]	G. O. Young, "Synthetic structure of industrial plastics," in Plastics, 2nd ed., vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15-64.
Journal article	[4]	G. Liu, K. Y. Lee, and H. F. Jordan, "TDM and TWDM de Bruijn networks and shufflenets for optical communications," IEEE Trans. Comp., vol. 46, pp. 695-701, 1997.

Problema real

- Exemplo de citações APA:

Type	In-text citation	Formatted Citation
Book	(Sapolsky, 2017)	Sapolsky, R. M. (2017). Behave: The biology of humans at our best and worst. Penguin Books.
Chapter in book	(Dillard, 2020)	Dillard, J. P. (2020). Currents in the study of persuasion. In M. B. Oliver, A. A. Raney, & J. Bryant (Eds.), Media effects: Advances in theory and research (4th ed., pp. 115–129). Routledge.
Journal article	(Weinstein, 2009)	Weinstein, J. (2009). "The market in Plato's Republic." Classical Philology, 104(4), 439-458.

Problema real

O seguinte repositório contém a modelação deste problema e respetivas aplicações dos vários padrões *fábrica*:

https://github.com/brunomnsilva/JavaPatterns_Factories

O ficheiro `README.md` faz uma descrição resumida do conjunto de classes que suportam a modelação deste problema:

- Veja o *package* `model` (neste momento).

Simple Factory

⚠ *Package* `variants.simplefactory`

O padrão **simple factory** traduzido para este problema pode ser implementado da seguinte forma:

```
public class IEEE CitationStyleFactory {  
    /**  
     * Documentação omitida. Veja repositório.  
     * É de extrema importância para informar de como são solicitados os produtos à fábrica.  
     */  
    public static Citation create(String type, String... args) {  
        switch(type.toLowerCase()) {  
            case "book":  
                return new IEEEBookCitation(args[0], args[1], args[2], args[3], args[4]);  
            case "bookchapter":  
                return new IEEEBookChapterCitation(args[0], args[1], args[2], args[3], args[4],  
                                                    args[5], args[6], args[7]);  
            case "journal":  
                return new IEEEJournalCitation(args[0], args[1], args[2], args[3], args[4], args[5]);  
            default:  
                throw new UnsupportedOperationException("Type not supported: " + type);  
        }  
    }  
}
```

Simple Factory

A utilização da fábrica pelo *cliente*:

```
List<Citation> bibliography = new ArrayList<>();

Citation citation1 = IEEEECitationStyleFactory.create("book", "D. Sarunyagate", "Lasers",
    "New York", "McGraw-Hill", "1996");

Citation citation2 = IEEEECitationStyleFactory.create("bookchapter", "G. O. Young",
    "Synthetic structure of industrial plastics", "Plastics, 2nd ed",
    "J. Peters", "New York", "McGraw-Hill", "15-64", "1996");

Citation citation3 = IEEEECitationStyleFactory.create("journal", "G. Liu, K. Y. Lee, and H. F. Jordan",
    "TDM and TWDM de Bruijn networks and shufflenets for optical communications",
    "EEE Trans. Comp.", "46", "695-701", "1997");

bibliography.add( citation1 );
bibliography.add( citation2 );
bibliography.add( citation3 );

for(Citation c : bibliography) {
    System.out.println(c.toStringFormatted());
}
```

- Execute este exemplo da classe `MainSimpleFactory`.

Simple Factory

- Note que o *cliente* apenas trabalha com a interface dos produtos (i.e., participante *product*), embora a fábrica devolva produtos concretos (i.e., *concrete product*) cujo tipo depende da solicitação efetuada.
- O código faz menção explícita a `IEEECitationStyleFactory`, pelo que o padrão apenas contempla um estilo de citação.
- ⚠ Este padrão é utilizado quando existe apenas **uma família** de produtos (situação mais frequente).

Factory Method

- Proposto em *Design Patterns: Elements of Reusable Object-Oriented Software* (1994)

THE 23 GANG OF FOUR DESIGN PATTERNS

<div>C</div> Abstract Factory	<div>S</div> Facade	<div>S</div> Proxy
<div>S</div> Adapter	<div>C</div> Factory Method	<div>B</div> Observer
<div>S</div> Bridge	<div>S</div> Flyweight	<div>C</div> Singleton
<div>C</div> Builder	<div>B</div> Interpreter	<div>B</div> State
<div>B</div> Chain of Responsibility	<div>B</div> Iterator	<div>B</div> Strategy
<div>B</div> Command	<div>B</div> Mediator	<div>B</div> Template Method
<div>S</div> Composite	<div>B</div> Memento	<div>B</div> Visitor
<div>S</div> Decorator	<div>C</div> Prototype	

Factory Method

- É uma evolução do padrão **simple factory**.

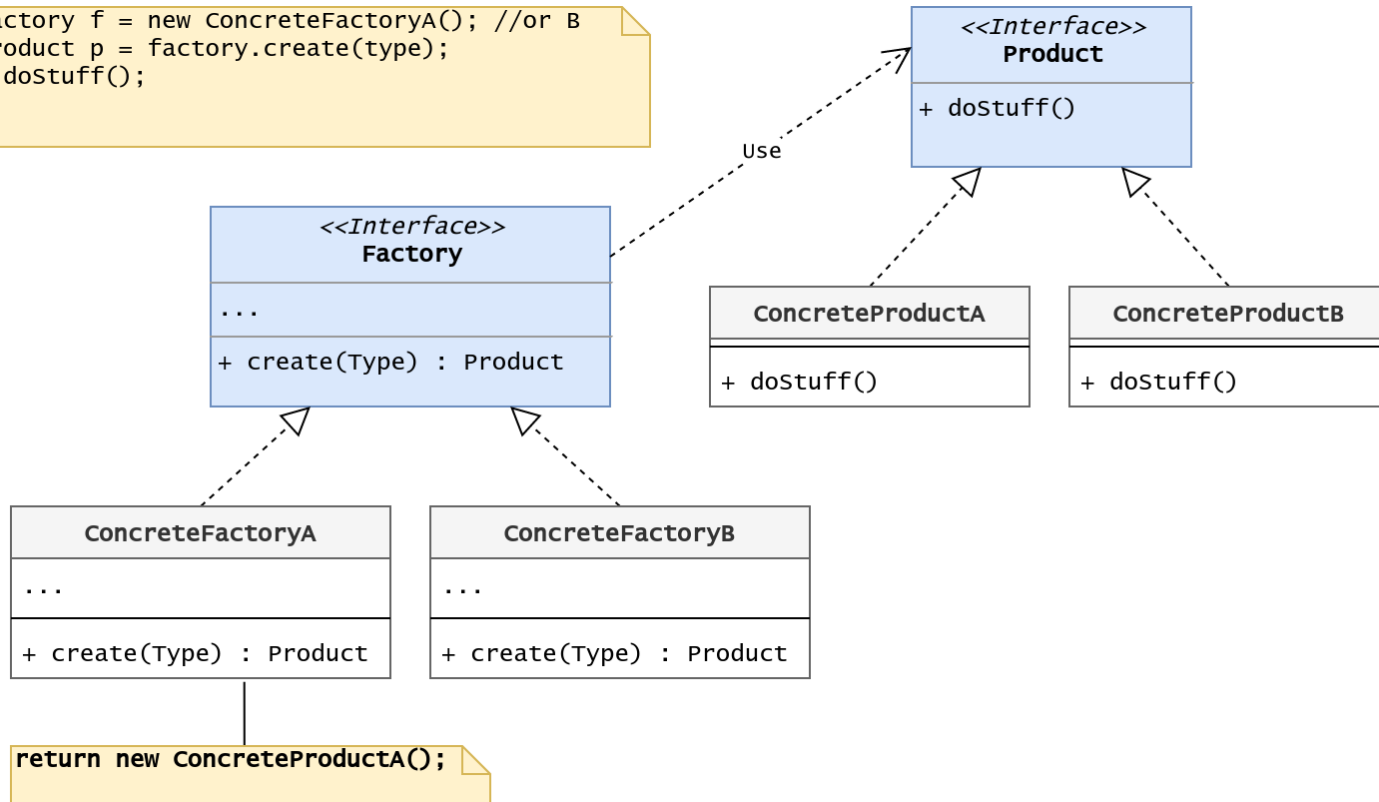
Motivação

- Torna possível e flexível a existência de fábricas alternativas para diferentes *famílias* de produtos.
 - (mas que partilham a mesma *interface*)

Factory Method

Diagrama de classes:

```
Factory f = new ConcreteFactoryA(); //or B
Product p = factory.create(type);
p.doStuff();
```



Factory Method

Participantes e responsabilidades:

- **Product:** define a interface comum a todos os objetos que podem ser produzidos pela fábrica e suas implementações.
- **Concrete products:** diferentes implementações da interface *product*, segmentadas por "famílias".
- **Factory:** declara o método de criação que retorna produtos. É importante que o tipo de retorno deste método coincida com o tipo da interface *product*.
- **Concrete factories:** implementam o método de criação de forma a retornar diferentes *famílias* de produtos.

Factory Method



⚠ *Package* `variants.factorymethod`

No nosso exemplo torna possível a existência das *famílias* de produtos *IEEE* e *APA*; outras podem ser adicionadas no futuro.

Mapeamento de participantes

- **Product:** `Citation`
- **Concrete products:**
 - *família IEEE*: `IEEEBookCitation`, `IEEEBookChapterCitation` e `IEEEJournalCitation`
 - *família APA*: `APABookCitation`, `APABookChapterCitation` e `APAJournalCitation`
- **Factory:** `CitationStyleFactory`
- **Concrete factories:** `IEEECitationStyleFactory` e `APACitationStyleFactory`

Factory Method

Interface `CitationStyleFactory` (participante **Factory**):

```
public interface CitationStyleFactory {  
    /**  
     * Documentação omitida. Ver repositório.  
     */  
    Citation create(String type, String ... args);  
}
```

⚠ O *cliente* utiliza a interface da fábrica (ver `MainFactoryMethod`):

```
public class MainFactoryMethod {  
    public static void main(String[] args) {  
        CitationStyleFactory factory = new IEEECitationStyleFactory();  
        //CitationStyleFactory factory = new APACitationStyleFactory();  
  
        Citation citation1 = factory.create("book", "D. Sarunyagate", "Lasers",  
                                           "New York", "McGraw-Hill", "1996");  
        //...  
    }  
}
```

Factory Method



Classes `APACitationStyleFactory` e `IEEECitationStyleFactory`
(participantes **Concrete factories**):

```
public class APACitationStyleFactory implements CitationStyleFactory {
    @Override
    public Citation create(String type, String... args) {
        switch(type.toLowerCase()) {
            case "book":
                return new APABookCitation(args[0], args[1], args[2], args[3], args[4]);
            case "bookchapter":
                return new APABookChapterCitation(args[0], args[1], args[2], args[3], args[4], args[5], args[6], args[7]);
            case "journal":
                return new APAJournalCitation(args[0], args[1], args[2], args[3], args[4], args[5]);
            default:
                throw new UnsupportedOperationException("Type not supported: " + type);
        }
    }
}
```

```
public class IEEECitationStyleFactory implements CitationStyleFactory {
    @Override
    public Citation create(String type, String... args) {
        switch(type.toLowerCase()) {
            case "book":
                return new IEEEBookCitation(args[0], args[1], args[2], args[3], args[4]);
            case "bookchapter":
                return new IEEEBookChapterCitation(args[0], args[1], args[2], args[3], args[4], args[5], args[6], args[7]);
            case "journal":
                return new IEEEJournalCitation(args[0], args[1], args[2], args[3], args[4], args[5]);
            default:
                throw new UnsupportedOperationException("Type not supported: " + type);
        }
    }
}
```

Factory Method

- O *cliente* apenas trabalha com as interfaces da fábrica e dos produtos.
 - Torna flexível a utilização de diferentes fábricas concretas sem alterar código do cliente.
 - Pode até haver uma "fábrica de fábricas": **simple factory** para a criação das fábricas concretas 😊
- ⚠ Este padrão é utilizado quando existem **várias famílias** de produtos.

Abstract Factory

- Proposto em *Design Patterns: Elements of Reusable Object-Oriented Software* (1994)

THE 23 GANG OF FOUR DESIGN PATTERNS

<div>C</div> Abstract Factory	<div>S</div> Facade	<div>S</div> Proxy
<div>S</div> Adapter	<div>C</div> Factory Method	<div>B</div> Observer
<div>S</div> Bridge	<div>S</div> Flyweight	<div>C</div> Singleton
<div>C</div> Builder	<div>B</div> Interpreter	<div>B</div> State
<div>B</div> Chain of Responsibility	<div>B</div> Iterator	<div>B</div> Strategy
<div>B</div> Command	<div>B</div> Mediator	<div>B</div> Template Method
<div>S</div> Composite	<div>B</div> Memento	<div>B</div> Visitor
<div>S</div> Decorator	<div>C</div> Prototype	

Abstract Factory

- É uma evolução do padrão **factory method**.

Motivação

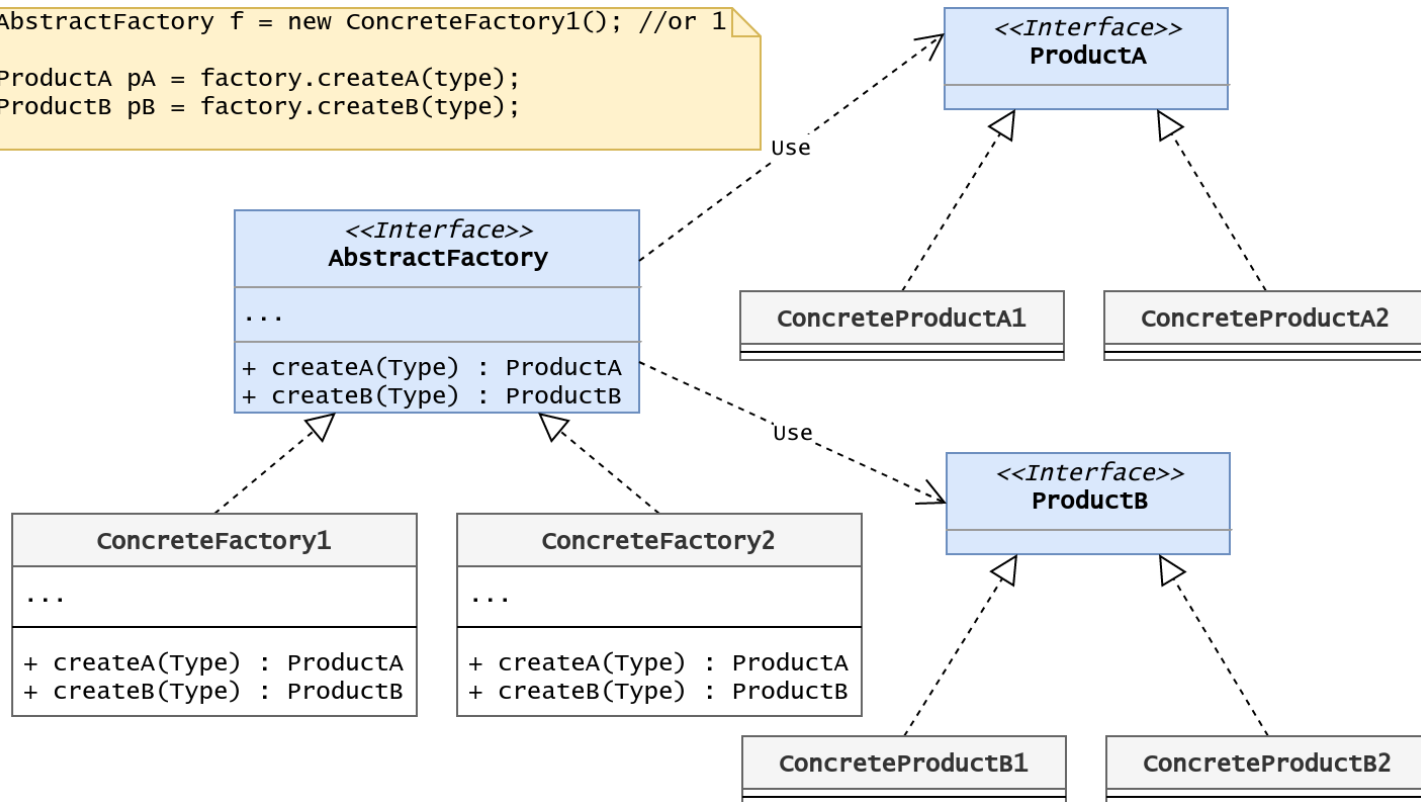
- Permite que uma fábrica produza produtos de tipos diferentes, mas "relacionados", segmentados por *famílias*.

Abstract Factory

Diagrama de classes:

```
AbstractFactory f = new ConcreteFactory1(); //or 1
```

```
ProductA pA = factory.createA(type);  
ProductB pB = factory.createB(type);
```



Abstract Factory

Participantes e responsabilidades:

- **Abstract product:** definem as interfaces de produtos distintos, mas relacionados.
- **Concrete products:** diferentes implementações das interfaces *abstract products*, divididos por "famílias".
- **Abstract Factory:** declara os métodos de criação para cada *abstract product*
- **Concrete factories:** implementam os métodos de criação de forma a retornarem as variantes de produtos dentro da mesma família.

Abstract Factory

⚠ *Package* `variants.abstractfactory`

No exemplo fornecido, para além de permitir criar instâncias de `Citation`, também permite criar o *gestor de bibliografia* (`BibliographyManager`) correspondente.

- O *gestor de bibliografia* fará o output completo, i.e., juntamente com a *in-text citation* correspondente ao estilo de citação:

[1] D. Sarunyagate, Lasers, New York: McGraw-Hill, 1996.

ou

(D. Sarunyagate, 1996) D. Sarunyagate (1996). Lasers. McGraw-Hill.

Abstract Factory

Mapeamento de participantes

- **Abstract products:** `Citation` e `BibliographyManager`
- **Concrete products:**
 - *família IEEE: citações* `IEEEBookCitation` ,
`IEEEBookChapterCitation` e `IEEEJournalCitation` ; **gestor**
`IEEEBibliographyManager`
 - *família APA: citações* `APABookCitation` , `APABookChapterCitation`
e `APAJournalCitation` ; **gestor** `APABibliographyManager`
- **Abstract factory:** `CitationStyleFactory`
- **Concrete factories:** `IEEECitationStyleFactory` e
`APACitationStyleFactory`

Abstract Factory

Interface `CitationStyleFactory` (participante **Abstract Factory**):

```
public interface CitationStyleFactory {  
    /**  
     * Documentação omitida. Ver repositório.  
     */  
    Citation createCitation(String type, String ... args);  
  
    /**  
     * Returns the respective bibliography manager for the family of citations.  
     * @return a bibliography manager.  
     */  
    BibliographyManager createManager();  
}
```

- Os nomes dos métodos de criação verbalizam que tipo de objeto será produzido dentro da variante, e.g., IEEE ou APA.

Abstract Factory

⚠ O *cliente* utiliza apenas as interface da fábrica e dos produtos abstratos (ver `MainAbstractFactory`):

```
public class MainFactoryMethod {  
    public static void main(String[] args) {  
        CitationStyleFactory factory = new IEEECitationStyleFactory();  
        //CitationStyleFactory factory = new APACitationStyleFactory();  
  
        Citation citation1 = factory.createCitation("book", "D. Sarunyagate", "Lasers",  
            "New York", "McGraw-Hill", "1996");  
        //...  
  
        BibliographyManager manager = factory.createManager();  
  
        manager.add(citation1);  
        //...  
  
        System.out.println(manager.output());  
    }  
}
```

Exercícios

No repositório fornecido existe código em falta:

1. A implementação de `toStringFormatted()` em `APABookCitation`, `APABookChapterCitation` e `APAJournalCitation`. Implemente de acordo com os seguintes formatos de citação:
 - **Book:** `[author] ([year]). [title]. [publisher].`
 - **Book chapter:** `[author] ([year]). [title]. In [editor] (Eds.), [book_title]. (pp. [pages]). [publisher].`
 - **Journal:** `[author] ([year]). "[title]." [journal_title], [volume_issue], [pages].`
2. A implementação de `IEEECitationStyleFactory.getManager()` e o correspondente *produto concreto*, i.e., `IEEEBibliographyManager`.
3. Execute os exemplos comutando a fábrica utilizada.

Prós e contras

- ✓ Promove-se o fraco acoplamento de classes entre o *cliente* e os *produtos concretos*
- ✓ *Single Responsibility Principle*. O código de criação dos produtos está centralizado num único sítio do código.
- ✓ *Open/Closed Principle*. É fácil adicionar novos tipos de produtos sem "quebrar" o código existente.
- ✗ O código pode tornar-se mais complicado, dado que é necessário introduzir várias interfaces e subclasses para implementar os padrões.

Bibliografia

- *Head First Design Patterns* (2004)
- *Design Patterns: Elements of Reusable Object-Oriented Software* (1994)
- <https://refactoring.guru/design-patterns/factory-comparison>
 - Comparação e contém referências para os diferentes padrões fábrica.