Engenharia de Software

11/8/2025

TP1

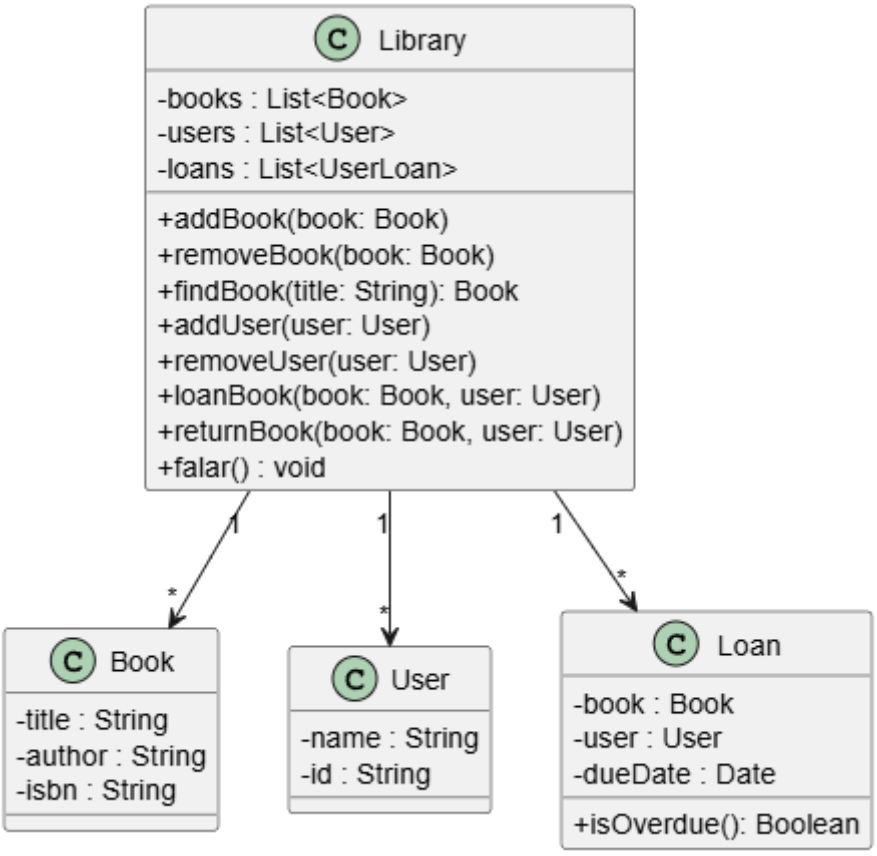
Design Patterns eDomain-Driven Design(DDD) com Java

Professor(a): Armênio Torres Santiago Cardoso

Samuel Hermany
INSTITUTO INFNET

LINK GITHUB

ORIGINAL



1. DESACOPLAMENTO E SEPARAÇÃO DE RESPONSABILIDADES - ETAPA01

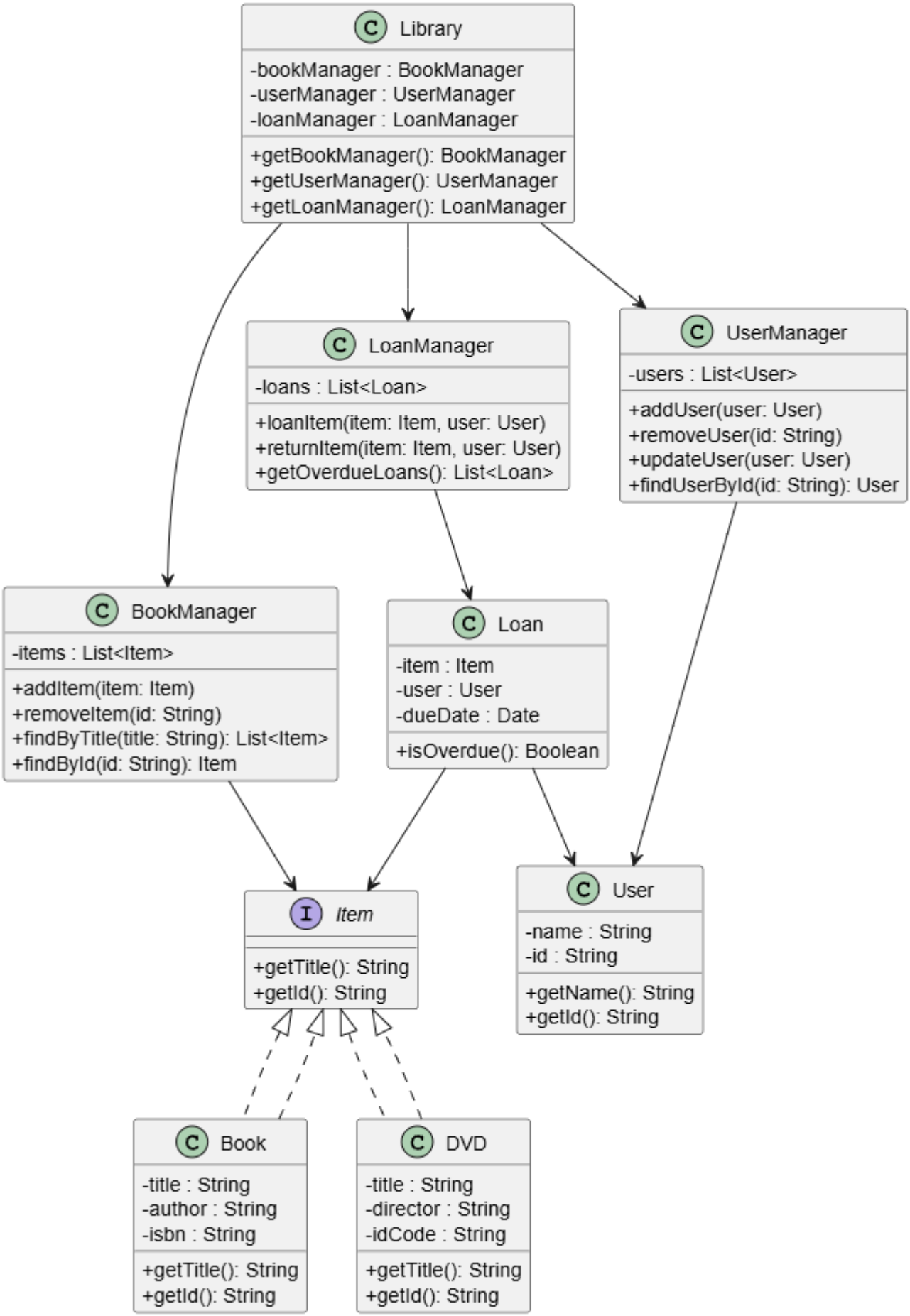
Aqui na refatoração inicial divide-se a Library em managers e usar Item como abstração para livros e outros itens.

1. Extrai-se os managers(gerentes):
2. BookManager cuida apenas de itens (livros, DVDs…).
3. UserManager cuida apenas dos usuários.
4. LoanManager cuida apenas dos empréstimos.
5. Library agora coordena tudo, mas não tem listas nem lógica interna.

6. Criada a interface para o Item:
7. Permite adicionar novos tipos de item no futuro (ex.: Magazine) sem alterar os managers.
8. Book e DVD implementam Item.

9. Reduzido o acoplamento:
10. Loan não conhece Book ou DVD diretamente, apenas Item.
11. Library conhece apenas os managers, não os detalhes internos.

12. Agora o diagrama está pronto para inserir Factory, Strategy, Observer e Facade, pois o código está organizado.


Library
-bookManager : BookManager
-userManager : UserManager
-loanManager : LoanManager
+getBookManager(): BookManager
+getUserManager(): UserManager
+getLoanManager(): LoanManager
LoanManager
-loans : List<Loan>
+loanItem(item: Item, user: User)
+returnItem(item: Item, user: User)
+getOverdueLoans(): List<Loan>
UserManager
-users : List<User>
+addUser(user: User)
+removeUser(id: String)
+updateUser(user: User)
+findUserById(id: String): User
BookManager
-items : List<Item>
+addItem(item: Item)
+removeItem(id: String)
+findByTitle(title: String): List<Item>
+findById(id: String): Item
Loan
-item : Item
-user : User
-dueDate : Date
+isOverdue(): Boolean
Item
+getTitle(): String
+getId(): String
User
-name : String
-id : String
+getName(): String
+getId(): String
Book
-title : String
-author : String
-isbn : String
+getTitle(): String
+getId(): String
DVD
-title : String
-director : String
-idCode : String
+getTitle(): String
+getId(): String

2. PADRÕES BÁSICOS – ETAPA02

Aqui foi aplicado padrões de projeto básicos em cima da estrutura da ETAPA 01.

1. ItemFactory + ConcreteItemFactory: centralizam a criação de livros, DVDs e futuros itens (Factory Method).

```
abstract class ItemFactory
class ConcreteItemFactory extends ItemFactory
```

- Acima centraliza-se a criação dos objetos Item (como Book, DVD).
- A interface/abstração **ItemFactory** define o método **createItem(type: String, data: Map): Item**.

- A classe **ConcreteItemFactory** implementa esse método e decide qual tipo concreto criar (ex: Book, DVD), isolando essa decisão do resto do sistema.

- Benefício: Facilita a adição de novos tipos de item sem modificar clientes, seguindo o **Open/Closed Principle**.

2. SearchStrategy: encapsulam diferentes buscas, podendo trocar dinamicamente o algoritmo usado (Strategy).

```
interface SearchStrategy
class TitleSearchStrategy implements SearchStrategy
class AuthorSearchStrategy implements SearchStrategy
class BookManager
```

- BookManager tem um atributo searchStrategy do tipo SearchStrategy.
- Diferentes classes concretas (ex: TitleSearchStrategy) implementam algoritmos variados de busca.
- BookManager delega a busca para a estratégia configurada via setSearchStrategy().

3. LoanManager com Observer: mantém lista de observadores e notifica eventos de empréstimos (Observer).

```
interface Observer
interface Subject
class LoanManager implements Subject
```

- LoanManager mantém uma lista de observers (addObserver(), removeObserver() e notifyObservers()).

  Como funciona:
- Quando um empréstimo é criado, devolvido ou fica vencido, LoanManager chama notifyObservers().
- Qualquer classe que implemente Observer pode se inscrever para receber notificações (ex: sistema de alertas).

  Resultado:
- Desacopla-se o mecanismo de notificação da lógica principal, facilitando extensão e manutenção.

4. LibraryFacade: camada simplificada que esconde a complexidade dos managers e factory, servindo como interface para clientes (Facade).

```
class LibraryFacade
```

- LibraryFacade é a única interface que clientes externos usam para interagir com o sistema.

- Esconde a complexidade dos managers, factory e outras classes internas.

Como funciona:

- Fornece métodos simples (ex: addBook(), searchBooks(), loanItem()) que delegam para os managers apropriados.

Resultado:

- Simplifica o uso do sistema, diminui acoplamento entre clientes e o núcleo

5. Library com static instance: implementação do Singleton para garantir que só exista uma instância do sistema.

```
class Library {
   - static instance : Library
   + getInstance(): Library
   ...
}
```

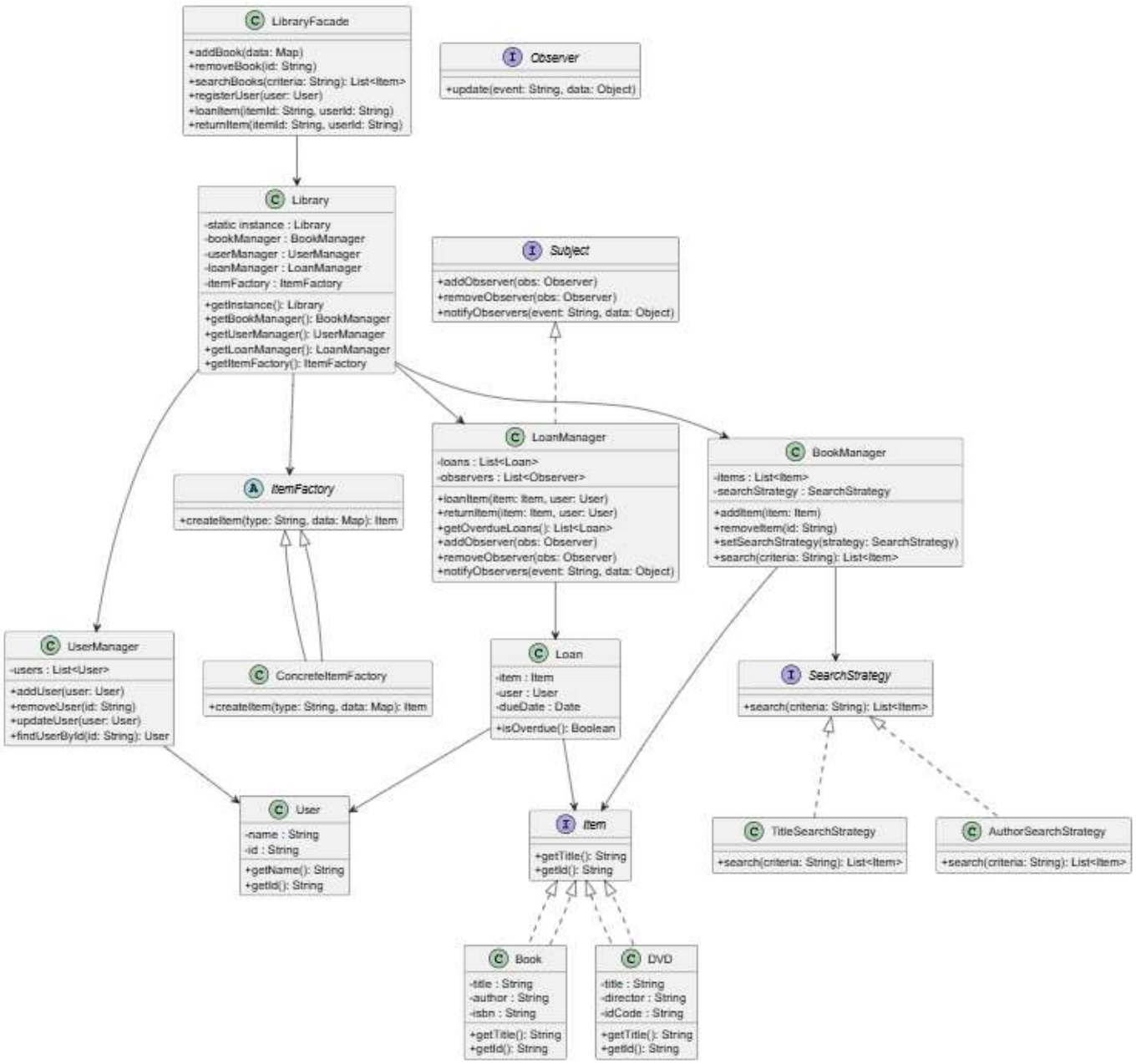
- Library tem um atributo estático instance e um método getInstance() que garante criação única.

Como Funciona:

- Só existe uma instância de Library no sistema, que coordena todos os managers e a factory.

Resultado:

- Garante consistência e ponto único de controle da biblioteca.



3. PADRÕES BÁSICOS – ETAPAFINAL

Como funciona o fluxo:

- Cliente chama método no LibraryFacade (ex: addBook(data)).
- Facade usa o ItemFactory para criar o item correto (Book ou DVD).
- O item é passado para o BookManager, que gerencia o armazenamento e buscas (usando uma estratégia configurada).
- Empréstimos são feitos via LoanManager, que notifica observadores inscritos (como alertas de empréstimos vencidos).
- Tudo isso acontece coordenado pela única instância de Library (Singleton).

Resumo da Arquitetura Final:

Padrão	Onde	Benefício Principal
Factory	`ItemFactory`	Criação centralizada e extensível de itens.
Strategy	`SearchStrategy` + `BookManager`	Busca flexível e intercambiável.
Observer	`LoanManager` + `Observer`	Notificações desacopladas para eventos de empréstimo.
Facade	`LibraryFacade`	Interface única e simplificada para o sistema.

Padrão	Onde	Benefício Principal
Singleton	`Library`	Garantia de instância única e ponto central de controle.

1. Interfaces e Itens (Item, Book, DVD)

- **Item (interface)** define a API comum para qualquer coisa que pode ser emprestada (getTitle(), getId()).
- **Book** e **DVD** implementam Item, encapsulando propriedades específicas (autor, diretor, ISBN, etc).
- **Por quê?**
  - Permite o sistema trabalhar genericamente com diferentes tipos de itens.
  - Facilita futura extensão (ex: revista, jogo) sem mexer nos managers.

---

2. Factory Method (ItemFactory, ConcreteItemFactory)

- ItemFactory define o método abstrato createItem(type, data).
- ConcreteItemFactory implementa esse método, criando objetos Book, DVD conforme tipo.
- **Por quê?**
  - Centraliza a criação de objetos, reduz acoplamento e facilita adição de novos tipos.
  - Clientes (como LibraryFacade) não precisam saber detalhes de construção.

---

3. Strategy para Busca (SearchStrategy e Implementações)

- SearchStrategy define search(criteria) que retorna lista de itens.
- TitleSearchStrategy e AuthorSearchStrategy implementam diferentes algoritmos de busca.
- BookManager mantém uma referência para a estratégia atual (searchStrategy).
- **Por quê?**
  - Permite alterar dinamicamente o algoritmo de busca sem alterar o manager.
  - Evita duplicação e acoplamento de lógica de busca dentro do manager.

---

4. Observer para Empréstimos (LoanManager e Interfaces)

- LoanManager implementa Subject, gerenciando lista de ObserverS.
- Métodos para adicionar, remover e notificar observadores (addObserver, notifyObservers, etc).
- Loan representa o empréstimo (item, usuário, data).
- **Por quê?**
  - Desacopla notificações (ex: empréstimo vencido, devolução) da lógica de empréstimo.
  - Permite implementar vários observadores para diferentes ações (email, alertas, logs).

---

5. Usuários e Managers (User, BookManager, UserManager)

- User encapsula dados do usuário.
- BookManager gerencia a coleção de itens (lista books : List<Item>) e realiza buscas usando SearchStrategy.

- UserManager gerencia a lista de usuários e suas operações (add, remove, update).
- **Por quê?**
  - Separa claramente as responsabilidades, reduzindo acoplamento e facilitando manutenção.
  - Cada manager é responsável por uma parte do sistema.

---

6. Facade e Singleton (LibraryFacade e Library)

- **LibraryFacade** é a interface simples e única para os clientes usarem o sistema (adicionar livro, buscar, registrar usuário, emprestar, devolver).
- **Library** é o núcleo do sistema, implementando o padrão Singleton para garantir uma única instância, e contém referências para os managers e factory.
- **Por quê?**
  - ***Facade*** simplifica e isola os clientes da complexidade interna do sistema.
  - ***Singleton*** garante integridade do estado, evitando múltiplas instâncias conflitantes.