Análise e Projeto Orientados por Objetos

Aula 05 – Padrões GoF (Singleton e Iterator)

Edirlei Soares de Lima
<edirlei@iprj.uerj.br>

Padrões GoF

- Criação:
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - **Singleton**
- Estruturais:
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- Comportamentais:
  - Chain of Responsibility
  - Command
  - Interpreter
  - **Iterator**
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

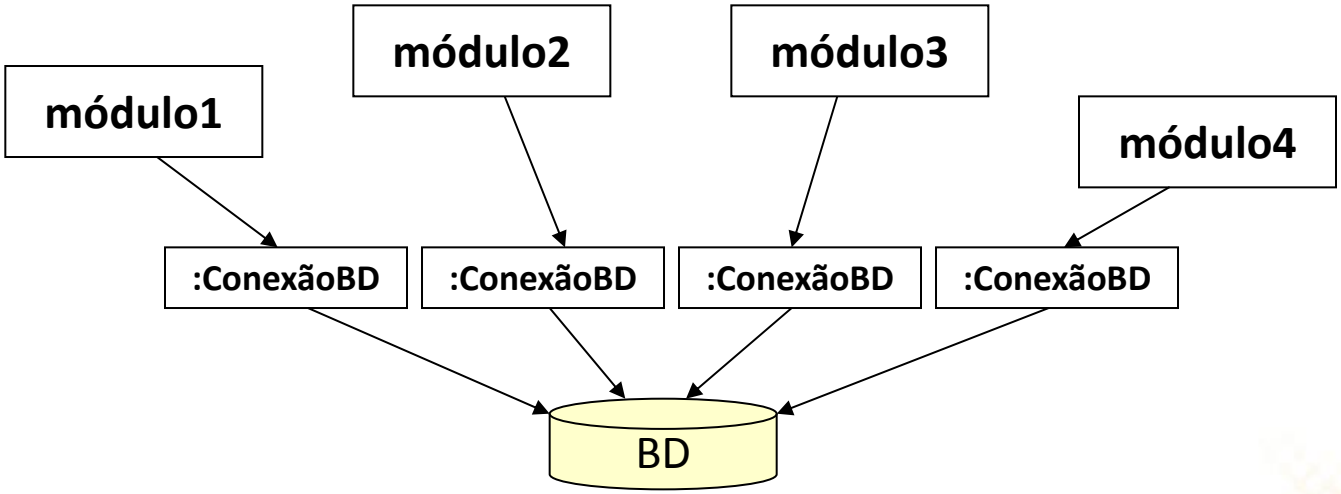
Singleton

- **Motivação:** algumas classes devem ser instanciadas uma única vez:
  - Um spooler de impressão;
  - Um sistema de arquivos;
  - Um Window manager;
  - Um objeto que contém a configuração do programa;
  - Um ponto de acesso ao banco de dados;

- **Obstáculo:** a definição de uma variável global deixa a instância (objeto) acessível mas não inibe a instanciação múltipla. Como assegurar que somente uma instância de uma classe seja criada para toda a aplicação?

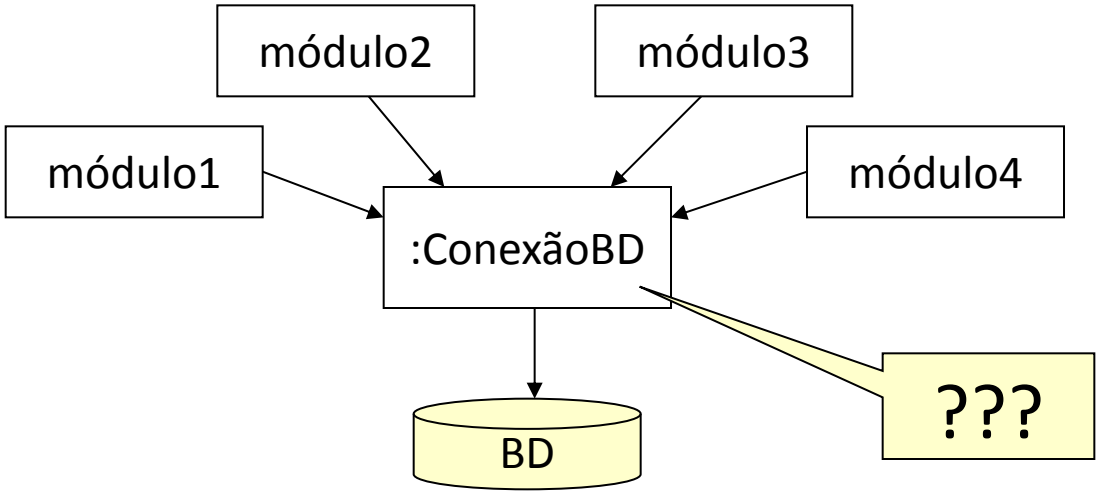
Singleton – Exemplo

- **Exemplo:**
  - Vários módulos do sistema utilizam o mesmo banco de dados;
  - A classe ConexãoBD dá acesso ao banco;



Singleton – Exemplo

- E se você quiser garantir que haja apenas uma conexão para todos os módulos?
  - Como garantir que haja apenas uma instância de ConexãoBD?

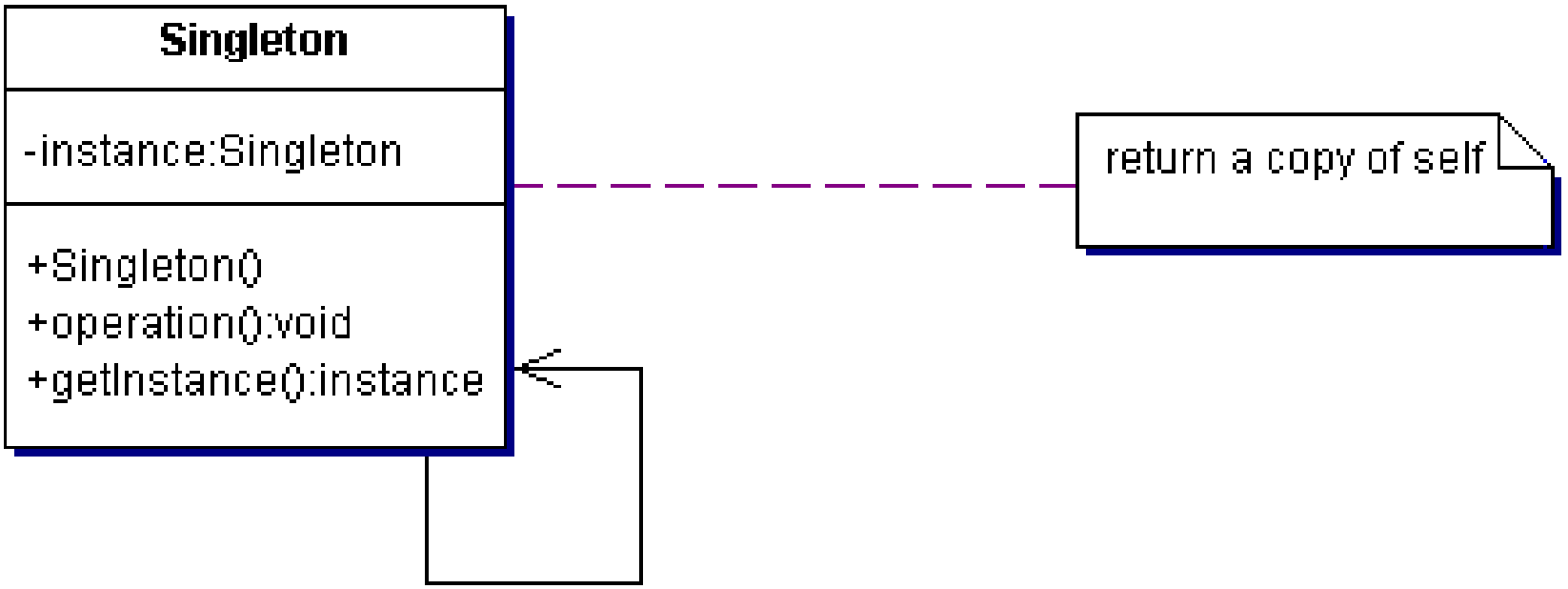


Singleton

- **Intenção:** garantir que uma classe tem apenas uma instância, e prover um ponto de acesso global a ela;

- **Solução:** fazer com que a própria classe seja responsável pela manutenção da instância única, de tal forma que:
  - Quando a instância for requisitada pela primeira vez, essa instância deve ser criada;
  - Em requisições subsequentes, a instância criada na primeira vez é retornada.
  - A classe Singleton deve:
    - armazenar a única instância existente;
    - garantir que apenas uma instância será criada;
    - prover acesso a tal instância.

Singleton


Singleton
-instance:Singleton
+Singleton()
+operation():void
+getInstance():instance
return a copy of self

Singleton – Implementação

```
public class Singleton
{
  private static Singleton instance = null;

  private Singleton ()
  {
    ...
  }

  public static Singleton getInstance()
  {
    if (instance == null)
    {
      instance = new Singleton();
    }
    return instance;
  }
  ...
}
```

Singleton – Implementação

```
public class UsoDoSingleton
{
   .
   .
   .
   Singleton obj;
   .
   .
   .
   obj = Singleton.getInstance();
   .
   .
   .
}
```

Iterator

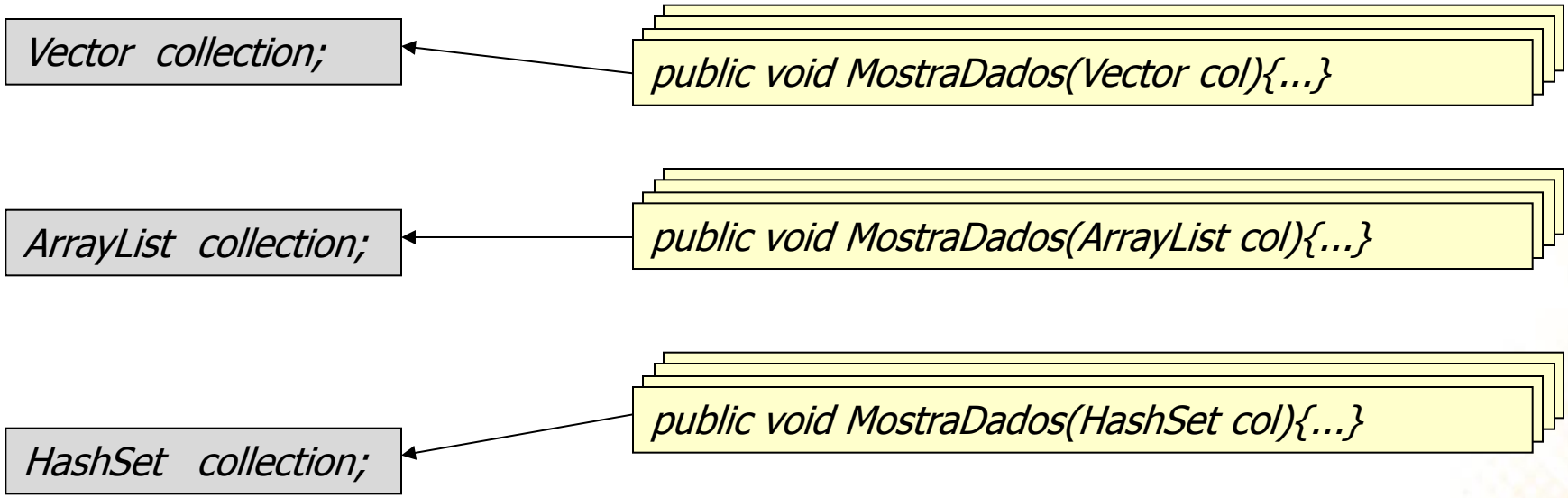
- Toda coleção possui uma **representação interna** para o armazenamento e organização de seus elementos.
  - Por outro lado, essa coleção deve permitir que seus elementos sejam acessados sem que sua estrutura interna seja exposta.

- Pode-se desejar que estes elementos sejam percorridos de várias maneira, sem no entanto ter que modificar a interface da coleção em função do tipo de varredura desejado.
  - de frente para trás, vice-versa, ou mesmo em ordem aleatória.

- O padrão Iterator permite descrever uma forma de percorrer os elementos de uma coleção sem violar o encapsulamento dessa coleção.

Iterator – Exemplo

- Utilização de diferentes estruturas de dados: *Vector*, *ArrayList*, *HashSet*, ...



- E se for necessário mudar a estrutura de dados?

Iterator

- **Intenção:** iterar sobre (percorrer sequencialmente) uma coleção de objetos sem expor sua representação.
  - Obedecendo o princípio do encapsulamento

- **Solução:** um objeto intermediário (iterator) é usado entre o cliente e a coleção de objetos.
  - Este objeto **conhece a estrutura interna** da coleção a ser percorrida, e apresenta uma interface para percorrer tal estrutura.
  - Esta interface é **independente dessa estrutura interna**.
  - Os clientes que desejam percorrer a coleção **utilizam a interface do objeto intermediário**, em vez de se comunicarem diretamente com a coleção de objetos.

Iterator

- Requisitos de um iterador:
  - Um modo de **localizar um elemento** específico da coleção, tal como o primeiro elemento;
  - Um modo de obter **acesso ao elemento atual**;
  - Um modo de obter o **próximo elemento**;
  - Um modo de **indicar que não há mais elementos** a percorrer.

- Exemplo em Java:
  - As classes List, Set e Hash são subclasses de Collection, e herdam um método iterator() que retorna um objeto iterador.
  - O objeto Iterator possui métodos hasNext() e next().

Iterator

```
// ICollection.java
// interface para obtenção de Iterator para coleções

public interface ICollection
{
  // obtenção de um Iterator
  public IIterator getIterator();

  // determina existência de um elemento
  public boolean has(Object object);

  // adição de um elemento
  public boolean add(Object object);

  // remoção de um elemento
  public boolean remove(Object object);

  // remoção de todos os elementos
  public void removeAll();
}
```

Iterator

```
// IIterator.java
public interface IIterator {

  // verifica a existência de um próximo elemento
  public boolean hasNext();

  // retorna o próximo elemento
  public Object next();
}
```

Iterator – Uso

```
public static void main(String[] args)
{
  ArrayList<String> lista = new ArrayList();
  lista.add("A");
  lista.add("B");
  lista.add("C");
  lista.add("D");
  lista.add("E");

  Iterator iterator = lista.iterator();
  while(iterator.hasNext())
  {
    String str = (String)iterator.next();
    System.out.print(str + " ");
  }
  System.out.println();
}
```

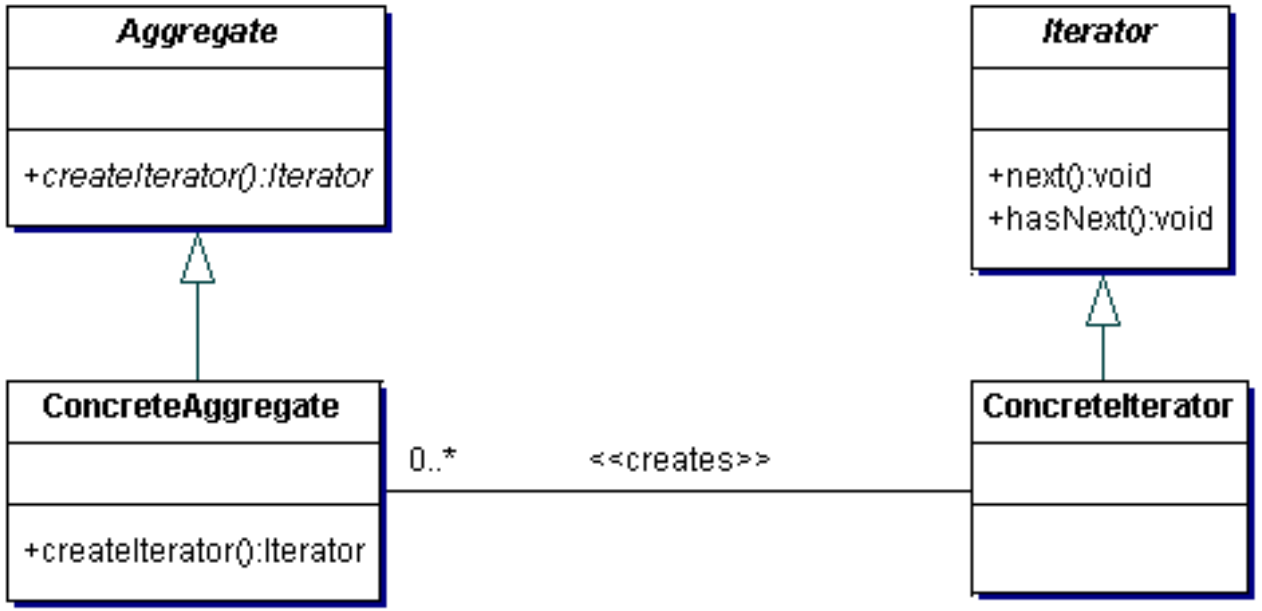
Iterator – Uso

```
public static void main(String[] args)
{
  ArrayList<String> lista = new ArrayList();
  lista.add("A");
  lista.add("B");
  lista.add("C");
  lista.add("D");
  lista.add("E");

  for(String str:lista)
  {
    System.out.print(str + " ");
  }
  System.out.println();

}
```

Iterator


Aggregate
+createIterator():Iterator
Iterator
+next():void
+hasNext():void
ConcreteAggregate
+createIterator():Iterator
0..*
<<creates>>
ConcreteIterator

Iterator – Implementação

```
public class MinhaLista implements Iterable<Integer>
{
    private int[] arrayList;
    private int currentSize;

    public MinhaLista(int[] newArray)
    {
        this.arrayList = newArray;
        this.currentSize = arrayList.length;
    }

    @Override
    public Iterator<Integer> iterator()
    {
        Iterator<Integer> it = new Iterator<Integer>()
        {
            private int currentIndex = 0;

            ...
```

Iterator – Implementação

```
            @Override
            public boolean hasNext()
            {
                return currentIndex < currentSize;
            }
            @Override
            public Integer next()
            {
                int v = arrayList[currentIndex];
                currentIndex+=2;
                return v;
            }
            @Override
            public void remove()
            {
                throw new UnsupportedOperationException();
            }
        };
        return it;
    }
}
```

Iterator – Implementação

```
public static void main(String[] args)
{
   int[] numeros = new int[]{1,2,3,4,5,6,7};
   MinhaLista lista = new MinhaLista(numeros);

   for(int val:lista)
   {
       System.out.print(val + " ");
   }
   System.out.println();

   Iterator iterator = lista.iterator();
   while(iterator.hasNext())
   {
       int val = (int)iterator.next();
       System.out.print(val + " ");
   }
   System.out.println();
}
```

Iterator – Aplicabilidade

- O uso do padrão Iterator se aplica quando se quer:
  - Acessar o conteúdo de objeto agregados **sem expor sua representação interna**;
  - Dar suporte a **mais de uma maneira de percorrer** a estrutura;
  - Prover **interface única** para percorrer estruturas agregadas diferentes.

Iterator – Consequências

- Mantém separadas a representação interna e a responsabilidade de navegação pelas partes.
  - O iterador conhece a estrutura interna das partes, mas os clientes do iterador não conhecem.
  - Move da coleção de objetos para o objeto iterator a responsabilidade de acesso e varredura da coleção.

- Há a possibilidade de utilizar mais de um iterador simultaneamente.
  - Dá suporte a múltiplas maneiras de percorrer a coleção e, se necessário, essas varreduras podem ocorrer ao mesmo tempo.