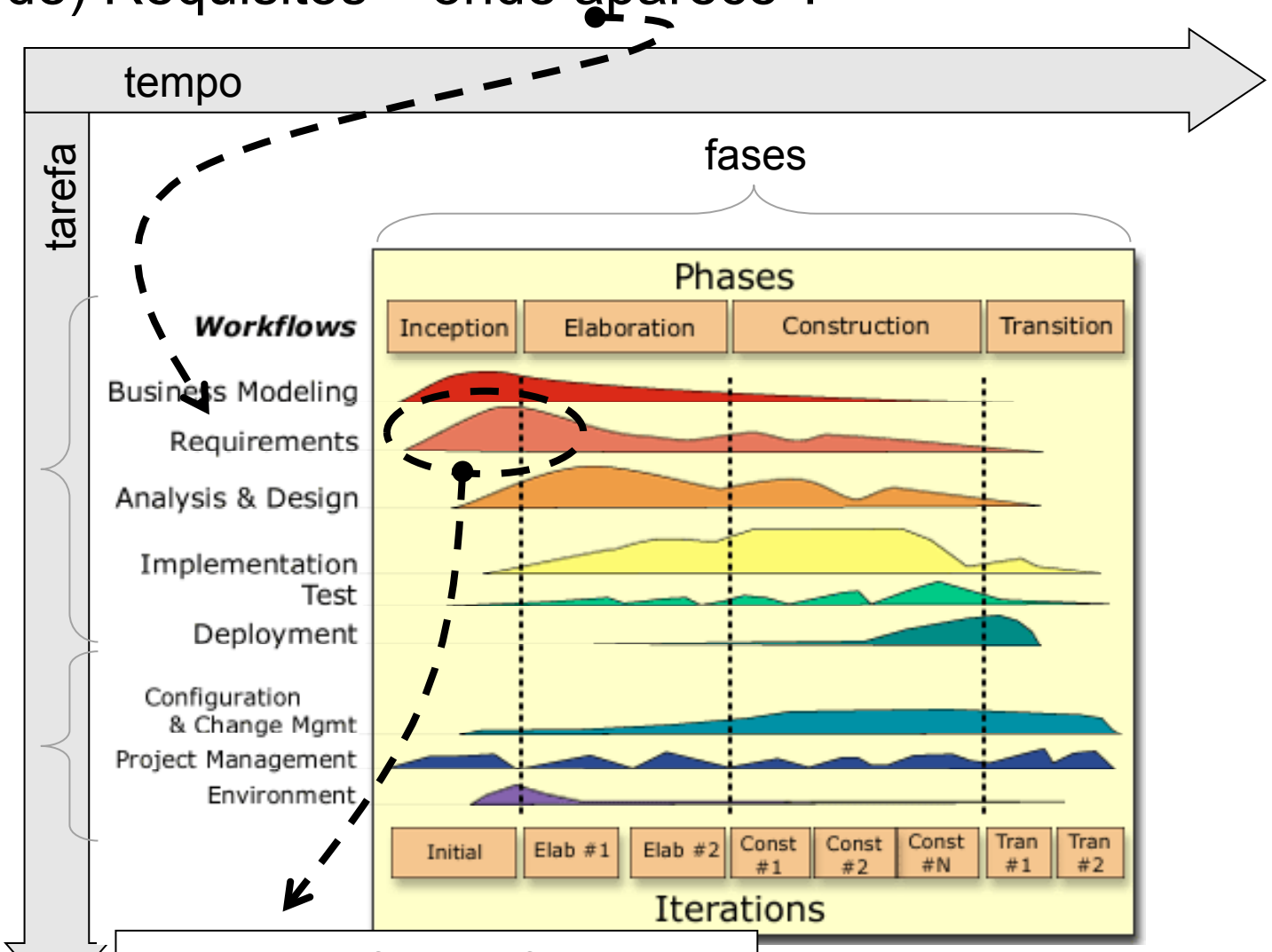
Requisitos, Análise e Desenho, Implementação – padrões

Paulo Trigo Silva

(Análise de) Requisitos – onde aparece ?

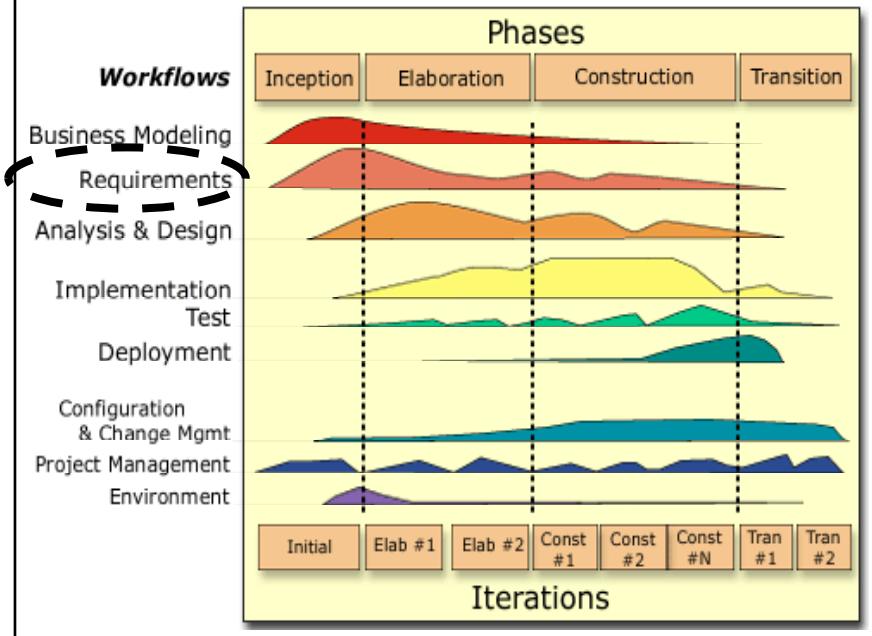


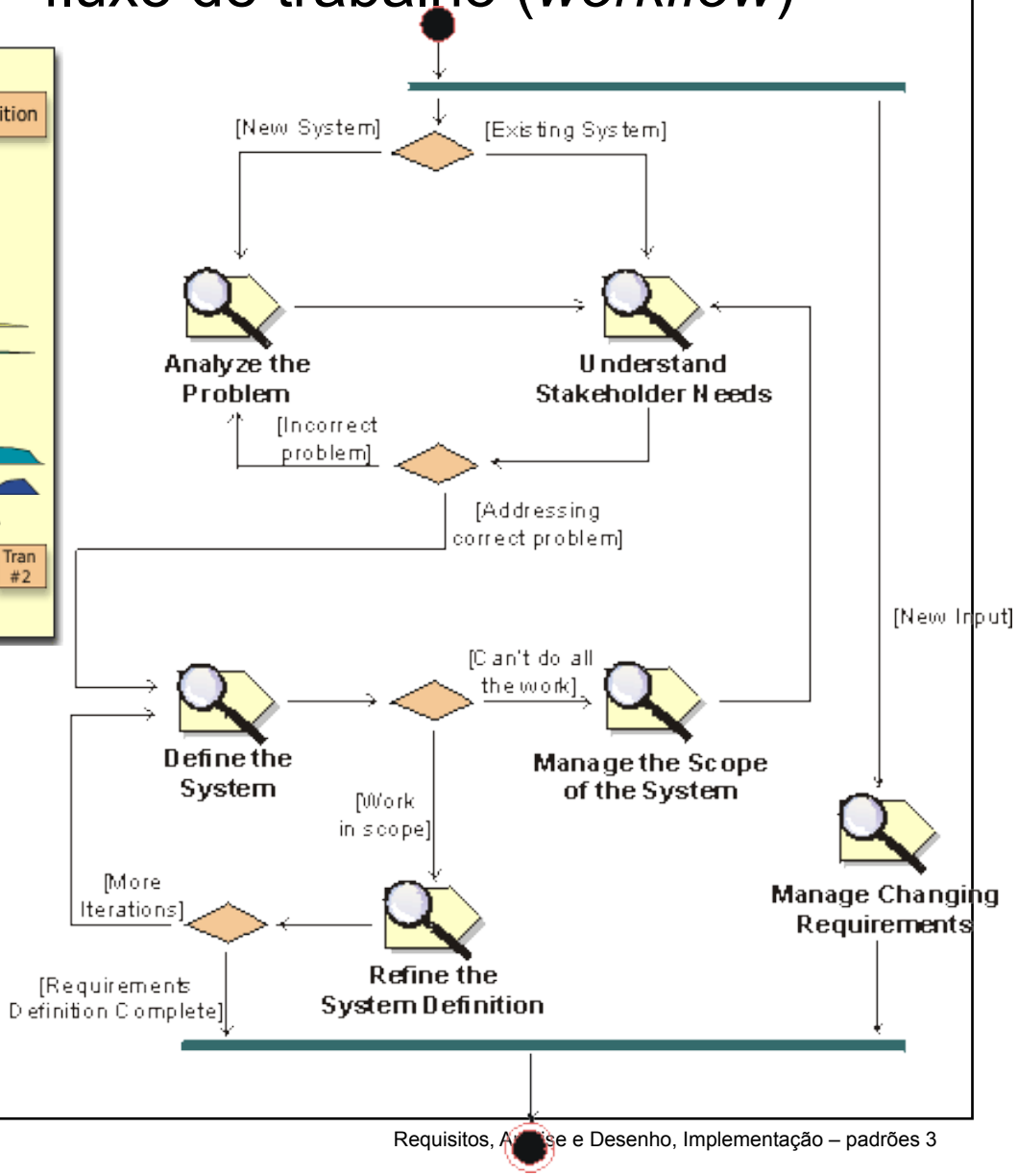
tarefas nucleares (*core workflows*)

tarefas de suporte (*supporting workflows*)

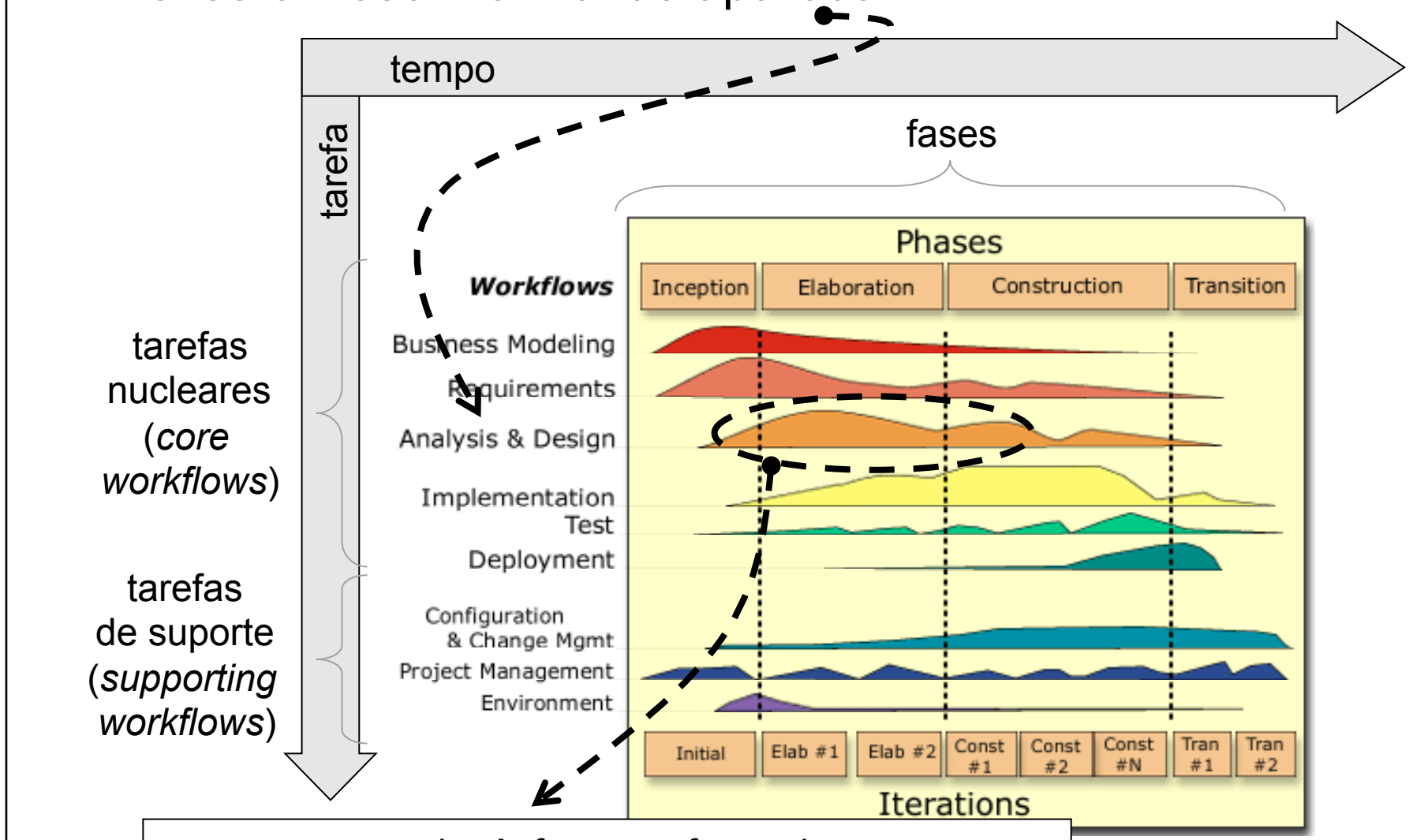
maior ênfase na fase de Avaliação Preliminar (*inception*)

(Análise de) Requisitos – fluxo de trabalho (*workflow*)



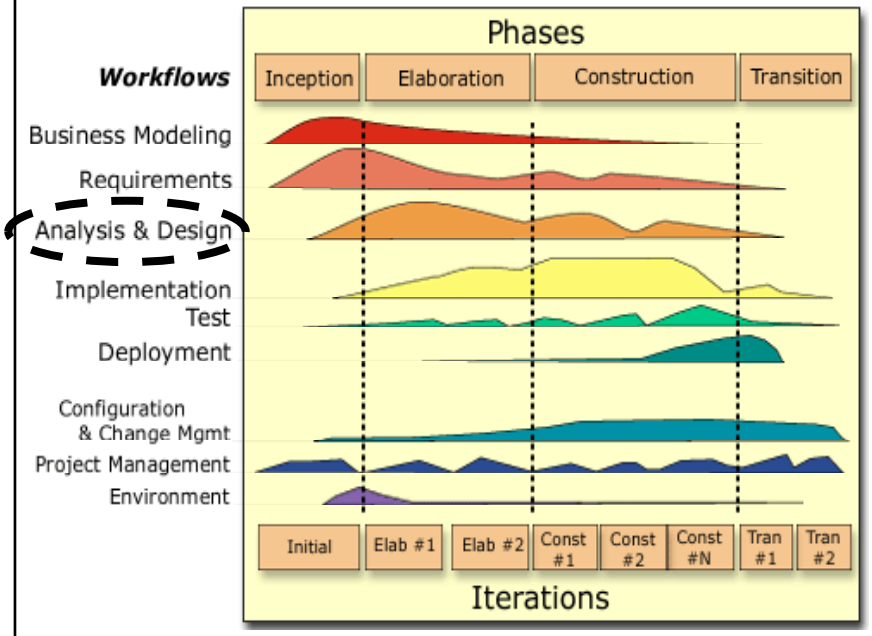


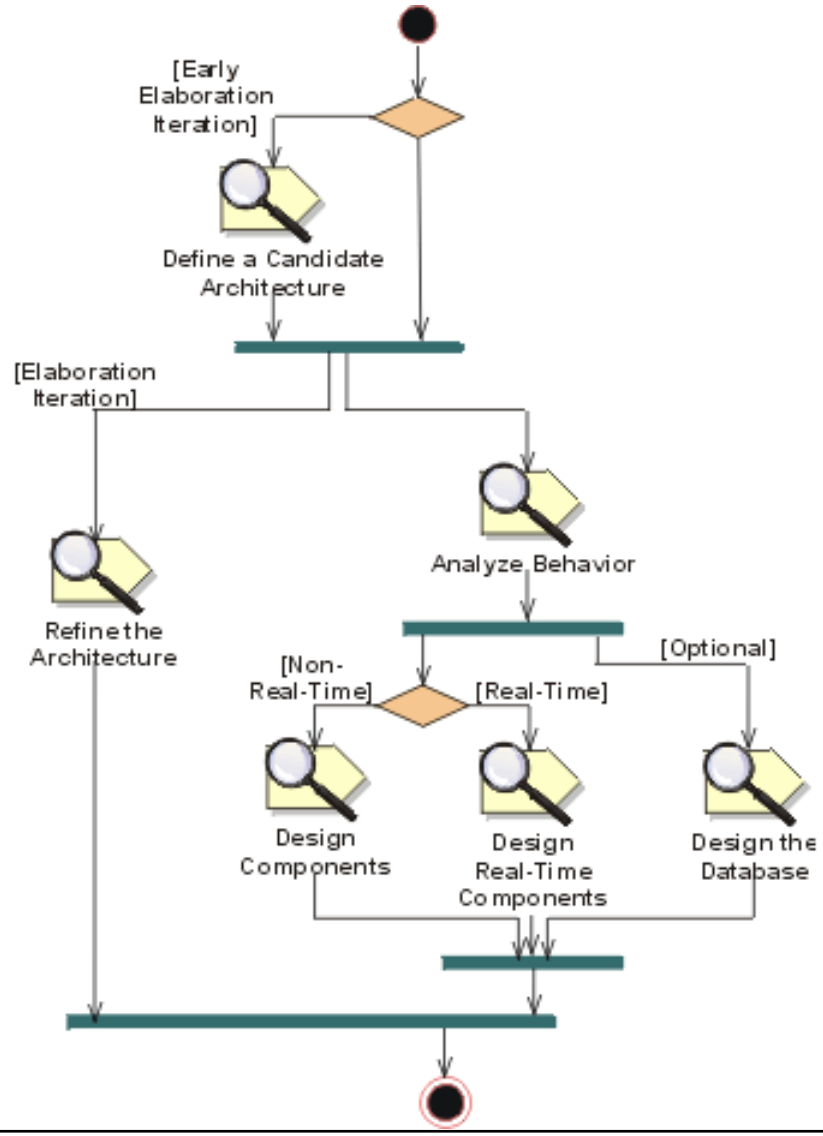
Análise e Desenho – onde aparece ?



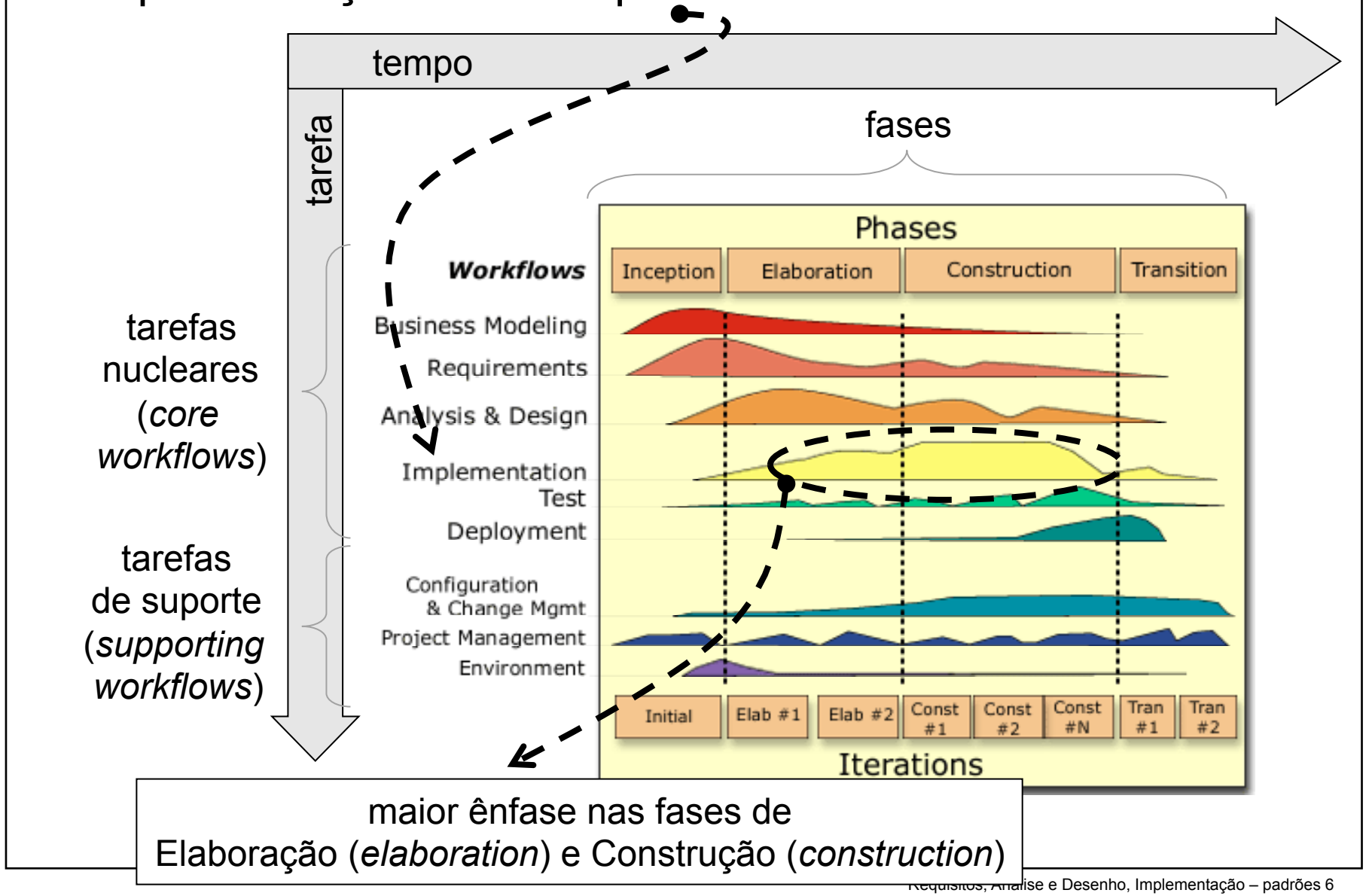
maior ênfase nas fases de Elaboração (*elaboration*) e Construção (*construction*)

Análise e Desenho – fluxo de trabalho (*workflow*)



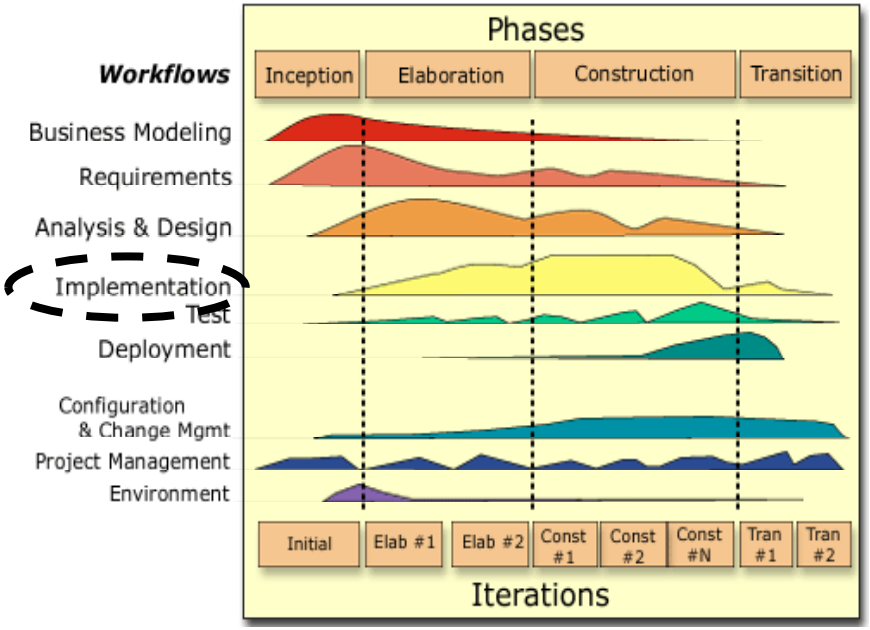


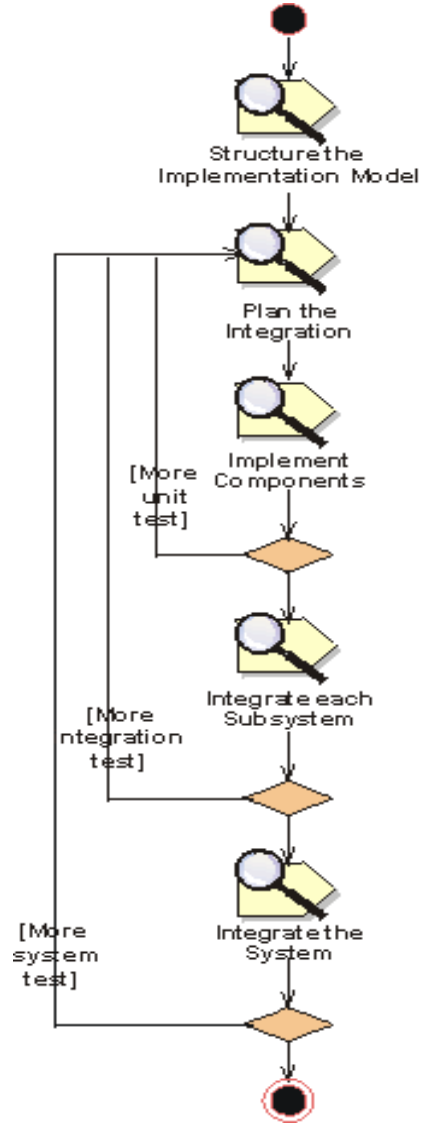
Implementação – onde aparece ?



maior ênfase nas fases de Elaboração (*elaboration*) e Construção (*construction*)

Implementação – Fluxo de trabalho (*workflow*)



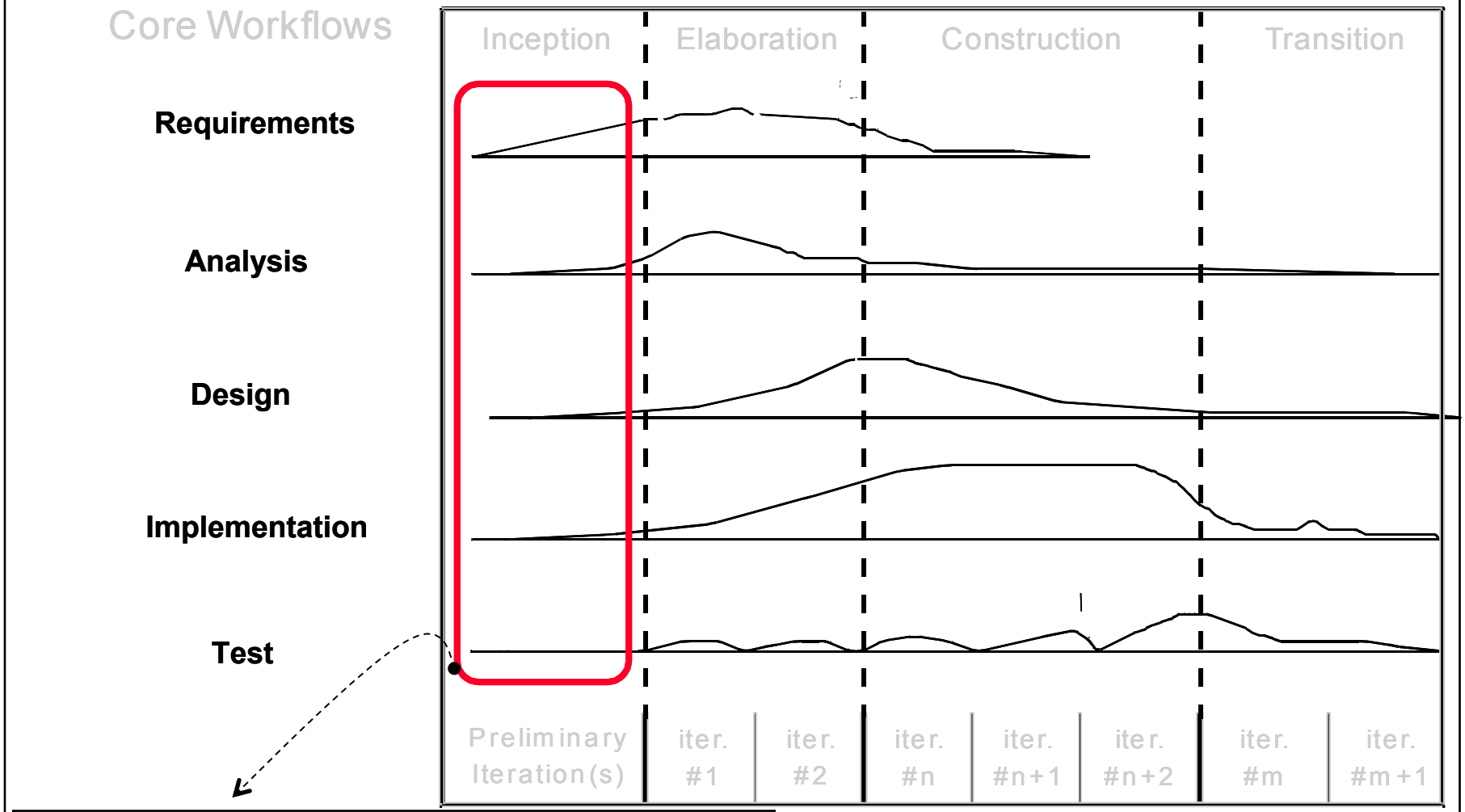


Um exemplo ... percorrer ciclo de desenvolvimento

- Considere-se o seguinte texto, que
  - exibe os requisitos de um sistema de suporte à persistência

> O grupo MO – Modelamos com Objectos, pretende construir uma bancada de trabalho (*framework*) que ofereça persistência aos seus modelos de objectos.
>
> A bancada dever ser usada por programadores e a persistência garantida através do sistema de ficheiros. A bancada não deve impor restrições à construção do modelo de classes (e.g. não deve impor que todas as classes derivem de alguma outra que represente a bancada). O programador apenas deve indicar, no momento da construção do objecto, que ele é persistente. O programador será também responsável por indicar como "desfazer" e "refazer" os objectos persistentes de cada classe. A bancada deve permitir que o programador indique que pretende consolidar (registar de modo permanente) as alterações a que os objectos tenham sido submetidos.
>
> Ao programador deve estar visível o conceito de "gestor do repositório". Este gestor deve oferecer o conceito de *iterator* sobre todas os objectos de determinada classe, permitindo a acção de remoção. Os objectos podem ser gravados num mesmo ficheiro com o nome da classe. A cada objecto o gestor tem que atribuir um identificador único.

... Requisitos, Análise e Desenho, Implementação



Um ciclo de desenvolvimento (iteração) na fase de avaliação preliminar (*inception*)

Paulo Trigo Silva

Artefactos – caracterização geral

- Síntese de objectivos (*overview statement*)

> Neste projecto será construída uma bancada de suporte à persistência de objectos. A bancada deverá ser utilizada por programadores.

- Clientes (*customers*)

> Grupo MO – Modelamos com Objectos.

- Metas a alcançar (*goals*)

> Pretende-se que este sistema contribua para:
> • reduzir a distância entre o modelo de objectos e o suporte à persistência
> • resolver o problema da persistência em problemas de pequena escala
> • não expor no modelo de objectos do "negócio", a faceta da persistência

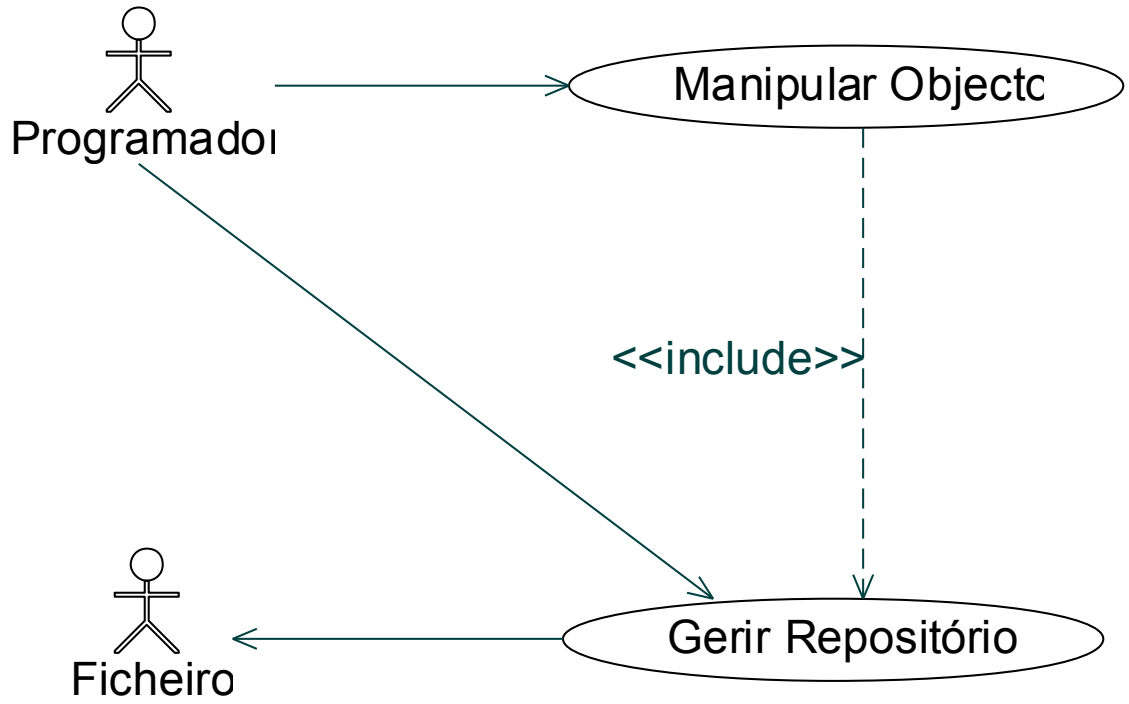
Funções do sistema

Ref. #	Função	Categoria
R1.1	Construir (instanciar) objecto	Evidente
R1.2	Invocar métodos da "interface" do modo "usual"	Evidente
R1.3	Tratar questão das múltiplas referências para o mesmo objecto	Invisível
R1.4	Atribuir identificador único a cada objecto	Invisível
R1.5	Consolidar alterações sempre que solicitado	Evidente
R1.6	Gravar objectos em ficheiro	Invisível
R1.7	Devolver iterador para uma classe	Evidente
R1.8	Formatar objectos a ser escritos / lidos	Invisível
R1.9	Remover objecto	Evidente

Atributos do sistema

Atributo	Detalhe / Restrição de Fronteira	Categoria
interacção Homem-Máquina	Detalhe ao nível da linguagem de programação	Obrigatório
facilidade de utilização	Restrição de Fronteria construir objecto, consolidar altrações e obter iterador	Obrigatório
	Detalhe outras formas avançadas de utilização	Desejável
plataformas	Detalhe Windows e Linux	Obrigatório
tolerância a falhas	Detalhe no mínimo a oferecida pelo sistema de ficheiros	Obrigatório
tempo de resposta	Detalhe o menor possível	Desejável
escalabilidade	Detalhe gestão de espaço na ordem das centenas de objectos	Obrigatório
	Detalhe preparado para vir a suportar probelams de maior dimensão	Desejável

Diagrama de Casos de Utilização



Paulo Trigo Silva

Casos de Utilização – formato Resumido

Manipular Objecto

Cabeçalho	
Nome:	***Manipular Objecto***
Resumo:	*Um objecto é tornado persistente de modo transparente para o programador*
Referências:	*R1.1, R1.2, R1.3, R1.5*

Gerir Repositório

Cabeçalho	
Nome:	***Gerir Repositório***
Resumo:	*Gestão (inserção, actualização, remoção e pesquisa) de objectos sobre o repositório que serve de suporte à persistência*
Referências:	*R1.4, R1.6, R1.7, R1.8, R1.9*

Paulo Trigo Silva

... formato Expandido – Manipular Objecto

Cenário Principal (fluxo típico de eventos)			
Acção do Actor		**Resposta do Sistema**	
1	O Caso de Utilização inicia quando o programador constrói um objecto que indica ser persistente		
		2	Objecto é tornado persistente (escrito em ficheiro) sendo-lhe atribuido um identificador único
		3	É devolvido ao programador um *proxy* (representante / invocador) do objecto
Cenário Alternativo 1			
Número de Sequência		**Alternativa**	
1	É indicado o objecto e também o seu identificador único	2	sem efeito
Cenário Alternativo 2			
Número de Sequência		**Alternativa**	
1	Não é indicado o objecto, mas apenas a sua classe e identificador único	2	sem efeito
		3	É devolvido ao programador um *proxy* (representante / invocador) do objecto. Embora o objecto não exista em memória, será carregado assim que um dos seus métodos for invocado

... formato Expandido – Gerir Repositório

Cenário Principal (fluxo típico de eventos)			
Acção do Actor		**Resposta do Sistema**	
1	O Caso de Utilização inicia quando o programador solicita um iterador para determinada classe		
		2	Objecto sobre o qual o programador poderá iterar, obtendo em cada iteração uma das instâncias gravadas da classe
Cenário Alternativo 1			
Número de Sequência		**Alternativa**	
1	O Caso de Utilização inicia quando o programador solicita que sejam consolidados (escritos) todos os objectos persistentes	2	Todos os objectos, indicados como persistentes, são actualizados (reescritos no ficheiro)
Cenário Alternativo 2			
Número de Sequência		**Alternativa**	
1	O Caso de Utilização inicia quando "Manipular Objecto" solicita um identificador para um novo objecto	2	É gerado e devolvido um identificador único para o objecto
Cenário Alternativo 3			
Número de Sequência		**Alternativa**	
1	O Caso de Utilização inicia quando "Manipular Objecto" solicita que um objecto seja inserido ou actualizado	2	Objecto é registado em ficheiro

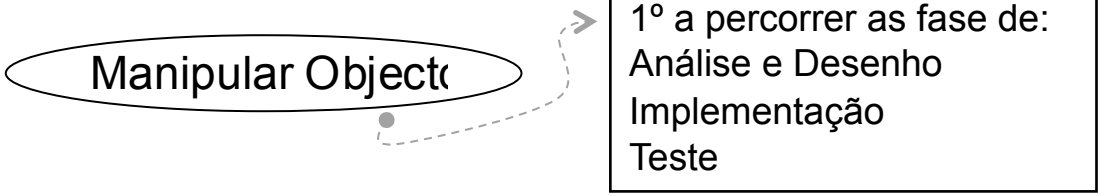
Escalonar Casos de Utilização

a	envolve esforço de investigação em tecnologias novas ou arriscadas
b	inclui funções de alta complexidade ou de tempo de resposta crítico
c	tem alto contributo na adição das classes que descrevem os conceitos do domínio, ou requer serviços especiais de suporte à persistência
d	exige grandes volume de informação ou detalhado conhecimento de negócio
e	representa processo essencial na linha-de-negócio
f	suporta directamente o retorno do investimento ou tem alta contribuição na redução de custos

Caso de Utilização	**a**	**b**	**c**	**d**	**e**	**f**	***Soma***
Peso	***3***	***1***	***2***	***1***	***1***	***1***	
Manipular Objecto	2	2	2	1	1	0	*14*
Gerir Repositório	1	4	2	1	0	0	*12*

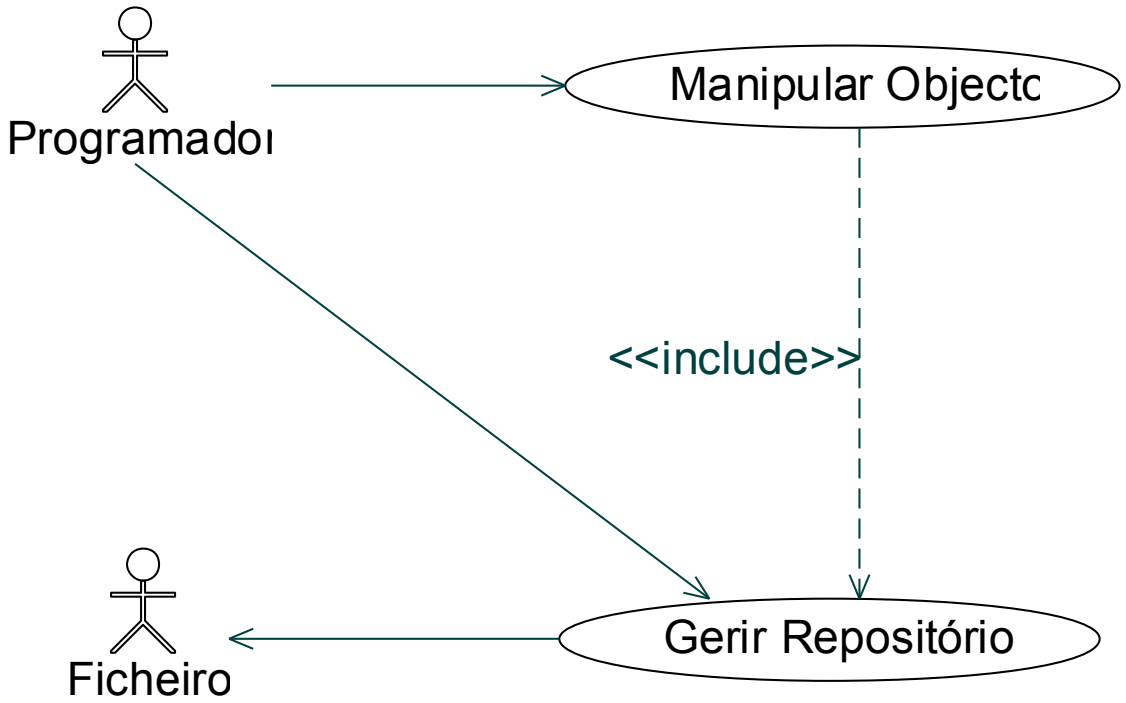
Paulo Trigo Silva

... versões dos Casos de Utilização



- Manipular Objecto
  - versão 1
    - construção do *proxy* solicitando que o objecto seja gravado
    - tratamento da questão das múltiplas referências
    - responder ao pedido de consolidar alterações
  - versão 2
    - suporte para o desfazer e refazer do objecto a gravar
- Gerir Repositório
  - versão 1
    - atribuir identificador único (registar último identificador atribuído)
    - inserir objecto (definir formato de gravação)
  - versão 2
    - actualizar objecto e remover (iterando sobre a classe)
    - reutilizar espaço livre (nas operações de inserção)

Caso de Utilização (versão 1 / característica 1)

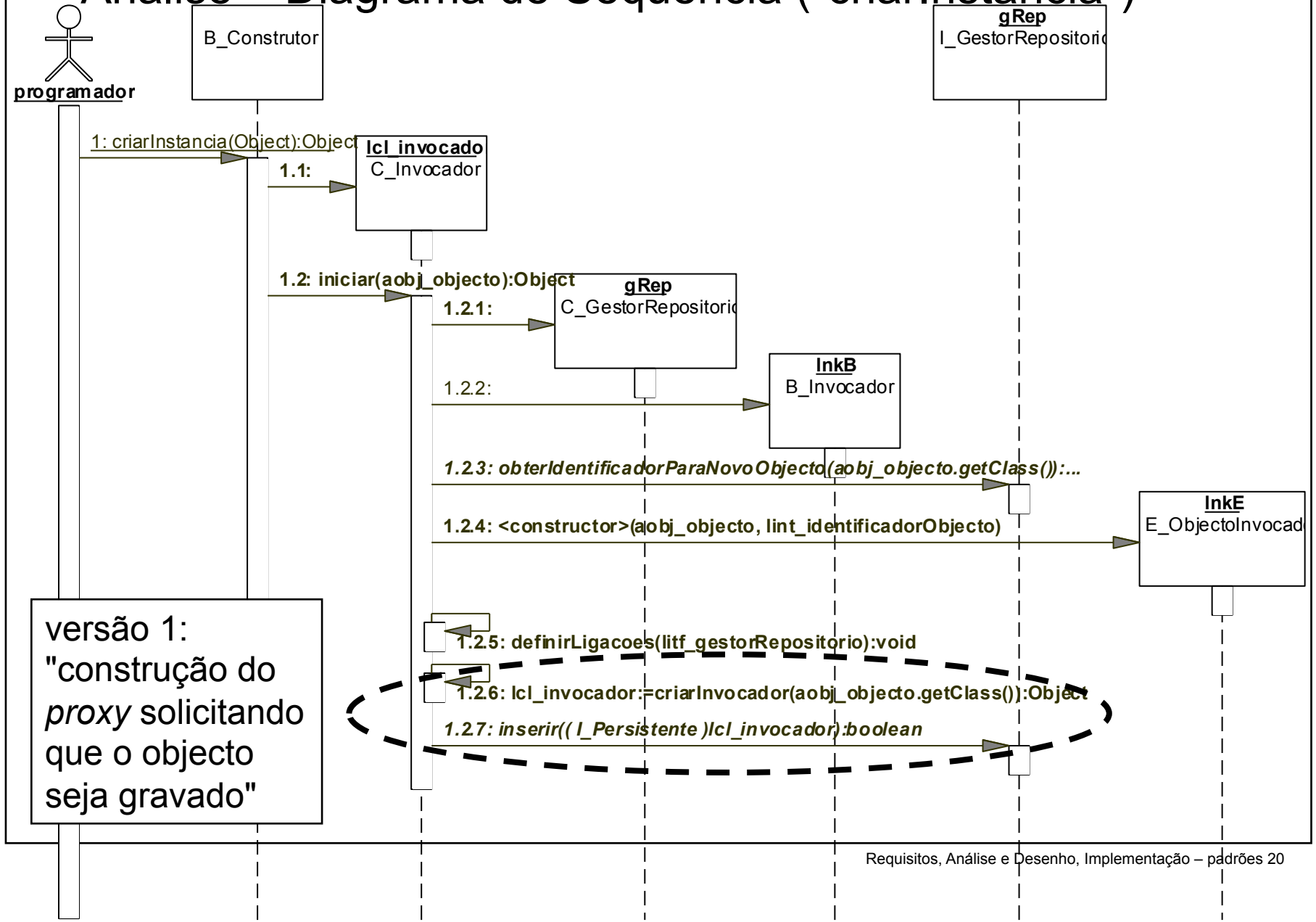


Manipular Objecto versão 1:

"construção do *proxy* solicitando que o objecto seja gravado"

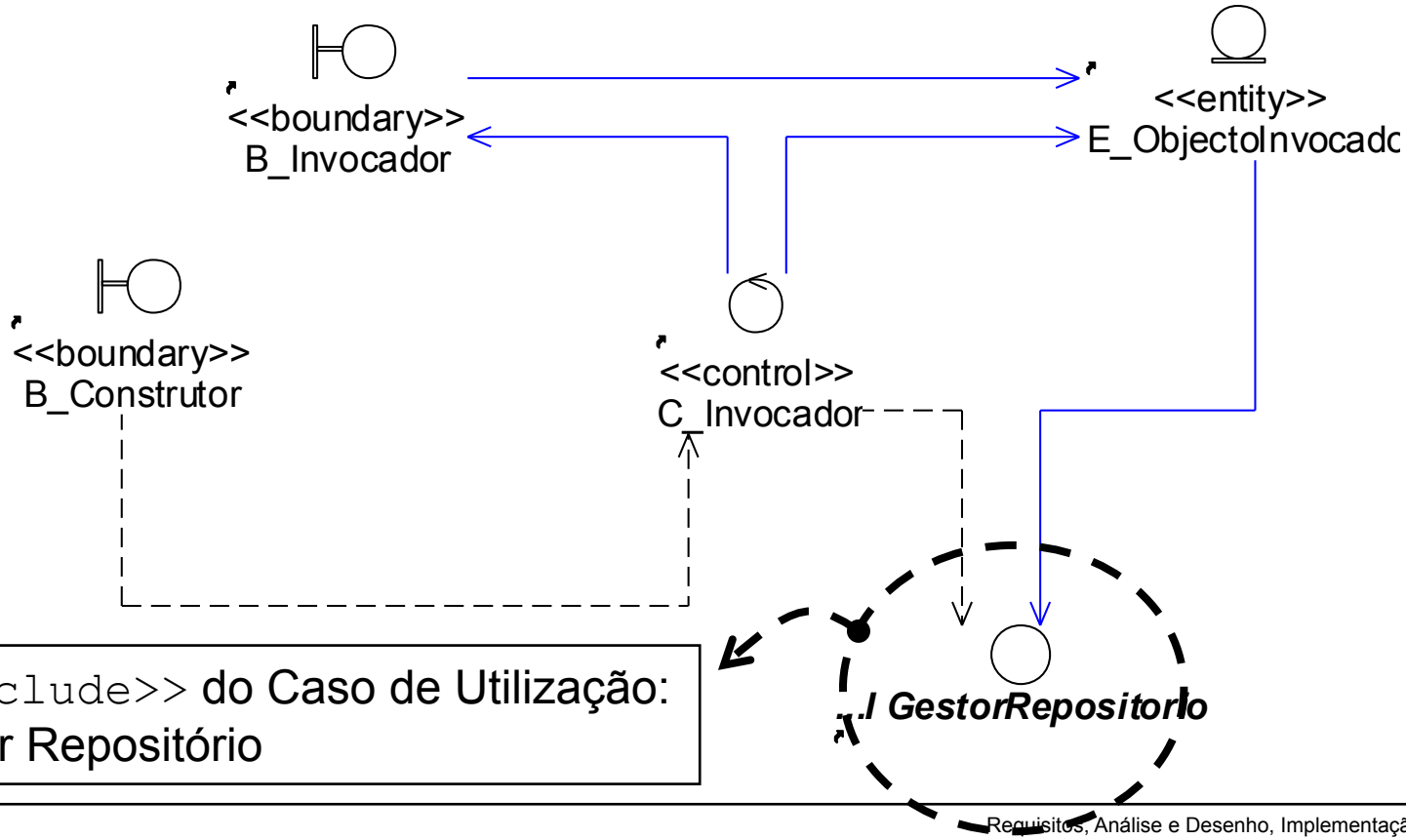
Paulo Trigo Silva

Análise – Diagrama de Sequência ("criarInstância")



Análise – Diagrama de Classes (Manipular Objecto)

Padrão de Análise – atribuição de responsabilidades:
- *Boundary*
- *Control*
- *Entity*



<<include>> do Caso de Utilização:
- Gerir Repositório

Padrões de Desenho

- em *Design Patterns*, por
  - *Erich Gamma*, *Richard Helm*, *Ralph Johnson*, *John Vlissides*
  - ... *the Gang of Four* (GoF)
- "*Each pattern describes a problem which occurs over and over again in our environment, and describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice*"
- Padrão de Desenho, apresenta os aspectos essenciais,
  - que tornam uma estrutura de Desenho
  - útil à criação de um Desenho O.O. reutilizável
- Cada Padrão de Desenho, foca,
  - um problema (ou aspecto) específico do Desenho O.O.

Padrões de Desenho – como se descrevem ?

- Alguns elementos da sua descrição
  - nome
    - "bom nome" é essencial pois fará parte do vocabulário do projecto
  - intenção – respostas simples às questões
    - "que aspectos específicos o padrão endereça ?"
    - "quais as razões para a sua existência ?"
  - participantes – classe e instâncias
    - os seus papeis e colaborações
    - a distribuição de responsabilidades
  - aplicabilidade
    - em que situações é usado
    - consequências e custo-benefício (*trade-off*) do seu uso

Padrões de Desenho – como se classificam ?

- Padrões classificam-se quanto a dois critérios
  - objectivo (*purpose*)
    - criação (*creational*)
    - estrutura (*structural*)
    - comportamento (*behavioral*)
  - âmbito (*scope*)

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter (class)	Interpreter
				Template Method
	Object	Abstract Factory	Adapter (object)	Chain of Responsability
		Builder	Bridge	Command
		Prototype	Composite	Iterator
		Singleton	Decorator	Mediator
			Facade	Memento
			Flyweight	Observer
			Proxy	State
				Strategy
				Vsitor

... alguns dos Padrões usados neste sistema

Padrões de Desenho – *Proxy*: Descrição

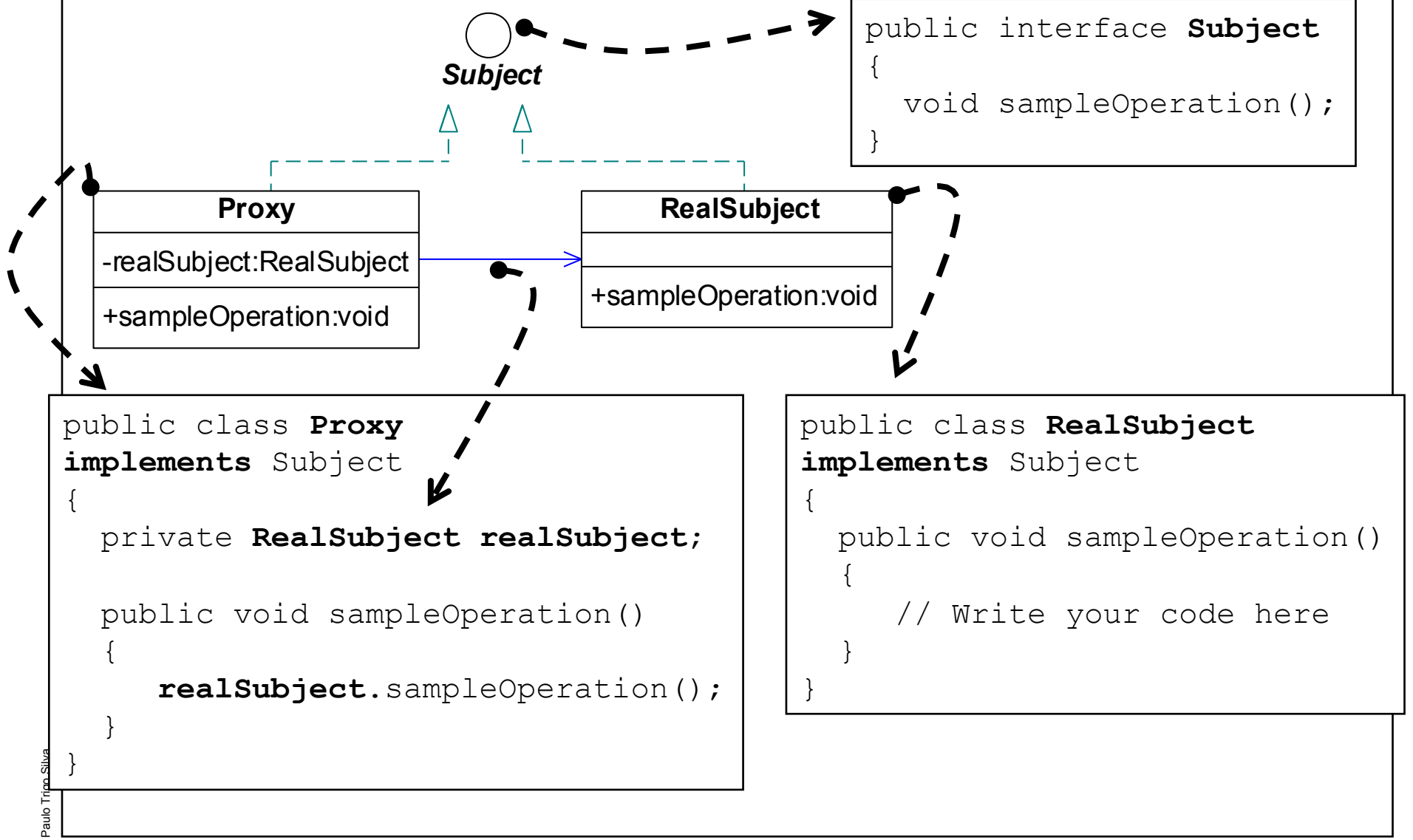
- Intenção
  - fornecer um substituto (ou representante) de outro objecto
  - através do substituto controlar o acesso ao objecto substituído
- Participantes
  - *Proxy* (não pode ser interface)
    - mantém uma referência que permite aceder ao *RealSubject*
    - controla acesso e pode ser ele a criar e destruir o *RealSubject*
  - *Subject*
    - define uma interface comum a *RealSubject* e *Proxy*
    - assim Proxy pode ser usado onde se espera *RealSubject*
  - *RealSubject*
    - define o objecto real que o *Proxy* representa

Padrões de Desenho – *Proxy*: Descrição (cont.)

- Aplicabilidade
  - *smart reference* – substitui o "simples apontador", e por exemplo,
    - cria instancia de um objecto persistente da 1ª vez que é referenciado
    - verifica se o objecto está bloqueado, antes de ser acedido
    - mantém contador para o número de referências ao objecto real
  - *remote proxy*
    - representa um objecto que existe noutro espaço de endereçamento
  - *virtual proxy*
    - cria "objecto caro" (e.g. imagem ou filme) apenas quando solicitado
  - *protection proxy*
    - efectua controlo de acesso ao objecto real
    - útil quando o objecto tem diferentes direitos de acesso

Paulo Trigo Silva

Padrões de Desenho – *Proxy*

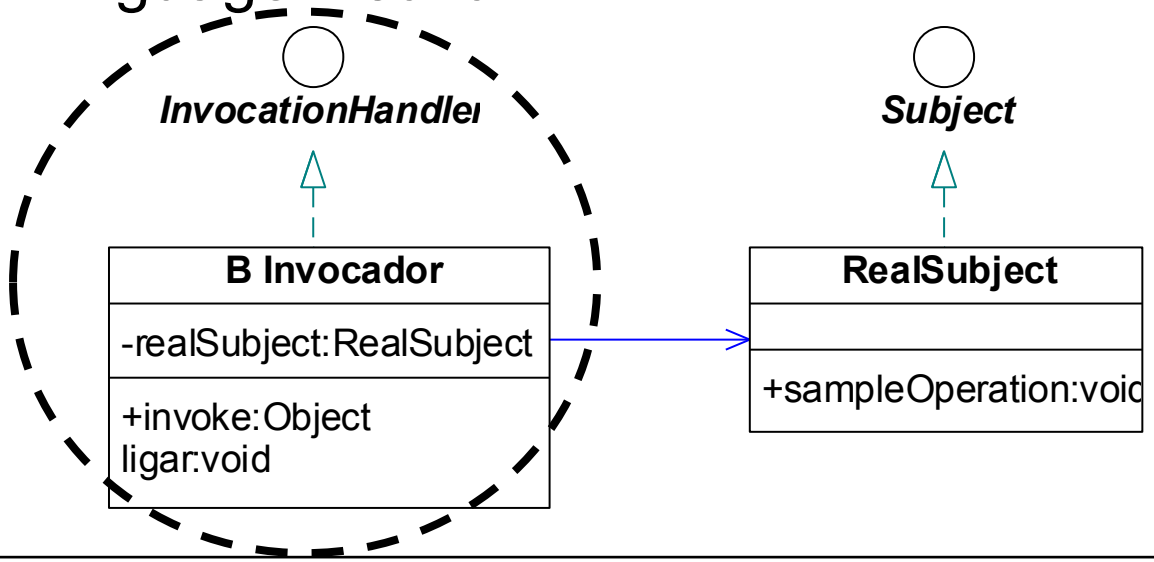



```
public interface Subject
{
   void sampleOperation();
}
```

```
public class Proxy
implements Subject
{
   private RealSubject realSubject;

   public void sampleOperation()
   {
      realSubject.sampleOperation();
   }
}
```

```
public class RealSubject
implements Subject
{
   public void sampleOperation()
   {
      // Write your code here
   }
}
```

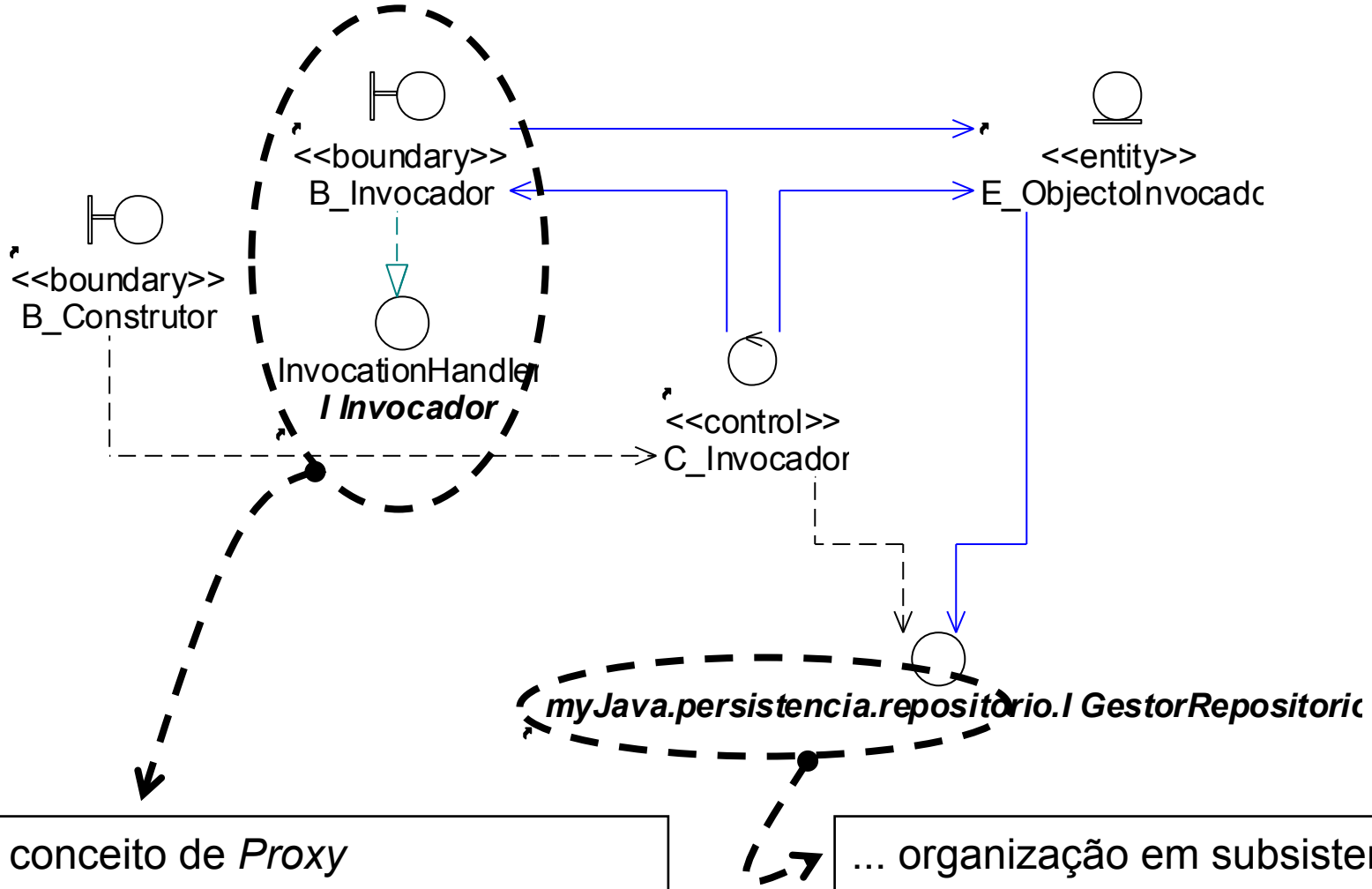
Proxy – linguagem *Java*




```
public class B_Invocador implements InvocationHandler
{
   private RealSubject realSubject;

   public Object invoke( Object acl_proxy, Method amth_metodo, Object[] aarr_argumentos )
   throws Throwable
   {
      Object lcl_resultado = null;
      try { lcl_resultado = amth_metodo.invoke( realSubject, aarr_argumentos ); }
      catch( InvocationTargetException lexp_excepcao ) { throw lexp_excepcao.getTargetException(); }
      catch( Exception lexp_excepcao ) { throw new RuntimeException( "Invocacao inesperada:" +
                                                                     lexp_excepcao.getMessage() ); }

      return lcl_resultado;
   }
```

```
void ligar( RealSubject acl_objecto )
{
   this.realSubject = acl_objecto;
}
```

Desenho – Diagrama de Classes (*Proxy*)

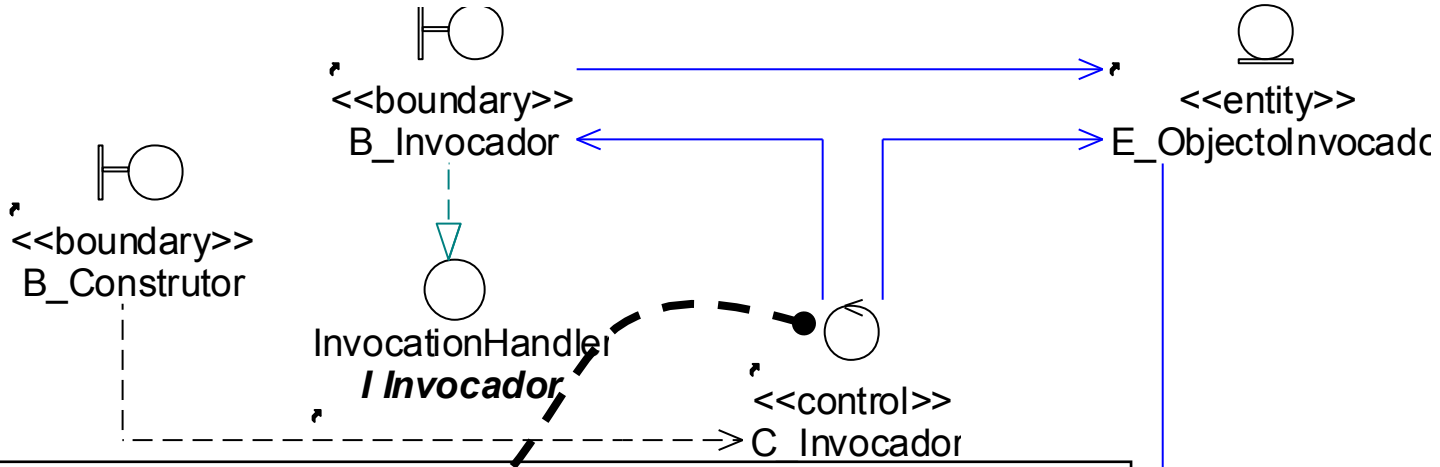


... conceito de *Proxy*
opção de Desenho: linguagem *Java*

... organização em subsistemas
opção de Desenho: *package Java*

Paulo Trigo Silva

Implementação (*Proxy*)

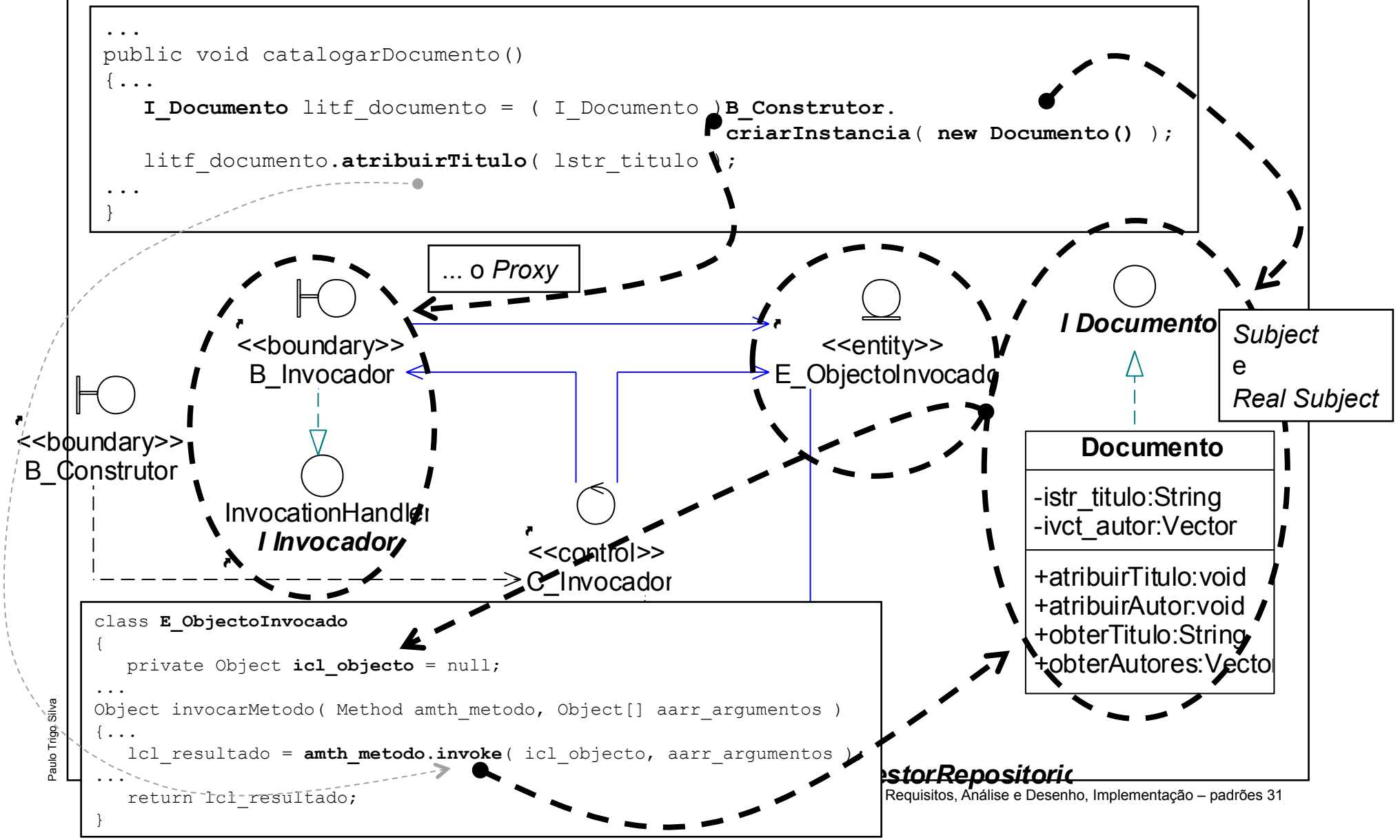



```
import java.lang.reflect.Proxy;
public class C_Invocador
{
   private B_Invocador lnkB_Invocador = null;
   ...
   public Object iniciar( Object aobj_objecto )
   {...
      lnkB_Invocador = new B_Invocador();
      Object lcl_invocador = criarInvocador( aobj_objecto.getClass() );
    ...
      return lcl_invocador;
   }

   private Object criarInvocador( Class acl_classe )
   { return Proxy.newProxyInstance( acl_classe.getClassLoader(),
                                    acl_classe.getInterfaces(),
                                    lnkB_Invocador ); }
```

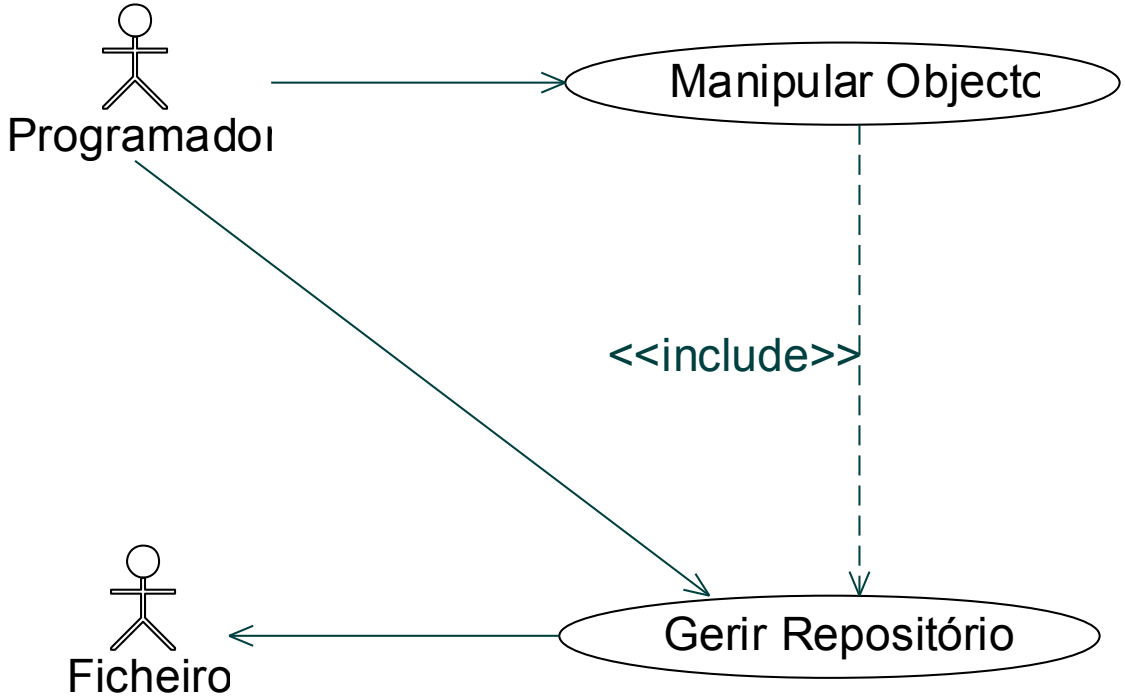
... é o *Proxy*

...rio.I GestorRepositorio

Teste – à implementação do *Proxy*



Caso de Utilização (versão 1 / característica 2)



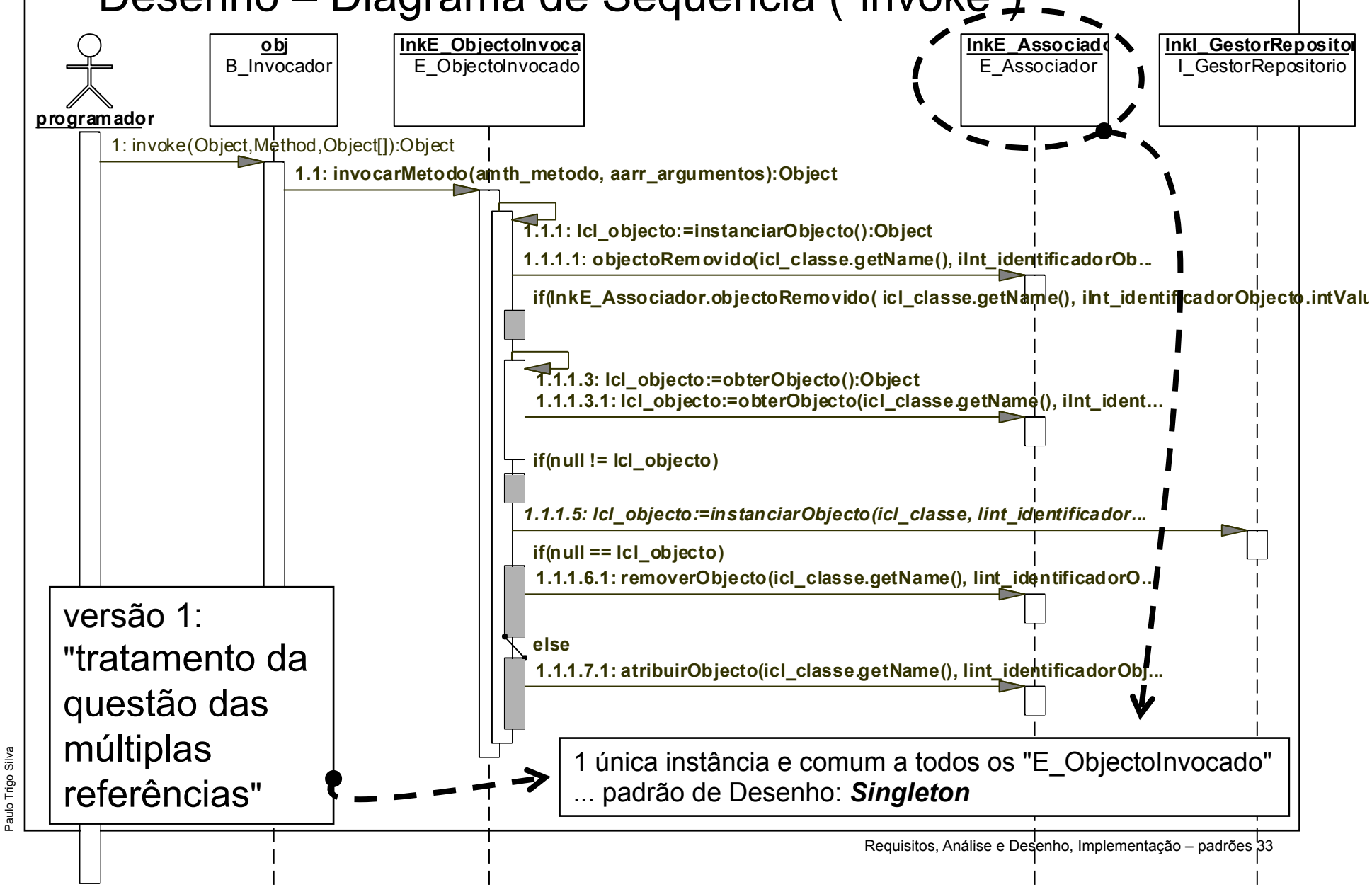
Manipular Objecto versão 1:

"construção do *proxy* solicitando que o objecto seja gravado"

"tratamento da questão das múltiplas referências"

Paulo Trigo Silva

Desenho – Diagrama de Sequência ("invoke")



Padrões de Desenho – *Singleton*: Descrição

- Intenção
  - garantir que uma classe apenas tem uma instância
  - fornecer um ponto global de acesso a essa instância
- Participantes (nenhum pode ser interface)
  - *Singleton*
    - a classe que apenas pode ter uma instância
  - *SingletonFactory*
    - a classe que tem um método estático para instanciar *Singleton*

Factory e *Singleton Factory* podem ser a mesma classe

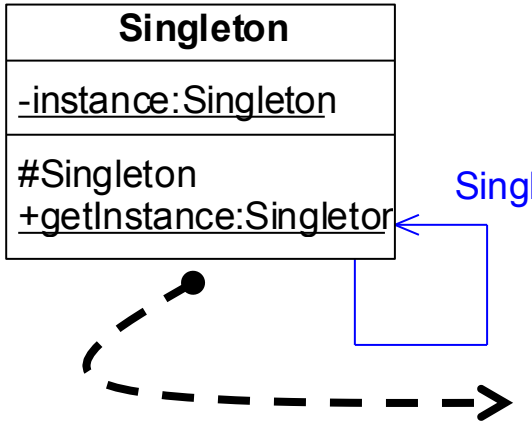
- Aplicabilidade
  - única instância acessível aos clientes de um ponto bem-conhecido
  - estender por herança sem que os clientes tenham que alterar código

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

alteração apenas no *Singleton Factory*

Paulo Trigo Silva

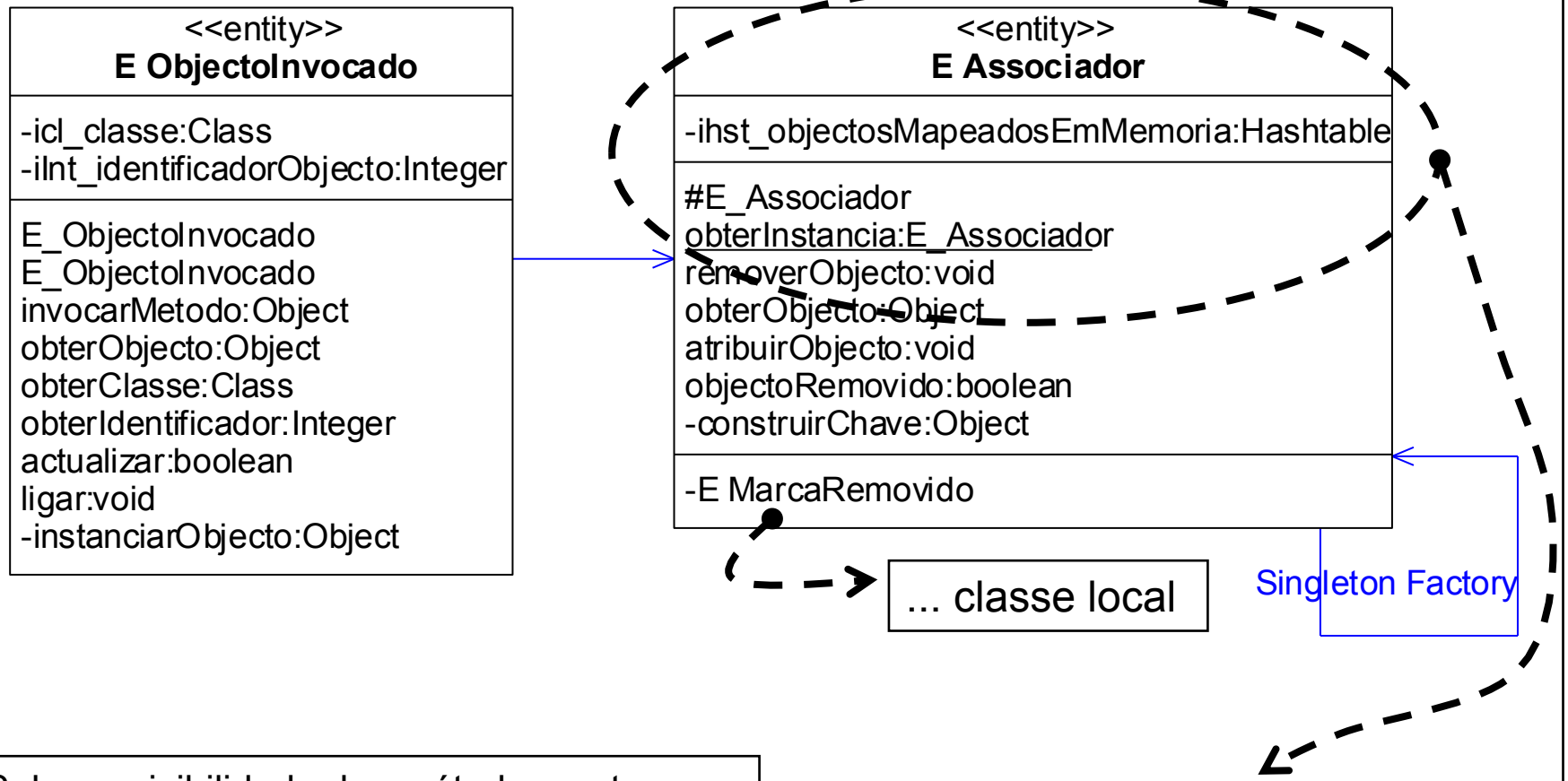
Padrões de Desenho – *Singleton*




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

    protected Singleton(){}

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

Paulo Trigo Silva

Desenho – Diagrama de Classes (*Singleton*)

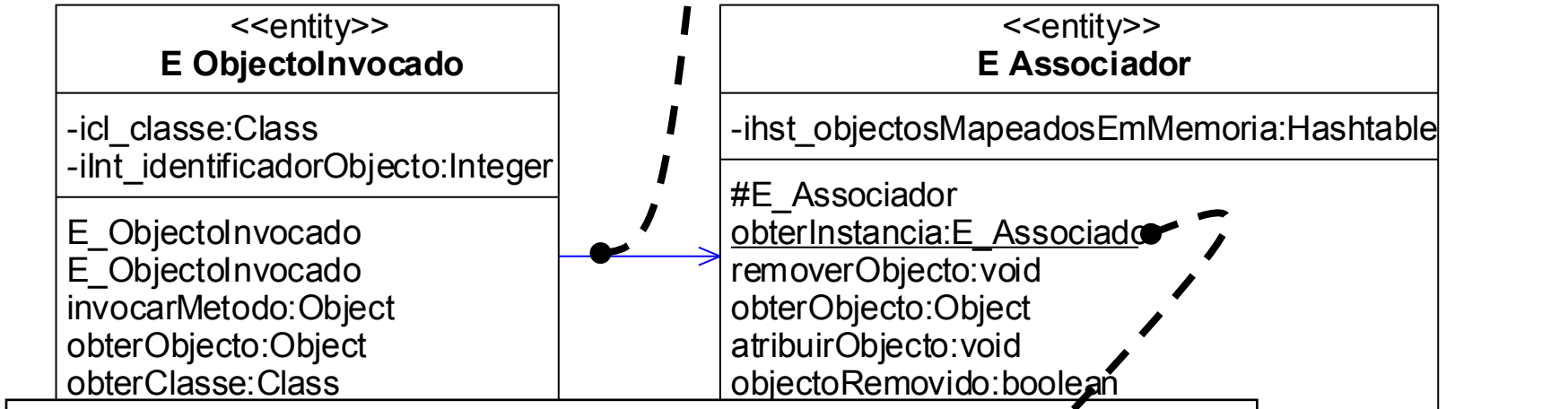


Sobre a visibilidade dos métodos, notar que:
- é *package local*
- não têm os símbolos (– ou +)

... conceito de *Singleton*
opção de Desenho: linguagem *Java*

Implementação (*Singleton*)

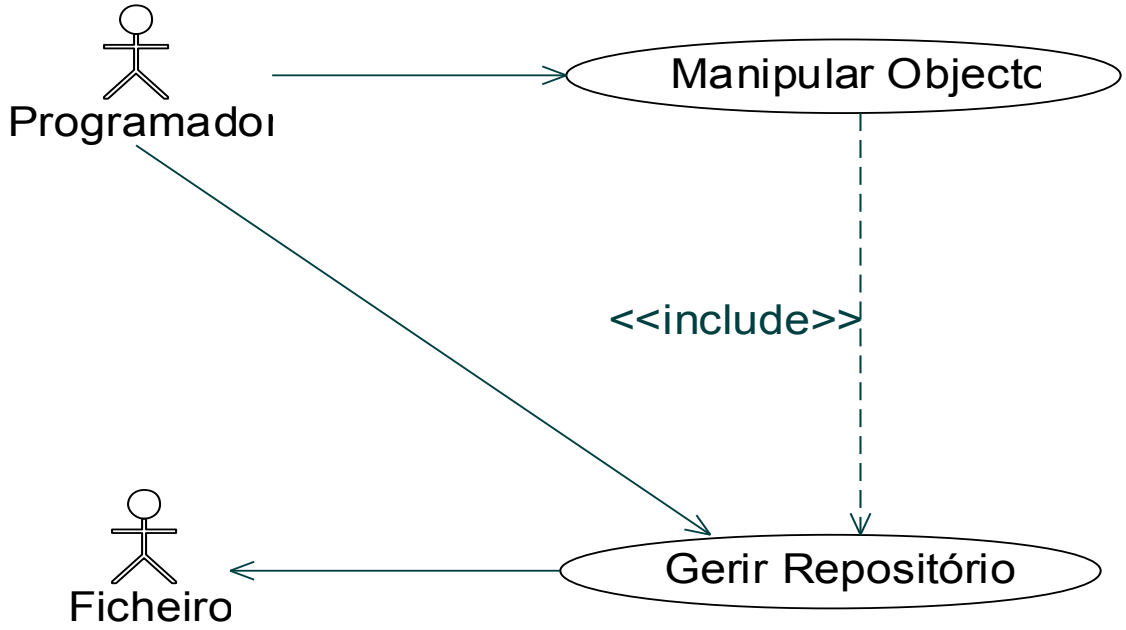
```
class E_ObjectoInvocado
{ ...
   private static E_Associador lnkE_Associador = E_Associador.obterInstancia();
  ...
```




```
class E_Associador
{
   private static E_Associador lnk_instancia = null;
   private Hashtable ihst_objectosMapeadosEmMemoria = new Hashtable();
...
   static E_Associador obterInstancia()
   {
      if( null == lnk_instancia )
      { lnk_instancia = new E_Associador (); }
      return lnk_instancia;
   }
...
```

Singleton Factory

Caso de Utilização (versão 1 / característica 3)



Manipular Objecto versão 1:

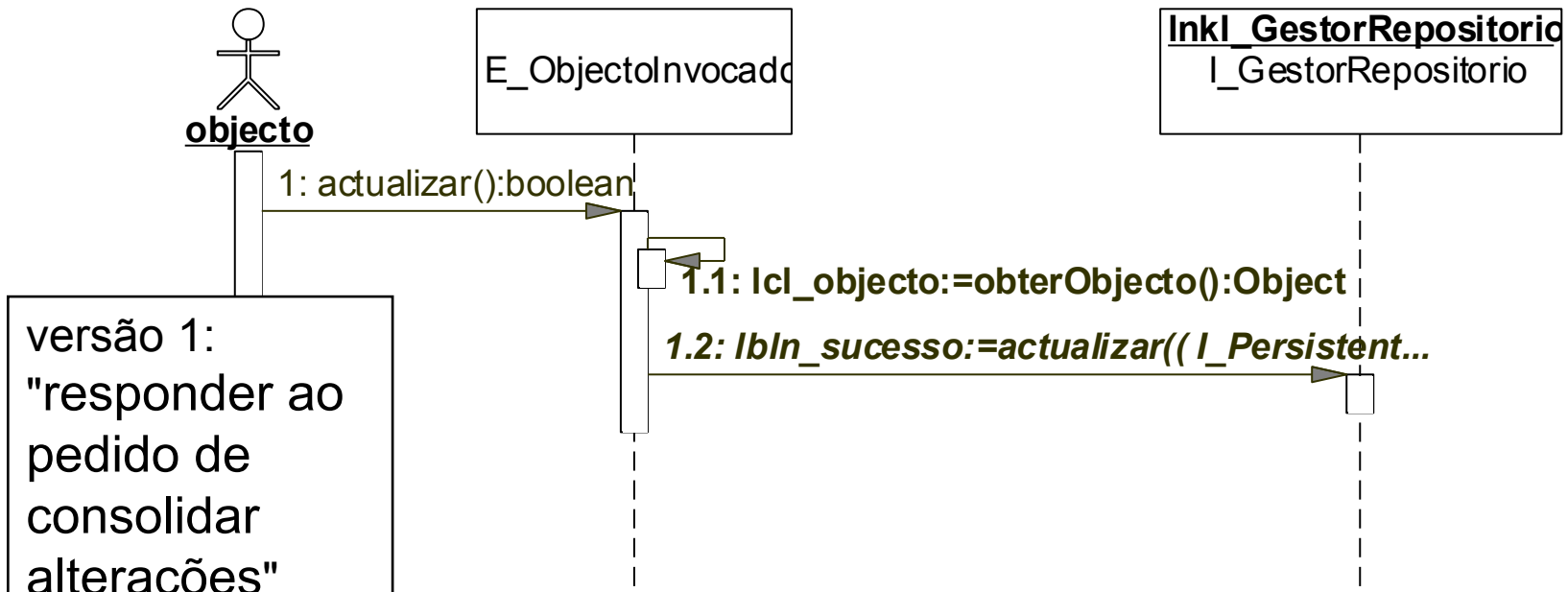
"construção do *proxy* solicitando que o objecto seja gravado"

"tratamento da questão das múltiplas referências"

"responder ao pedido de consolidar alterações"

Paulo Trigo Silva

Desenho – Diagrama de Sequência ("actualizar")



versão 1: "responder ao pedido de consolidar alterações"

Existem *N* instâncias de "E_ObjectoInvocado"

Questão:

Como garantir que todas essas instâncias executam o método "actualizar" depois de ter sido solicitada a consolidação das alterações ?

Padrões de Desenho – *Observer*: Descrição

- Intenção
  - definir dependência 1:N entre objectos
  - quando 1 objecto altera o seu estado, todos os dependentes,
    - são notificados e automaticamente actualizados
- Participantes
  - *Subject*
    - conhece os seus observadores
    - qualquer número de objectos *Observer* pode observar um *Subject*
    - define interface para ligar (*attach*) e desligar (*detach*) observadores
  - *Observer*
    - define interface para actualizar objectos que queiram ser notificados
    - a notificação ocorre na sequência de alterações ao *Subject*

Padrões de Desenho – *Observer*: Descrição (cont.)

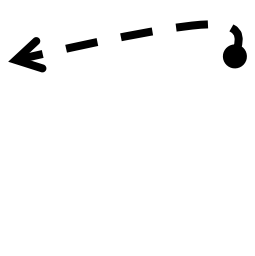
- Participantes
  - *Subject*
    - … já visto
  - *Observer*
    - … já visto
  - *ConcreteSubject* (não pode ser interface)
    - guarda estado de interesse para os objectos *ConcreteObserver*
    - envia notificação aos seus observadores quando altera o estado
  - *ConcreteObserver* (não pode ser interface)
    - mantém estado que deve ser consistente com o do seu *Subject*
    - implementa a interface de actualização do *Observer*
    - a implementação mantém consistência com o estado do seu *Subject*

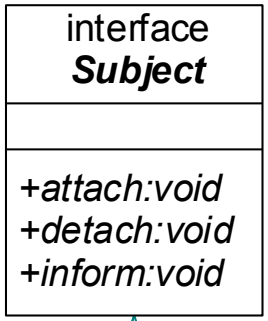
Padrões de Desenho – *Observer*: Descrição (cont. 1)

- Aplicabilidade
  - quando a alteração a um objecto requer alteração de outros e
    - não se sabe "quantos outros" precisam ser alterados
  - quando um objecto deve notificar outros sem nada assumir sobre eles
    - ou seja, pretende-se reduzir o acoplamento
  - quando uma abstracção tem dois aspectos, um dependente do outro
    - encapsular em objectos separados permite reutilizar esses aspectos
- Consequências da utilização deste Padrão
  - reutilizar *Subjects* sem reutilizar os seus *Observers* e vice-versa
  - adicionar *Observers* sem modificar o *Subject* ou os outros *Observers*
  - minimizar acoplamento
    - *Subject* apenas sabe que tem uma lista de *Observers*
    - *Subject* não conhece a classe concreta de nenhum dos *Observers*

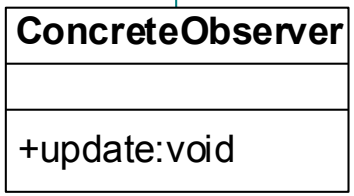
Padrões de Desenho – *Observer*

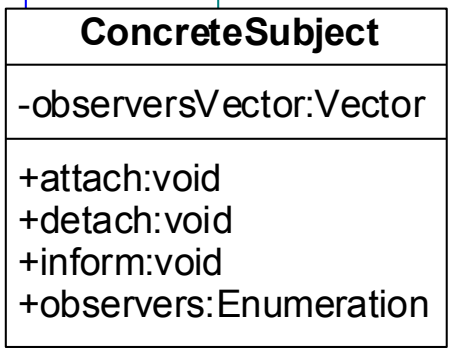
```
public interface Observer
{
    void update();
}
```





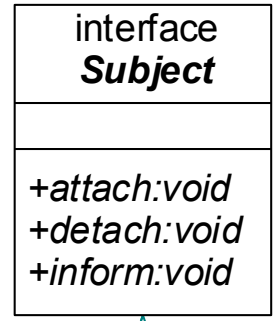

```
public class ConcreteObserver
implements Observer
{
    public void update()
    {
        // Write your code here
    }
}
```



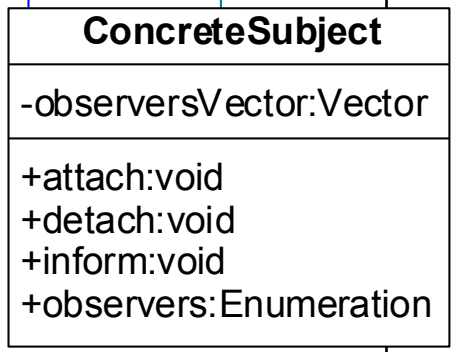


Padrões de Desenho – *Observer* (cont.)

```
public interface Subject
{
   void attach( Observer observer );
   void detach( Observer observer );
   void inform();
}
```

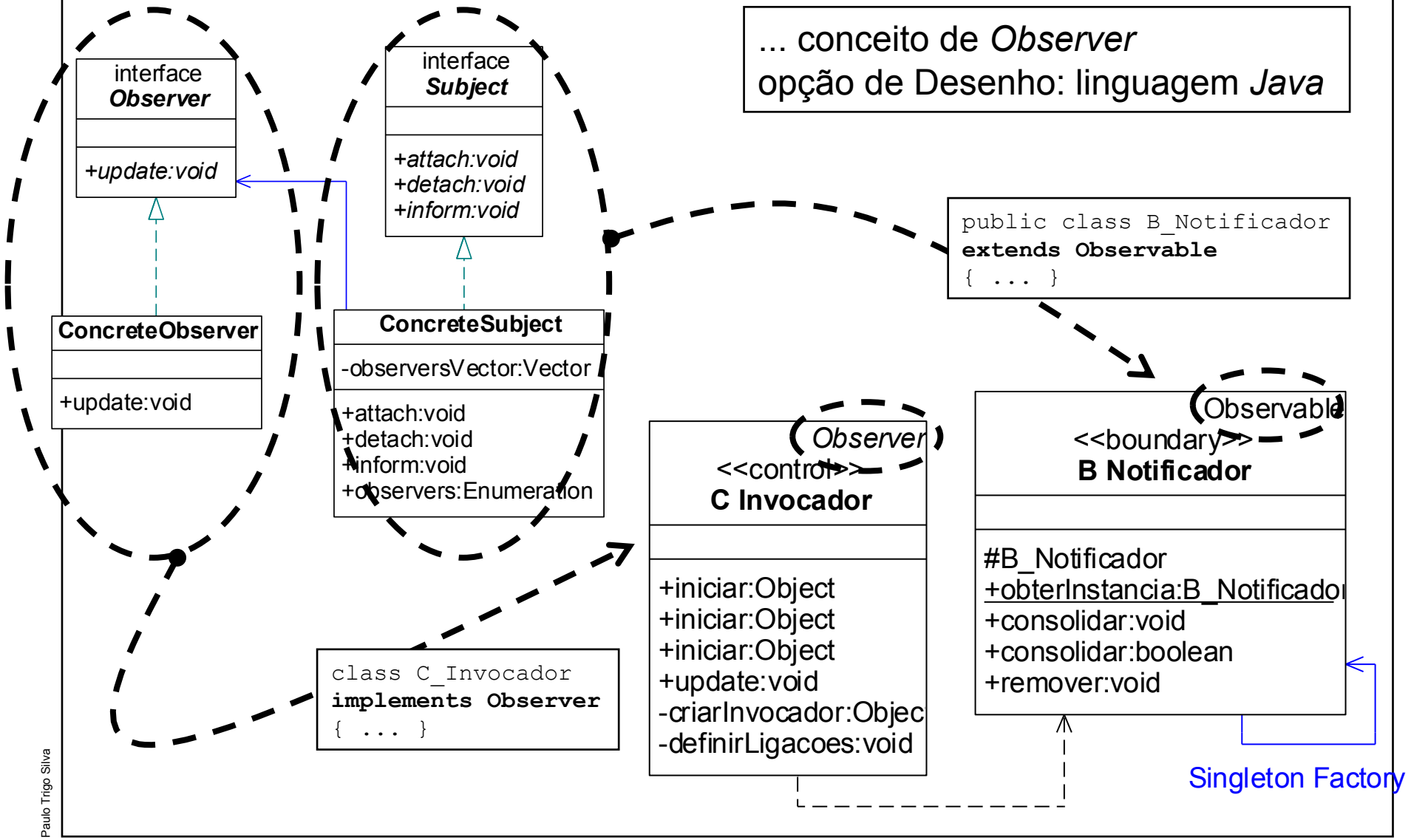



```
public class ConcreteSubject
implements Subject
{
   private Vector observersVector = new java.util.Vector();

   public void attach( Observer observer )
   { observersVector.addElement( observer ); }

   public void detach( Observer observer )
   { observersVector.removeElement( observer ); }

   public void inform()
   { java.util.Enumeration enumeration = observers();
     while( enumeration.hasMoreElements() )
     { ((Observer)enumeration.nextElement()).update(); } }

   public Enumeration observers()
   { return (( java.util.Vector )observersVector.clone() ).elements(); }
}
```

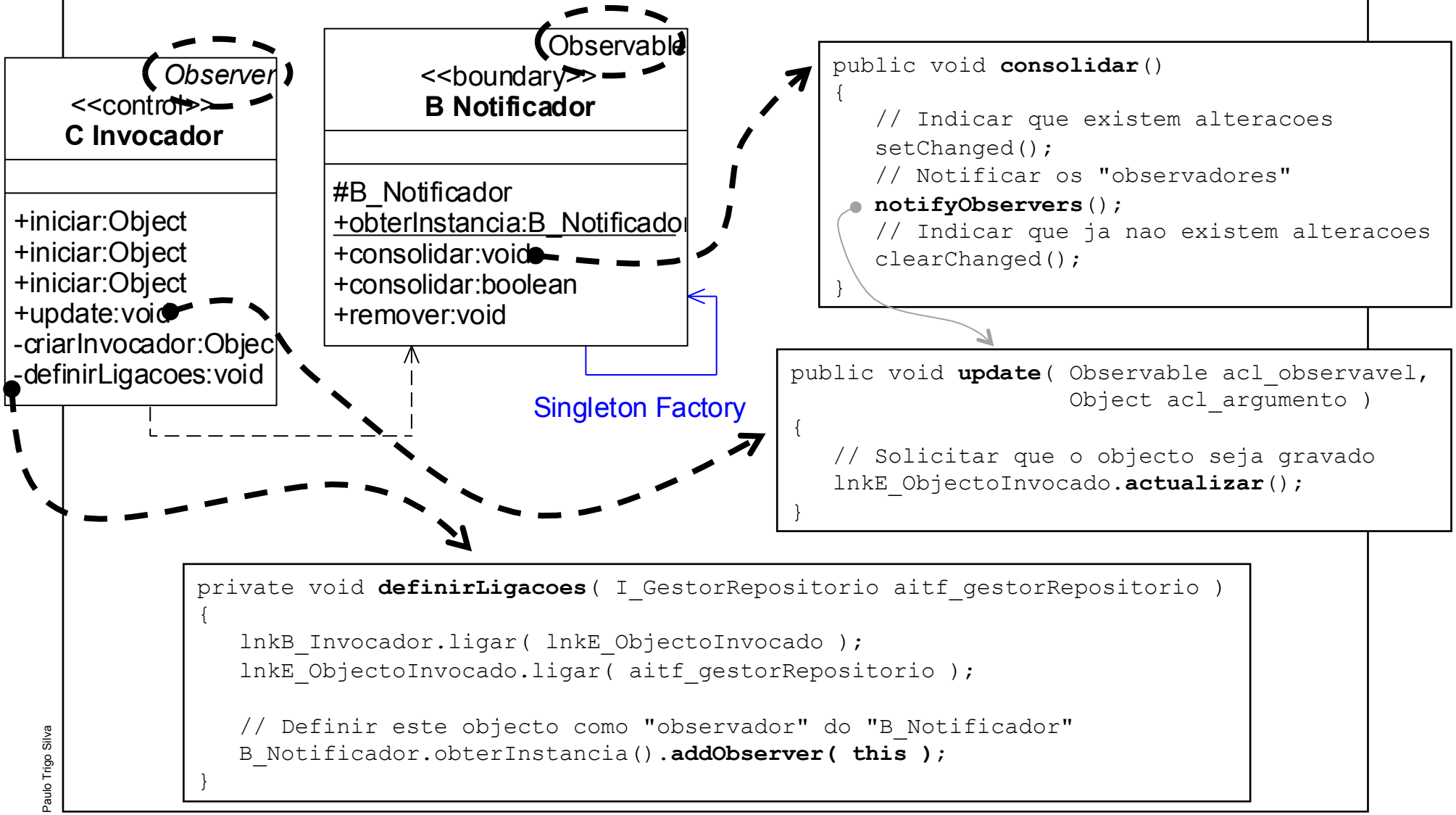


Desenho – Diagrama de Classes (*Observer*)

... conceito de *Observer*
opção de Desenho: linguagem *Java*



Implementação (*Observer*)



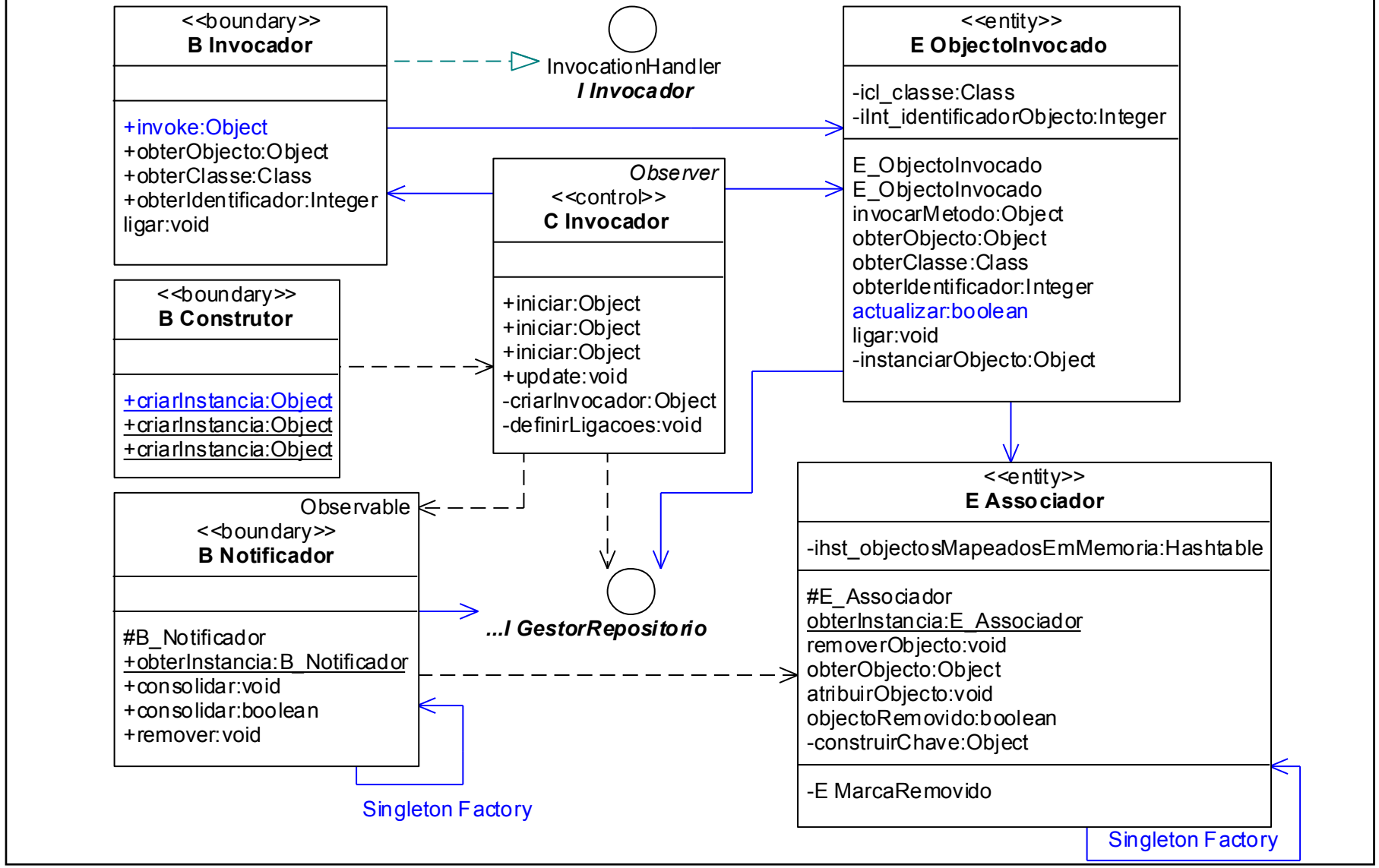

```
public void consolidar()
{
   // Indicar que existem alteracoes
   setChanged();
   // Notificar os "observadores"
   notifyObservers();
   // Indicar que ja nao existem alteracoes
   clearChanged();
}
```

```
public void update( Observable acl_observavel,
                    Object acl_argumento )
{
   // Solicitar que o objecto seja gravado
   lnkE_ObjectoInvocado.actualizar();
}
```

```
private void definirLigacoes( I_GestorRepositorio aitf_gestorRepositorio )
{
   lnkB_Invocador.ligar( lnkE_ObjectoInvocado );
   lnkE_ObjectoInvocado.ligar( aitf_gestorRepositorio );

   // Definir este objecto como "observador" do "B_Notificador"
   B_Notificador.obterInstancia().addObserver( this );
}
```

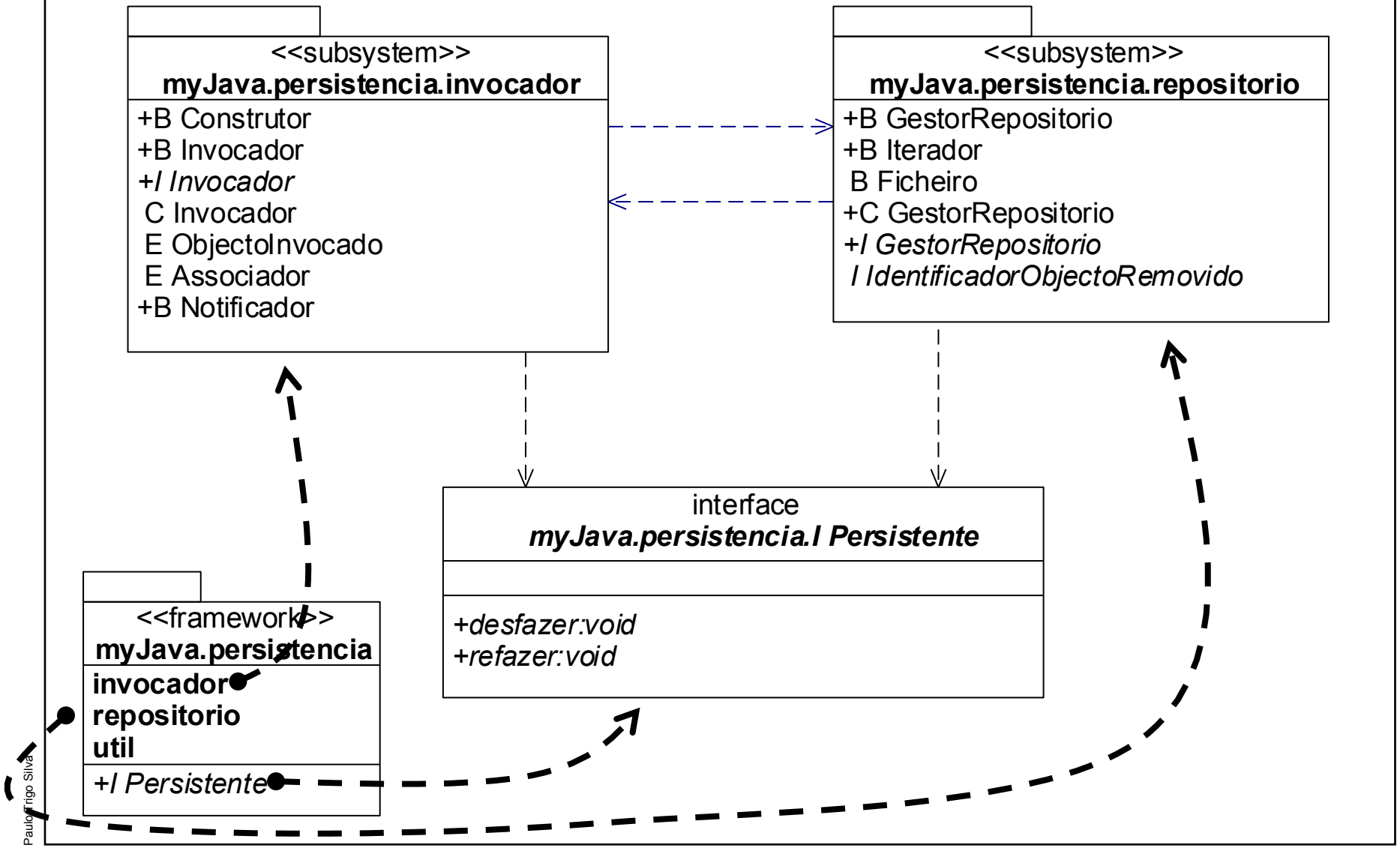
Paulo Trigo Silva

Desenho – Diagrama de Classes ("Manipular Objecto")

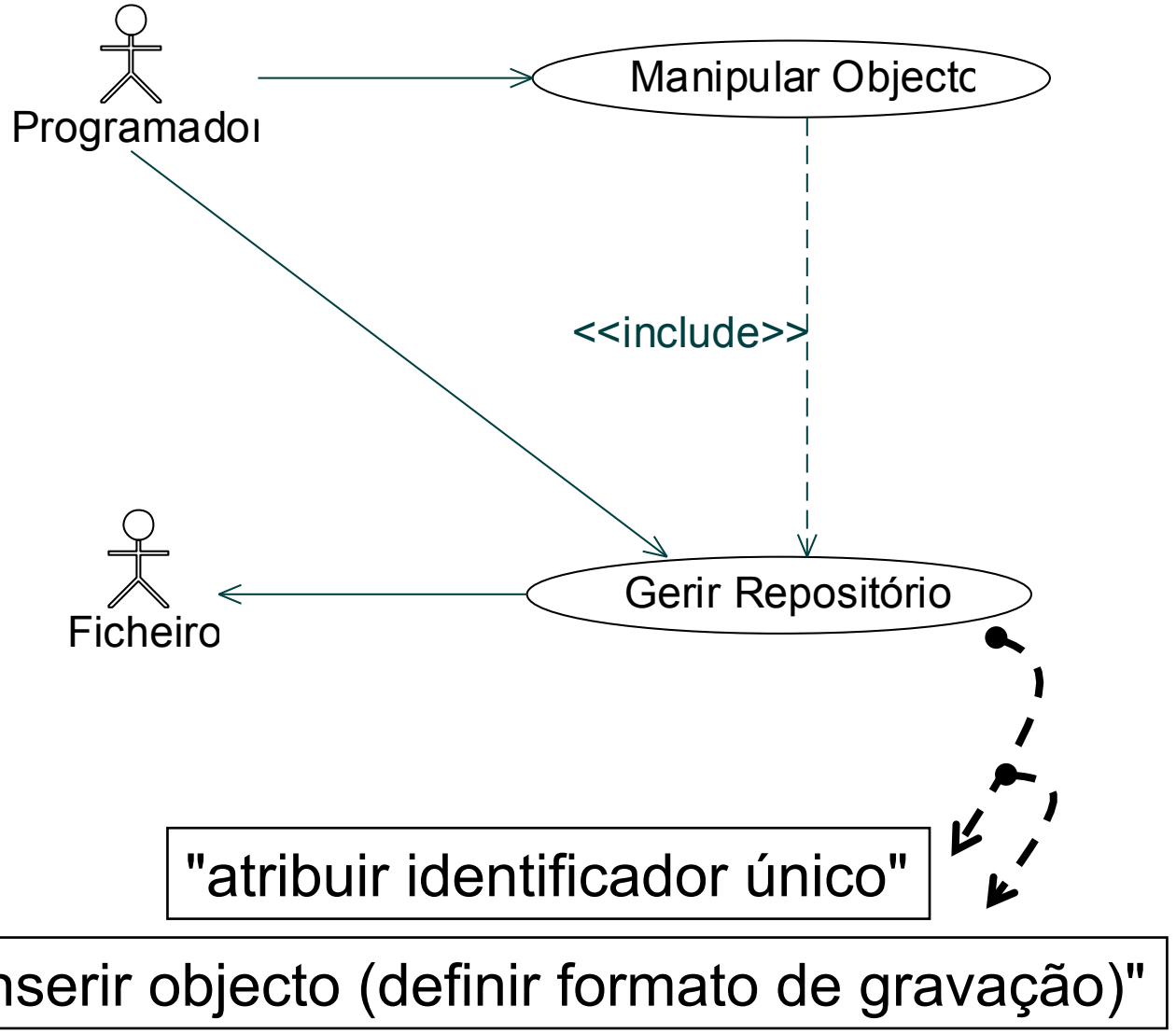


Paulo Trigo Silva

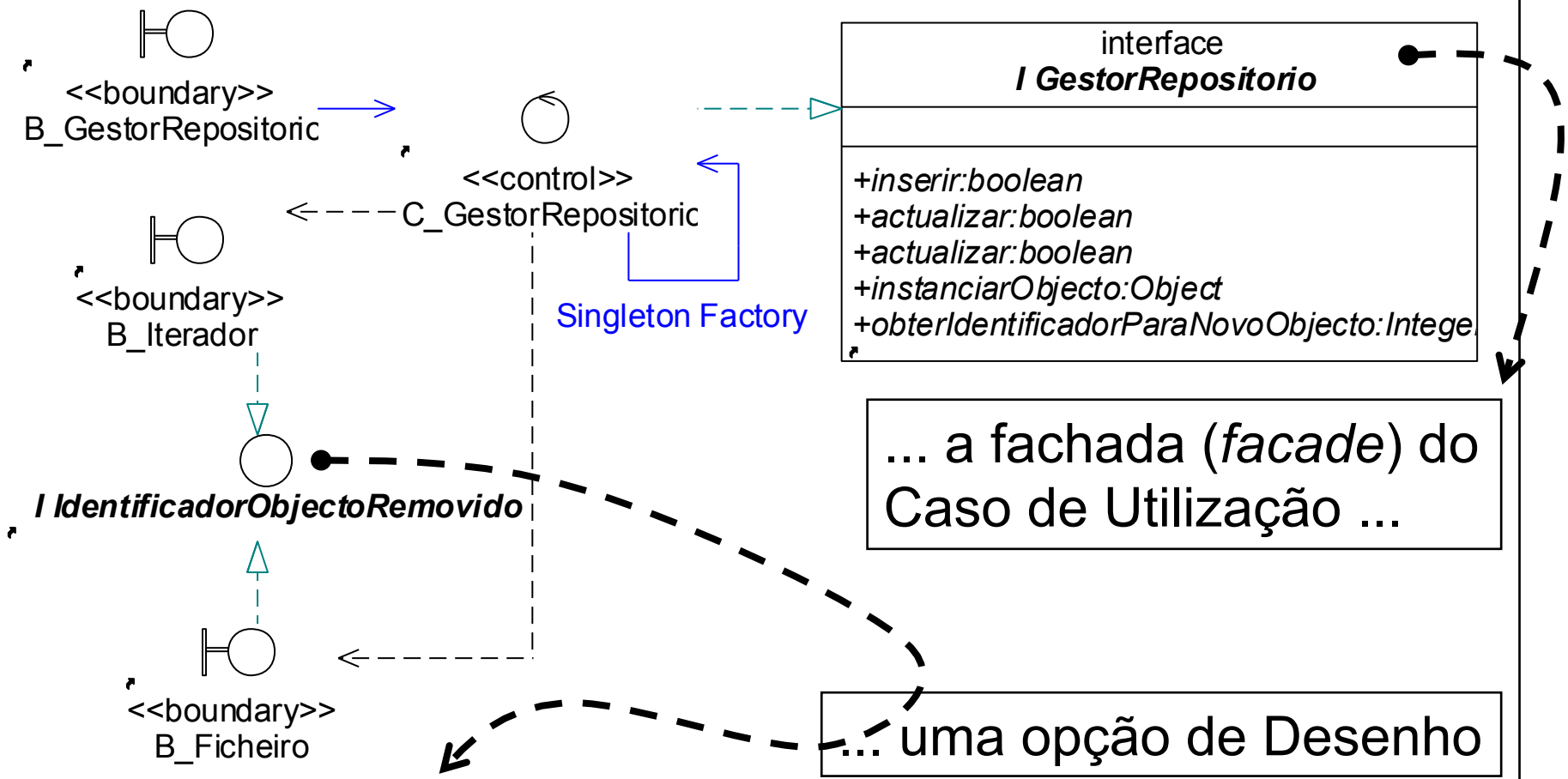
Subsistemas do *framework* "persistência"



Caso de Utilização (Gerir Repositório)



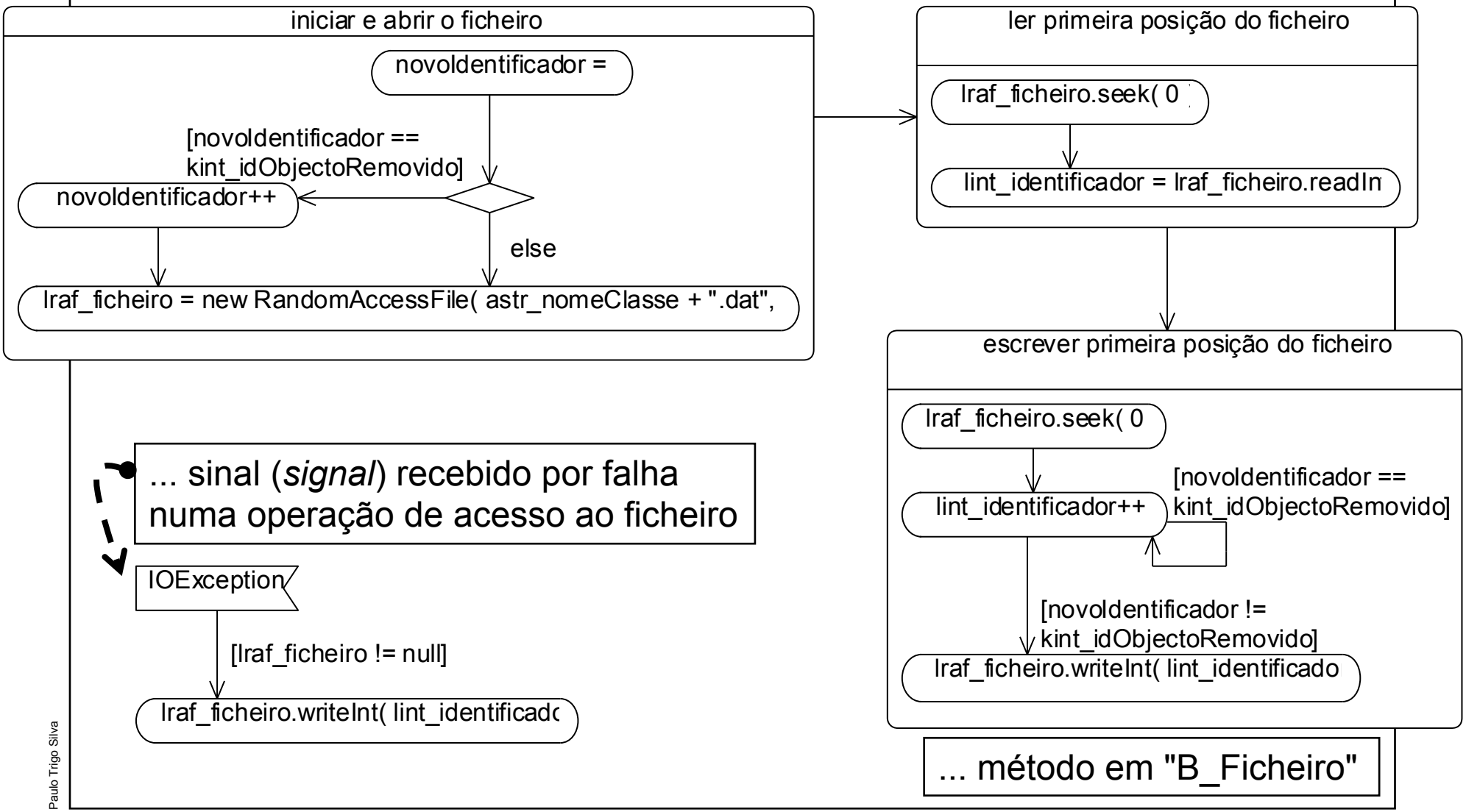
Análise – Diagrama de Classes (Gerir Repositório)



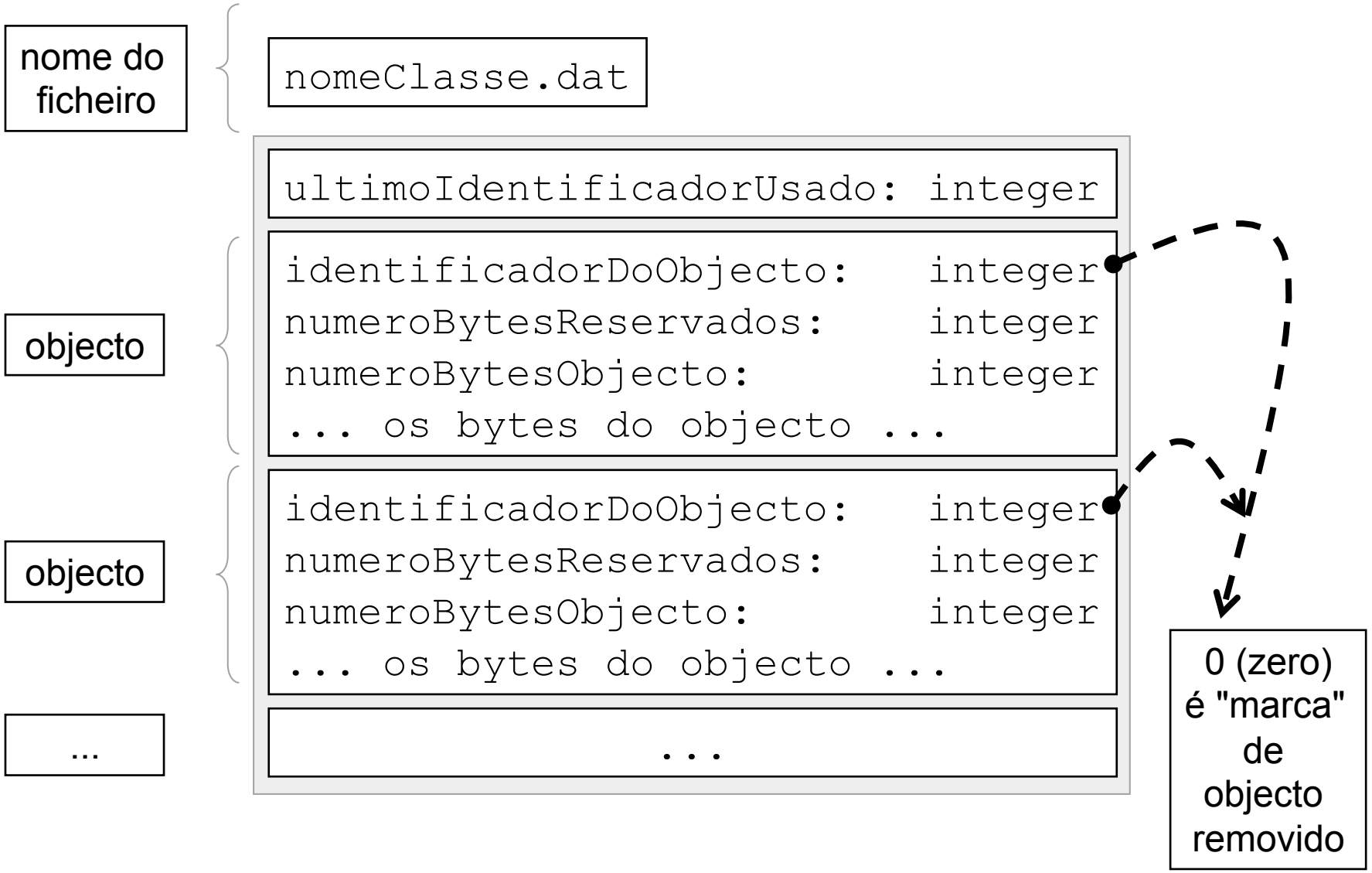

```
interface I_IdentificadorObjectoRemovido
{
    public final static int kint_idObjectoRemovido = 0;
}
```

Paulo Trigo Silva

Desenho – "atribuir identificador único"



Desenho – "definir formato de gravação"



Desenho – "inserir objecto"