PROGRAMAÇÃO ORIENTADA PELOS OBJECTOS

Aula 1

Desenvolvimento Orientado pelos Objectos

Metodologia de trabalho

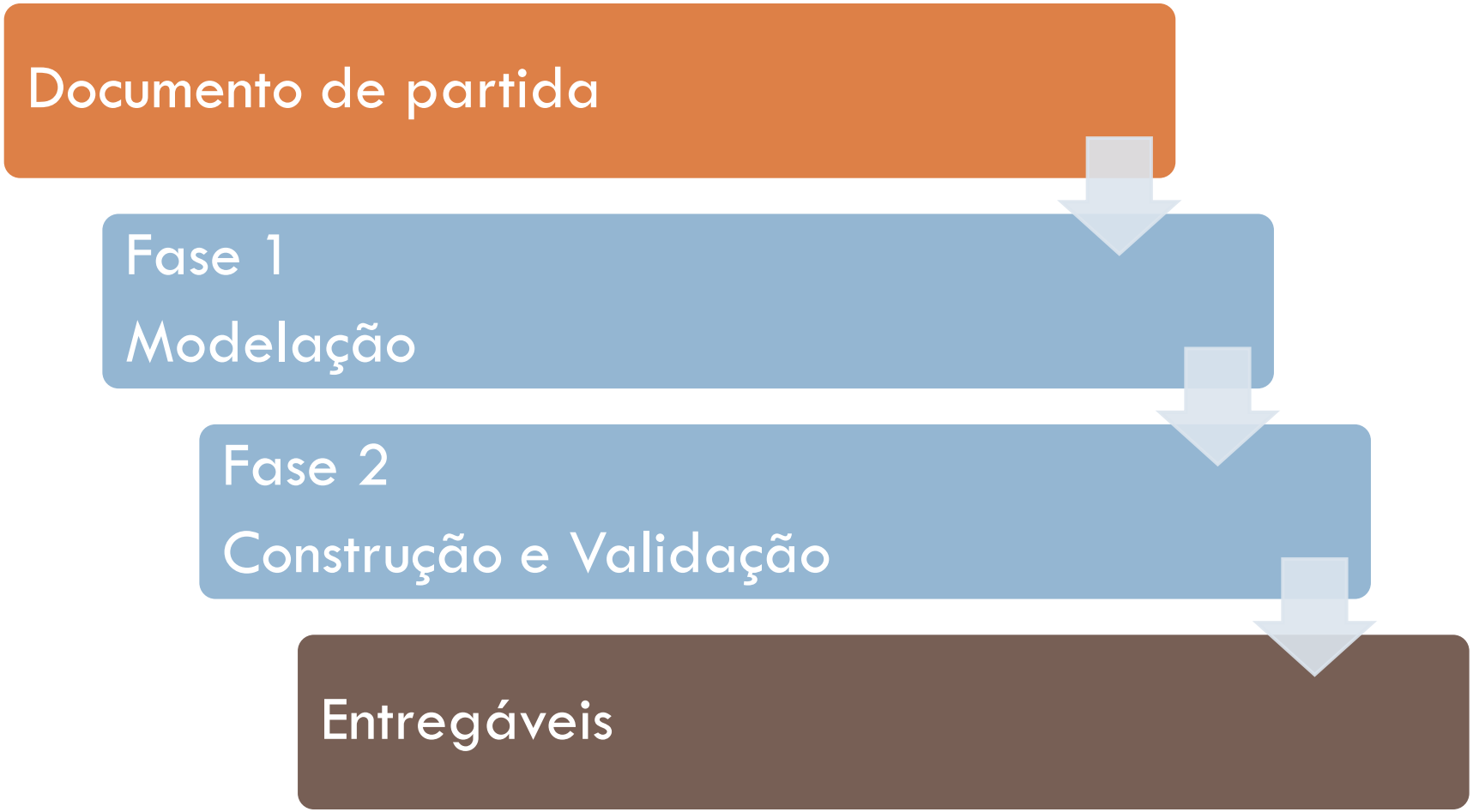
- Vamos estabelecer algumas regras para a resolução de problemas, usando a Programação Orientada pelos Objectos

StupidFriendsBook

Abordagem ao desenvolvimento de projectos em POO

- Documento de partida
- Fase 1: Modelação
- Fase 2: Construção e Validação
- Entregáveis

Abordagem ao desenvolvimento de projectos em POO



Documento de partida

- **Descrição informal** do sistema / aplicação, numa perspectiva de **alto nível**
  - Descrição dos **objectivos gerais** do sistema/aplicação
  - Descrição mais detalhada das **funcionalidades**
  - Descrição de **cenários** de utilização/interacção

Modelação (i)

- Análise e modelação do sistema a construir
  - A aplicação deve realizar **um conjunto de funcionalidades**. Tais funcionalidades têm que ser **identificadas** e **nomeadas**
    - Quais são?
  - Cada funcionalidade tem que ser **caracterizada**
    - O que faz?
  - O conjunto das funcionalidades permite definir uma **interface de topo**
    - Esta interface poderá mais tarde ser **refinada**

Modelação (ii)

- Análise e modelação do sistema a construir (cont.)
  - Deve-se **assegurar** (explicar / justificar) que as operações dessa **interface** de topo permitem realizar os cenários de utilização / interacção
  - Análise da realização de cada operação / funcionalidade de topo com a introdução de:
    - entidades (que encapsulam dados e expõem operações)
    - as operações das entidades
  - Por enquanto, preocupamo-nos sobretudo com o **encapsulamento** e a **ocultação de informação**

Modelação (iii)

- Output da fase de modelação:
  - Identificação da **interface da aplicação**
    - explicação informal da funcionalidade de cada operação
  - **Identificação das entidades do modelo**
    - Nome
    - Lista de operações
    - Explicação informal do efeito de cada operação de cada entidade

Modelação (iv)

- Output da fase de modelação (cont.):
  - **Explicação** de como as interfaces, operações e entidades identificadas contribuem para realizar as funcionalidades caracterizadas anteriormente
  - **Visão organizada da estrutura do sistema / aplicação** a desenvolver em termos de
    - uma unidade principal (interface de topo)
    - um conjunto de entidades e suas operações
      - Independente da representação de dados
      - Independente de algoritmos utilizados, na medida do possível.

Construção e Validação

- Começar pelo "esqueleto da aplicação"
- As interfaces, entidades e operações desenhadas na fase anterior devem ser implementadas
  - Identificar as classes que as suportam
  - Escolher, para cada classe, uma representação adequada
    - Operações de interface são públicas
    - Operações auxiliares devem ficar escondidas (privadas)
    - Variáveis e constantes para representar o estado
      - Para cada uma indicar o seu tipo e finalidade
        - Podemos usar objectos de um tipo como valores de variáveis usadas dentro de outros objectos

Construção e Validação

- Identificar entidades auxiliares
  - Exº: Colecções de objectos de um determinado tipo
- Nesta altura, deverá ter um esqueleto da sua aplicação
  - Interfaces
  - Classes
    - Métodos (por enquanto, vazios)
    - Constantes e variáveis
    - Documentação
- Programa ainda não passa no compilador

Construção e Validação

- Construção do sistema
  - Não tentar fazer tudo ao mesmo tempo!
  - Definir sequência de implementação que permita ir validando incrementalmente o seu programa
  - Exemplo:
    - Interpretador de comandos
      - Programa aceita comandos e dados associados, mas (ainda) não faz nada com eles
    - Implementação de comandos
      - Escolher uma sequência para a implementação de comandos que permita ir testando os comandos, à medida que os implementa

Construção e Validação

- Indicações gerais
  - Tente ter sempre uma versão estável do seu programa
    - Compila sem erros
    - Implementa algumas das funcionalidades do seu programa
    - Está bem testada
      - Teste as suas classes, método a método
      - Se necessário, construa pequenos programas de teste
  - Não avance para novas fases do desenvolvimento, sem as anteriores estarem sólidas
  - Use sempre o melhor estilo de programação e documentação possível

Entregáveis

- No final do desenvolvimento, deve disponibilizar
  - Código fonte do seu programa, devidamente comentado
  - Documentação completa
    - Javadoc
    - Diagramas de classes e interfaces
    - Outros que lhe sejam solicitados
      - Manual de instalação e utilização
      - ...

The StupidFriendsBook

Uma rede social para descobrir a essência dos seus amigos. Serão eles inteligentes, estúpidos, anjinhos, ou bandidos?

StupidFriendsBook

O que é o StupidFriendsBook ?

- Imagine uma rede social onde pode votar, para cada acção de cada um dos seus amigos, se considera que ela é **boa, ou má, para o seu amigo** e **boa, ou má, para os outros** que sentem as consequências dessa acção
- Nesta rede social, todos podem votar nas acções de todos
- Os votos acumulados permitem construir o perfil das pessoas

O que é o StupidFriendsBook ?

- Cada pessoa no StupidFriendsBook pode ter várias acções, cada uma delas votadas individualmente
- A média das votações em cada acção determina se o perfil de uma pessoa é visto como **inteligente**, **estúpido**, **anjinho** ou **bandido**; é possível consultar essa informação sobre qualquer pessoa
- A pessoa **mais aborrecida** é a pessoa cujas acções são o mais neutras possível
  - Ou seja, cujo efeito médio das acções, para si própria e para os outros, seja o mais próximo de 0 (nem positiva, nem negativa)
- Podemos listar pessoas com determinado perfil

Construção da personalidade

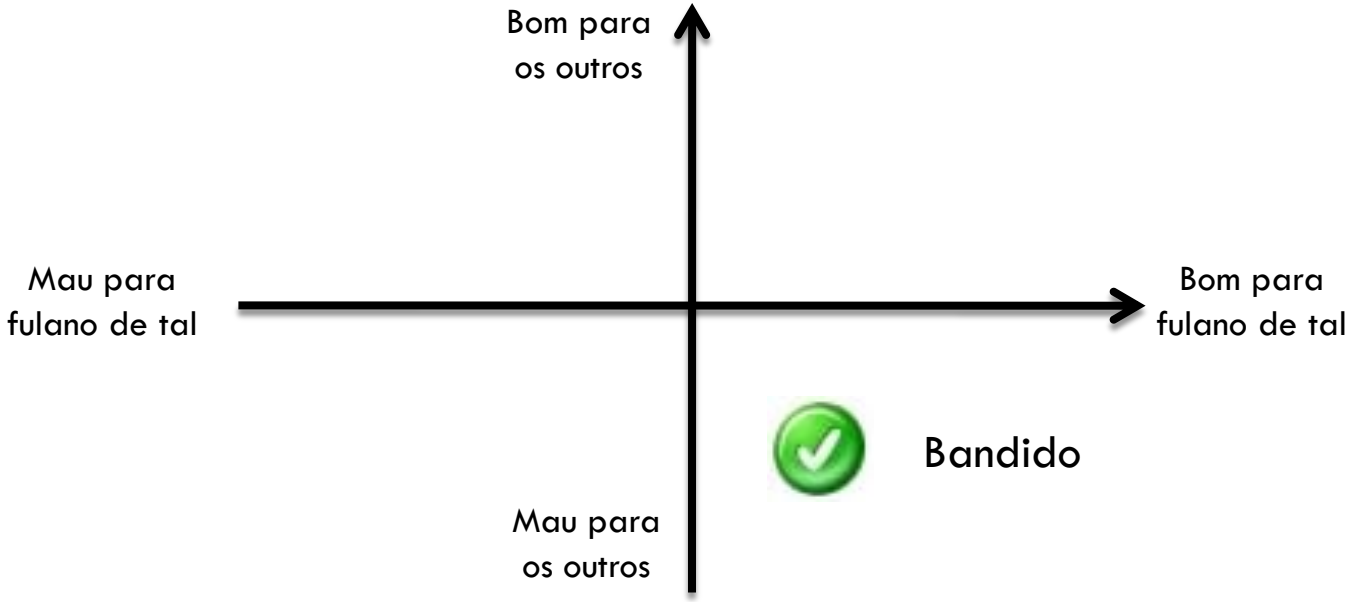
- Pessoa faz acções
- Cada vez que alguém vota numa dessas acções, indica se considera se a acção foi
  - Boa, ou má, para quem a praticou
  - Boa, ou má, para os outros
- Os votos vão-se acumulando
  - Se houver mais votos bons que maus para o próprio, ou pelo menos igual número, considera-se a acção boa para o próprio
  - Raciocínio semelhante, nos votos para os outros
- A média das votações determina a personalidade

Fulano de tal é inteligente?

- Fulano de tal fez frito e cozido
  - Bom para ele?
  - Bom para os outros?

Fulano de tal é inteligente?

- Fulano de tal fez frito e cozido
  - Bom para ele?
  - Bom para os outros?



Fulano de tal é inteligente?

- Fulano de tal fez frito e cozido
  - Bom para ele?
  - Bom para os outros?

Fulano de tal é inteligente?

- Fulano de tal fez frito e cozido
  - Bom para ele?
  - Bom para os outros?

Traço de execução

>Novo
>Adiciona
Fulano
Ok
>Adiciona
Beltrano
Ok
>Adiciona
Sicrano
Ok
>Adiciona
Beltrano
Beltrano já existe
>Faz
Fulano
Lava pratos
Ok
>Faz
Fulano
Jardinagem
Ok

>Adiciona
Mais Um
Ok
>Remove
Mais Um
Ok
>Faz
Fulano
Lava pratos
Fulano já Lava pratos
>Faz
Beltrano
Pinta grafitti
Ok
>Faz
Sicrano
Pinta grafitti
Ok
>Vota
Fulano
Lava pratos

Bem
Mal
Ok
>Vota
Fulano
Lava pratos
Bem
Bem
Ok
>Vota
Fulano
Lava pratos
Bem
Bem
Ok
>Vota
Fulano
Jardinagem
Bem
Bem
Ok

>Vota
Beltrano
Lava pratos
Bem
Bem
Beltrano não Lava pratos
>Vota
Beltrano
Pinta grafitti
Mal
Mal
Ok
>Lista
Inteligente
Fulano;Sicrano
>Personalidade
Beltrano
Estúpido
>Aborrecido
Sicrano
>Sair
Adeus

Identificando as acções do sistema

- Imagine uma rede social onde pode **votar**, para cada acção de cada um dos seus amigos, se considera que ela é **boa, ou má, para o seu amigo** e **boa, ou má, para os outros** que sentem as consequências dessa acção
- Nesta rede social, todos podem votar nas acções de todos
- Os votos acumulados permitem construir o perfil das pessoas
- Cada pessoa no StupidFriendsBook pode ter várias acções, cada uma delas votadas individualmente
- A média das votações em cada acção determina se o perfil de uma pessoa é visto como **inteligente, estúpido, anjinho** ou **bandido**; é possível consultar essa informação sobre qualquer pessoa
- A pessoa **mais aborrecida** é pessoa cujas acções são o mais neutras possível
  - Ou seja, cujo efeito médio das acções, para si própria e para os outros, seja o mais próximo de 0 (nem positiva, nem negativa)
- Podemos listar pessoas com determinado perfil

Acções do sistema:

Criar rede social
Criar pessoa
Remover pessoa
Criar acção
Votar em accção
Consultar personalidade
Descobrir mais aborrecido
Listar pessoas por perfil
Sair

Comandos a disponibilizar

- Novo (Criar rede social)
  - Cria um novo StupidFriendsBook, que começa por estar vazio
- Adiciona amigo (Criar pessoa)
  - Acrescenta um novo amigo ao StupidFriendsBook; se já existir, não faz nada
    - Informação que caracteriza o amigo: nome
- Remove amigo (Remover pessoa)
  - Remove um amigo do StupidFriendsBook; se não existir, não faz nada
    - Informação que caracteriza o amigo: nome

Comandos a disponibilizar

- Acção (Criar acção)
  - Adiciona uma nova acção ao amigo
    - Informação que caracteriza o amigo: nome
    - Informação que caracteriza a acção: descrição
- Vota (Votar em acção)
  - Vota numa determinada acção de um amigo
    - Informação que caracteriza o amigo: nome
    - Informação que caracteriza a acção: descrição, impacto no próprio e impacto nos outros
      - Ao votar, ou se considera uma acção benéfica, ou prejudicial

Comandos a disponibilizar

- Personalidade (Consultar personalidade)
  - Consultar a personalidade de uma pessoa
    - Informação que caracteriza a consulta: nome da pessoa
- Aborrecido (Descobrir mais aborrecido)
  - Lista a pessoa "mais aborrecida" no StupidFriendsBook
- Lista amigos (Listar pessoas por perfil)
  - Lista todos os amigos com um determinado traço de personalidade dominante (por exemplo, lista todos os anjinhos)
    - Informação que caracteriza a listagem: traço de personalidade comum às pessoas listadas
- Sair
  - Termina o programa

Traço de execução

>Novo
>Adiciona
Fulano
Ok
>Adiciona
Beltrano
Ok
>Adiciona
Sicrano
Ok
>Adiciona
Beltrano
Beltrano já existe
>Faz
Fulano
Lava pratos
Ok
>Faz
Fulano
Jardinagem
Ok

>Adiciona
Mais Um
Ok
>Remove
Mais Um
Ok
>Faz
Fulano
Lava pratos
Fulano já Lava pratos
>Faz
Beltrano
Pinta grafitti
Ok
>Faz
Sicrano
Pinta grafitti
Ok
>Vota
Fulano
Lava pratos

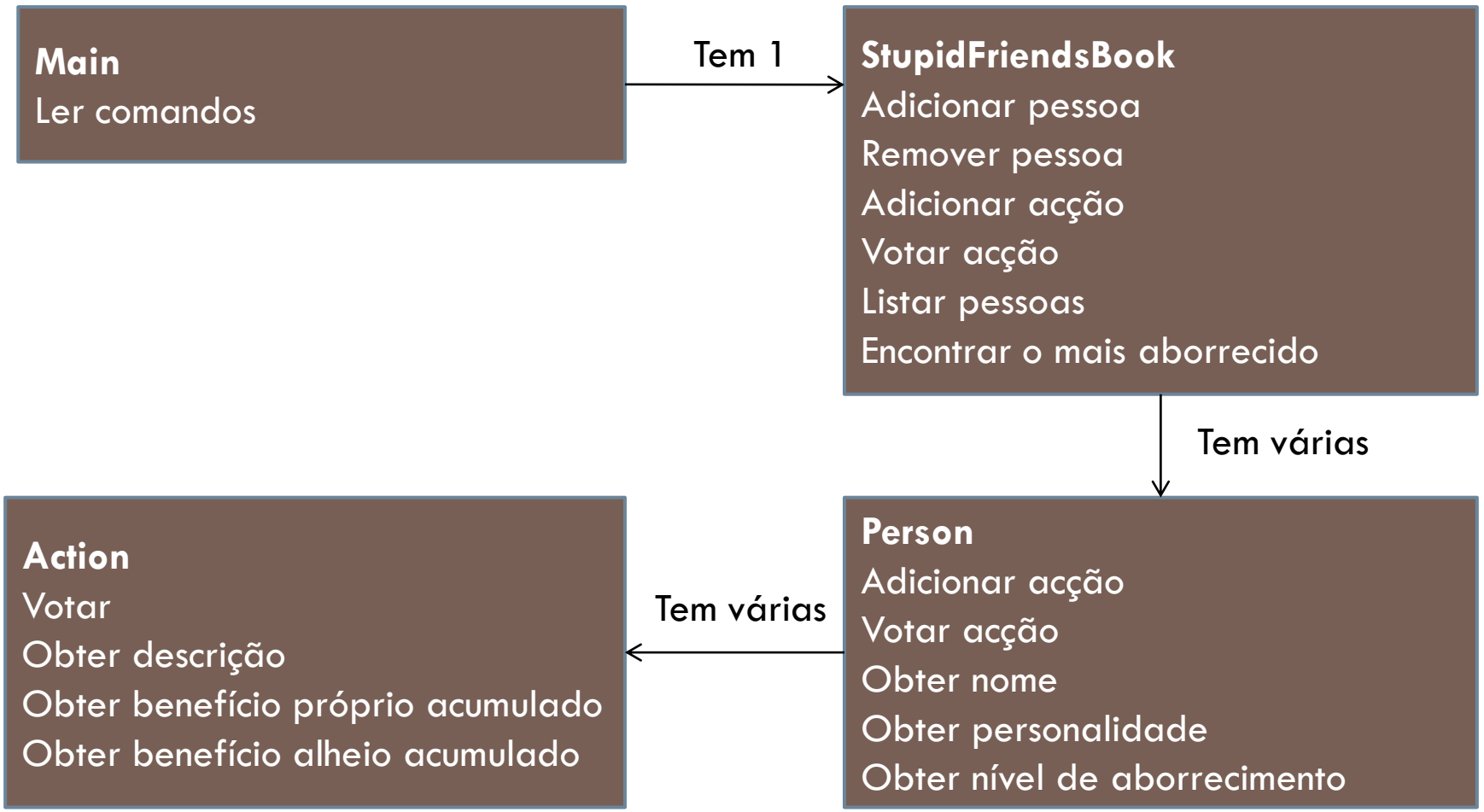
Bem
Mal
Ok
>Vota
Fulano
Lava pratos
Bem
Bem
Ok
>Vota
Fulano
Lava pratos
Bem
Bem
Ok
>Vota
Fulano
Jardinagem
Bem
Bem
Ok

>Vota
Beltrano
Lava pratos
Bem
Bem
Beltrano não Lava pratos
>Vota
Beltrano
Pinta grafitti
Mal
Mal
Ok
>Lista
inteligente
Fulano;Sicrano
>Personalidade
Beltrano
Estúpido
>Aborrecido
Sicrano
>Sair
Adeus

Acções do sistema:

- ✓ Criar rede social
- ✓ Criar pessoa
- ✓ Remover pessoa
- ✓ Criar acção
- ✓ Votar em accção
- ✓ Consultar personalidade
- ✓ Descobrir mais aborrecido
- ✓ Listar pessoas por perfil
- ✓ Sair

Entidades

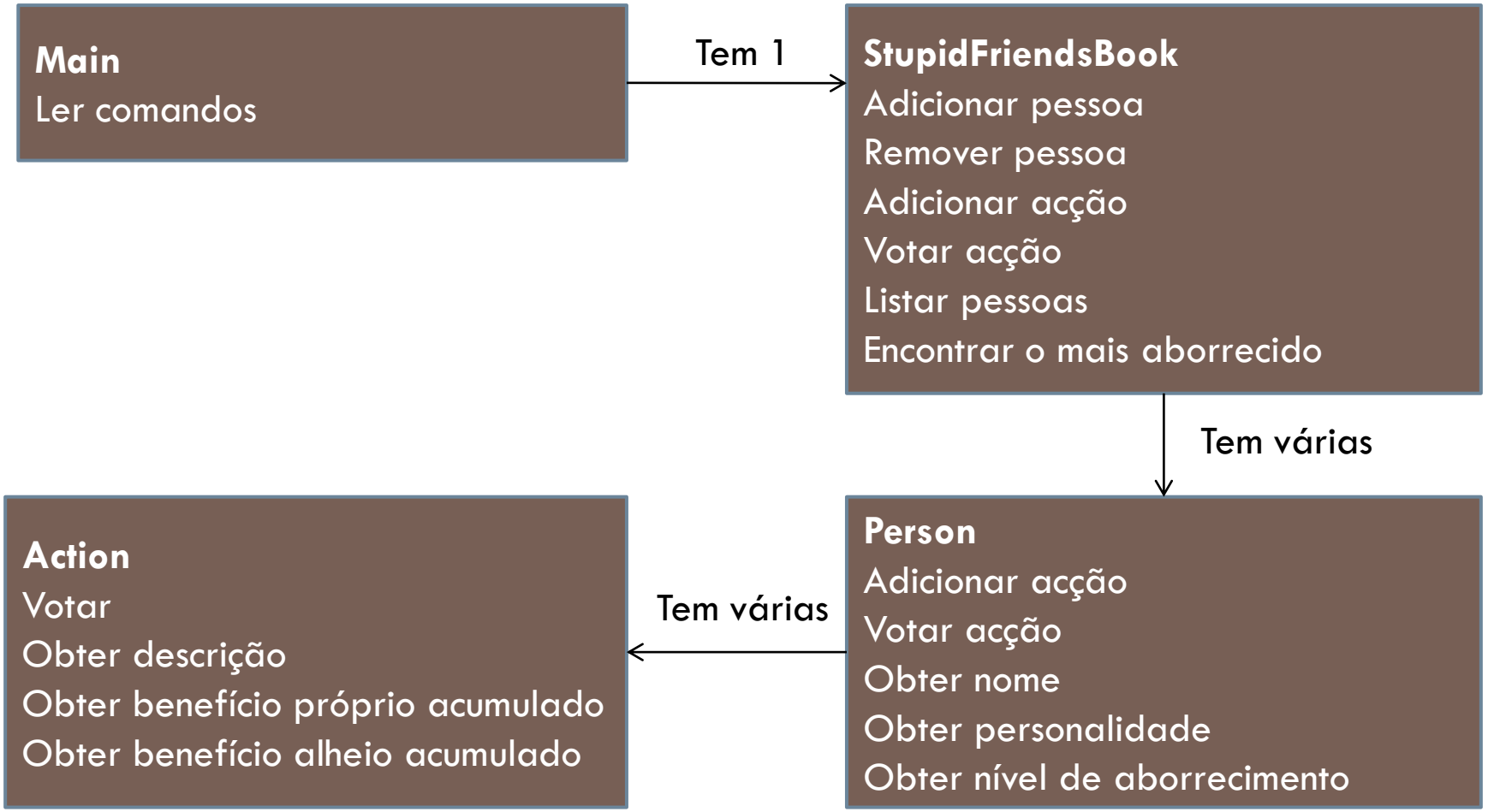

Main
Ler comandos
Tem 1
StupidFriendsBook
Adicionar pessoa
Remover pessoa
Adicionar acção
Votar acção
Listar pessoas
Encontrar o mais aborrecido
Tem várias
Person
Adicionar acção
Votar acção
Obter nome
Obter personalidade
Obter nível de aborrecimento
Tem várias
Action
Votar
Obter descrição
Obter benefício próprio acumulado
Obter benefício alheio acumulado

PROGRAMAÇÃO ORIENTADA PELOS OBJECTOS

Aula 2

Noção de Interface

Entidades


Main
Ler comandos
Tem 1
StupidFriendsBook
Adicionar pessoa
Remover pessoa
Adicionar acção
Votar acção
Listar pessoas
Encontrar o mais aborrecido
Tem várias
Action
Votar
Obter descrição
Obter benefício próprio acumulado
Obter benefício alheio acumulado
Tem várias
Person
Adicionar acção
Votar acção
Obter nome
Obter personalidade
Obter nível de aborrecimento

Que operações criar?

- Vamos listar as principais operações, como faríamos em Introdução à Programação...

Main


```
void main(String[] args)
```

- Programa principal, chama interpretador de comandos `commands`

```
void commands()
```

- Operação auxiliar que implementa o interpretador de comandos

StupidFriendsBook

`void reset()`

- Limpa a lista de amigos, para usar quando queremos uma nova lista

`boolean addFriend(String name)`

- Adiciona o amigo de nome name, devolvendo `true`, se conseguir, ou `false`, se o amigo já existe

`boolean removeFriend(String name)`

- Remove o amigo de nome name, devolvendo `true`, se conseguir, ou `false`, se o amigo não existia

StupidFriendsBook


```
int doSomething(String name, String description)
```

- Acrescenta a acção `description` à pessoa de nome `name`, devolvendo o código `OK`, se acrescentar a acção com sucesso, `NO_ACTION`, se o comando falhar porque a pessoa já tinha essa acção, ou `NO_FRIEND`, se a pessoa não existir

```
int vote(String name, String description,
         boolean goodForPerson,
         boolean goodForOthers)
```

- Vota numa acção, devolvendo o código `OK`, se votar na acção com sucesso, `NO_ACTION`, se o voto falhar por não existir a acção, ou `NO_FRIEND`, se o voto falhar por não existir a pessoa

```
String mostBoringFriend()
```

- Devolve o nome da pessoa mais aborrecida, ou a `String` vazia, se não houver amigos

StupidFriendsBook

`void initialize()`

- Inicializa o iterador de amigos

`boolean hasNext()`

- Devolve `true` se houver mais pessoas a iterar, ou `false` caso contrário

`Person next()`

- Devolve a próxima pessoa a iterar, avançando com o iterador, ou `null`, se não houver mais pessoas para iterar

Person

`boolean addAction(String description)`

- Acrescenta uma acção `description` à pessoa, desde que ela ainda não a tenha; devolve `true` se a acção é acrescentada, ou `false`, se já existia

`boolean vote(String description, boolean goodForPerson, boolean goodForOthers)`

- Vota na acção `description`, incrementando os votos positivos nos benefícios para a própria pessoa, se `goodForPerson` for `true`, ou decrementando-os, se for `false`, e fazendo o mesmo com os votos nos benefícios ou prejuízos causados a outros, neste caso com base em `goodForOthers`

Person

`String getName()`

- Devolve o nome da pessoa

`int getPersonality()`

- Devolve um inteiro representando o traço de personalidade dominante, com os valores `INTELIGENT`, `STUPID`, `BANDIT`, ou `ANGEL`

`double getBoredom()`

- Devolve o valor da "distância" dos votos acumulados à origem usando o teorema de Pitágoras, sendo a "distância" a "hipotenusa" e os valores acumulados de benefícios próprios e alheios os "catetos"

Action


```
void vote(boolean goodForPerson, boolean goodForOthers)
```

- Vota na acção, incrementando os votos nos benefícios próprios se `goodForPerson` for `true`, ou decrementando-os, caso contrário, e fazendo o mesmo em relação aos benefícios alheios, com o argumento `goodForOthers`

```
String getDescription()
```

- Devolve a descrição da acção

```
int getSelfBenefit()
```

- Devolve o total acumulado de votos nos benefícios próprios

```
int getOthersBenefit()
```

- Devolve o total acumulado de votos nos benefícios causados a outros

Como representar as entidades?

- Em IP usaríamos classes
- Em POO, usaremos interfaces
- Para isso, temos de definir o conceito de interface

Assinatura de um método

- A **Assinatura** de um método é constituída por:
  - Nome do método
  - Número, tipo e ordem dos parâmetros
- O tipo do resultado **não faz parte** da assinatura

- Em Java, os métodos são identificados pelas suas assinaturas
  - Podem existir vários métodos com o mesmo nome, desde que com assinaturas diferentes
    - Exº: vários construtores com listas de argumentos distintas

Protocolo

- **Protocolo** de um objecto é o conjunto das assinaturas dos seus métodos públicos
  - Determina o reportório de mensagens que esse método entende

- Um objecto **suporta um determinado protocolo** se definir todos os métodos públicos previstos nesse protocolo (pode definir outros, além desses)

Tipo-Objecto

- Um **tipo-objecto** representa um conjunto de objectos
- Normalmente, nas linguagens Orientadas pelos Objectos consideram-se duas filosofias:
  - Tipos-Objecto **Sintácticos**
    - Cada tipo-objecto é caracterizado por um **protocolo**
      - Um tipo-objecto representa o conjunto de todos os objectos que suportam esse protocolo, independentemente da representação interna dos objectos e significado das operações
  - Tipos-Objecto **Semânticos**
    - Cada tipo-objecto caracteriza-se por uma **implementação completa**, ou seja, por uma classe
      - Um tipo-objecto representa o conjunto de todos os objectos gerados por essa classe

Interface

- Em Java, um tipo-objecto sintáctico chama-se **interface**
- Uma **interface** representa todos os objectos que suportam um dado protocolo e que sejam instâncias de classes que declarem explicitamente implementar essa interface
- Ao contrário do que acontece com as classes, **as interfaces não fornecem uma implementação**

Definição de uma interface em Java

- Sintaxe

```
public interface interfaceName {
    // constantes
    // assinaturas de métodos
}
```

- Exemplo

```
public interface IntegerSet {
  int size();
  int get(int i);
  int add(int n);
  boolean belongs(int n);
  IntegerSet union(IntegerSet union);
}
```

- Os métodos são sempre **abstractos** e **públicos**

Interface vs. Classe

- Todos os métodos de uma interface são **abstractos**
  - Têm nome, parâmetros, tipo de retorno, mas **não têm uma implementação**
- Todos os métodos de uma interface são **públicos**
- Uma interface **não tem variáveis de instância**
- Ao contrário do que acontece com as classes, **não se pode instanciar uma interface**

Implementação de uma interface em Java

- Sintaxe

```
public class className implements interfaceName {
  // atributos e constantes
  // métodos
}
```

- Exemplo

```
public class IntegerSetClass implements IntegerSet {
  // atributos and constantes
  public int size() { ... }
  public int get(int i) { ... }
  public int add(int n) { ... }
  public boolean belongs(int n) {...}
  public IntegerSet union(IntegerSet union) {...}
}
```

- Classe implementa **todos** os métodos da interface

Classe

- Em Java os tipos-objecto semânticos são as **classes**
- Isso significa que, em Java, as classes, além de serem mecanismos geradores de objectos, também são tipos.

Opção metodológica em POO

- Nesta cadeira vamos adoptar a filosofia dos tipos-objecto sintácticos
- No código que escrevermos, trataremos as nossas interfaces como tipos e as nossas classes como implementações
- Vantagens:
  - O nosso código não assume compromissos escusados
  - Ao declararmos as nossas variáveis, parâmetros e resultados usando exclusivamente nomes de interfaces, o nosso código torna-se compatível com todas as classes que implementem essas interfaces (classes existentes ou classes a criar futuramente)

Utilização de classes e interfaces

1. Escrever a interface que captura o protocolo pretendido para os objectos

2. Escrever a classe que implementa a interface definida no ponto anterior

3. Usar o nome da interface como tipo, para declarar variáveis, parâmetros e resultados, e o nome da classe apenas para a instanciação (ao criar objectos com o operador **`new`**)

Erros típicos com interfaces

- Esquecermo-nos de implementar métodos na classe que implementa a interface como public
  - Os métodos de uma interface não se declaram públicos explicitamente, porque já são públicos, por omissão
  - Ao contrário, por omissão, a visibilidade de um método de uma classe não é public, mas sim "package"
    - Implementar com visibilidade "package" reduziria a visibilidade do método, definida na interface como public

```
public class IntegerSetClass implements IntegerSet {
  int size() {...} // ERRADO! Devia ser público!
}
```

Erros típicos com interfaces

- Tentar instanciar uma interface:
  - Podemos definir variáveis cujo tipo é uma interface

```
IntegerSet s; // Ok (no exemplo, IntegerSet é uma interface)
```

  - Nunca podemos construir uma interface

```
IntegerSet s = new IntegerSet(); // ERRADO!
```

  - Sempre que um objecto é declarado como sendo do tipo de uma interface, esse objecto pertence a uma classe que implementa essa interface

```
IntegerSet s = new IntegerSetClass(); // Ok
```

De volta ao StupidFriendsBook

-Como tirar partido das interfaces na resolução deste problema?

Esqueleto da classe Main


```
public class Main {
  // Comandos do utilizador
  private static final String QUIT = "Sair";
  // ...

  // Feedback dado pelo programa
  private static final String PROMPT = "> ";
  // ...


  public static void main(String[] args) {
    Main.commands();
  }

  // ...
```

Esqueleto da classe Main


```
  private static void commands() {
    StupidFriendsBook fb = new StupidFriendsBookClass();
    Scanner in = new Scanner(System.in);
    String command = getCommand(in);
    while (!command.equalsIgnoreCase(Main.QUIT)) {
      // ... Interpretar comandos aqui...
      command = Main.getCommand(in);
    }
    System.out.println(Main.BYE);
  }

  private static String getCommand (Scanner in) {
    System.out.print(Main.PROMPT);
    return in.nextLine();
  }
}
```

Interface StupidFriendsBook


```
public interface StupidFriendsBook {
  // Constantes com códigos de erro
  int OK = 0;
  int NO_FRIEND = -1;
  int NO_ACTION = -2;

  void reset();
  boolean addFriend(String name);
  boolean removeFriend(String name);
  int doSomething(String name, String description);
  boolean vote(String name, String description,
               boolean goodForPerson, boolean goodForOthers);
  String mostBoringFriend();
  void initialize();
  boolean hasNext();
  Person next();
}
```

Esqueleto da classe StupidFriendsBookClass


```
public class StupidFriendsBookClass
             implements StupidFriendsBook {
  private static final int DEFAULT = 10;
  private Person[] friends;
  private int counter;
  private int current;

  public StupidFriendsBookClass(){...}
  public void reset() {...}
  public boolean addFriend(String name) {...}
  public boolean removeFriend(String name) {...}
  public int doSomething(String name, String description)
  {...}
  // continua...
```

Esqueleto da classe StupidFriendsBookClass


```
  // Continuação da implementação
  public boolean vote(String name, String description,
          boolean goodForPerson, boolean goodForOthers){...}
  public String mostBoringFriend() {...}

  // Métodos para a iteração
  public void initialize() {...}
  public boolean hasNext() {...}
  public Person next() {...}

  // Métodos auxiliares
  private int indexOf(String name) {...}
}
```

Interface Action


```
public interface Action {
  void vote(boolean goodForPerson, boolean goodForOthers);
  String getDescription();
  int getSelfBenefit();
  int getOtherBenefitScore();
}
```

Esqueleto da classe ActionClass


```
public class ActionClass implements Action {
  private String description;
  private int selfBenefitScore;
  private int otherBenefitScore;

  public Action(String description){...}

  public void vote(boolean goodForPerson,
                   boolean goodForOthers){...}

  public String getDescription(){...}

  public int getSelfBenefit(){...}

  public int getOtherBenefitScore(){...}
}
```

Interface Person


```
public interface Person {
  // Constantes de perfis de personalidade
  int INTELLIGENT = 0;
  int ANGEL = 1;
  int BANDIT = 2;
  int STUPID = 3;

  // Métodos da interface
  boolean addAction(String description);
  boolean vote(String description, boolean goodForPerson,
               boolean goodForOthers);
  String getName();
  int getPersonality();
  double getBoredom();
}
```

Esqueleto da classe PersonClass


```
public class PersonClass implements Person{
  // Constantes INTELLIGENT, ANGEL, BANDIT e STUPID
  private Action[] myActions;
  private int counter;

  public Person(String name) {...}

  public boolean addAction(String description) {...}

  public boolean vote(String description,
           boolean goodForPerson, boolean goodForOthers) {...}

  public String getName() {...}

  public int getPersonality(){...}

  public double getBoredom() {...}
}
```