

PROGRAMAÇÃO ORIENTADA PELOS OBJECTOS

Noção de Interface

Que operações criar?

- Vamos listar as principais operações

Main

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

32

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

- Programa principal, chama interpretador de comandos

StupidFriendsBook

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

33

void reset()

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

boolean hasFriend(String name)

- Devolve true se o amigo de nome name existe, ou false caso contrário

void addFriend(String name)

- Adiciona o amigo de nome name
- Pre: !hasFriend(name) – Não existe nenhum amigo com esse nome

void removeFriend(String name)

- Remove o amigo de nome name
- Pre: hasFriend(name) – Existe um amigo com esse nome

StupidFriendsBook

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

34

boolean hasAction(String name, String description)

- Devolve **true** caso exista uma acção com descrição **description** feita por um amigo de nome **name**, ou **false** caso contrário

void addAction(String name, String description)

- Acrescenta a acção **description** ao amigo de nome **name**
- Pre: hasFriend(name) && !hasAction(name, description)

void vote(String name, String description,
 boolean goodForPerson, **boolean** goodForOthers)

- Vota na acção **description** feita pelo amigo de nome **name**
- Pre: hasAction(name, description)

StupidFriendsBook

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

35

int numberOfFriends ()

- Devolve o número de amigos na rede social

String mostBoringFriend ()

- Devolve o nome da pessoa mais aborrecida
- **Pre:** numberOfFriends () > 0 – a rede social não está vazia

StupidFriendsBook

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

36

Precisamos também de métodos que permitam iterar o conteúdo do StupidFriendsBook

void initialize(int kind)

- Inicializa o iterador de amigos para a personalidade `kind`
- **Pre:** `(kind == INTELIGENT) || (kind == STUPID) || (kind == BANDIT) || (kind == ANGEL)`

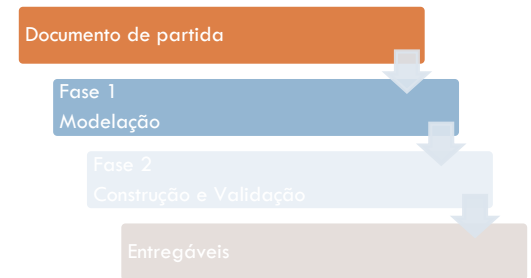
boolean hasNext()

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

String next()

- Devolve o nome da próxima pessoa com personalidade `kind` a iterar, avançando com o iterador
- **Pre:** `hasNext()`

Person



37

boolean hasAction(String description)

- Devolve true se a pessoa tem a acção description

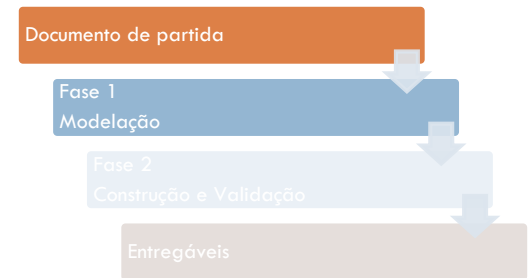
void addAction(String description)

- Acrescenta a acção description à pessoa
- Pre: !hasAction(description)

void 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.
- Pre: hasAction(description)

Person



38

`String getName()`

- Devolve o nome da pessoa

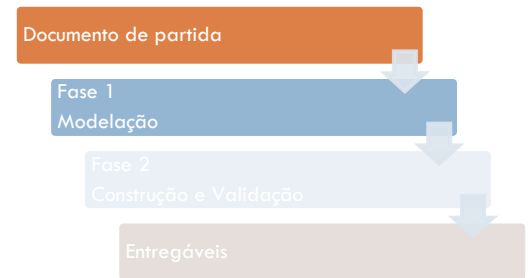
`int getPersonality()`

- Devolve um inteiro representando o traço de personalidade dominante, com os valores INTELLIGENT, 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



39

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

41

- A **assinatura** de um método é constituída por:
 - Nome do método
 - Número, tipo e ordem dos parâmetros
- 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

42

- **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

43

- 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

44

- 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

45

○ 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**

Definição de uma interface em Java

46

○ Sintaxe

```
public interface InterfaceName {  
    // constantes: valores public static final  
    // assinaturas de métodos: public abstract  
}
```

○ Exemplo

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

Os classificadores **public** e **abstract** podem ser incluídos ou não, mas as assinaturas duma interface são sempre **public abstract**.

Interface vs. Classe

47

- 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

48

○ Sintaxe

```
public class ClassName implements InterfaceName {  
    // atributos e constantes  
    // métodos  
}
```

○ Exemplo de classe implementadora

```
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

49

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

Opção metodológica em POO

50

- 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 desses tipos
- 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)

Opção metodológica em POO

51

- Interfaces são tipos e as classes são implementações desses tipos
- Vantagens:
 - O código não assume compromissos escusados
 - Ao declarar variáveis, parâmetros e resultados usando exclusivamente nomes de interfaces, o código torna-se compatível com todas as classes que implementem essas interfaces (classes existentes ou classes a criar futuramente)


Utilização de classes e interfaces

52

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

53

- 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
 - Mas numa classe, a visibilidade por omissão de um método não é public mas sim “package”
 - Implementar com visibilidade “package” reduziria a visibilidade do método em relação à definida na interface

```
public class IntegerSetClass implements IntegerSet {  
    int size() {...} //ERRADO: é “package”, devia ser public!  
}
```


Erros típicos com interfaces

54

- 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
 - Mas numa classe, a visibilidade por omissão de um método não é public mas sim “package”
 - Implementar com visibilidade “package” reduziria a visibilidade do método em relação à definida na interface

```
public class IntegerSetClass implements IntegerSet {  
    public int size() {...} //Assim já está bem: public  
}
```

Erros típicos com interfaces

55

○ 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(); //ERRO de Compilação!
```

○ 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
```

56

Curta introdução aos packages



O que é um package?

57

- É um espaço de nomes que organiza um conjunto de classes e interfaces relacionadas
- Semelhante às pastas nos computadores
- Forma de organizar classes e interfaces
 - Uma aplicação pode ser composta por milhares de classes
 - aplicações grandes podem ter dezenas de packages
- A plataforma do Java tem uma vasta biblioteca com milhares de classes
 - <http://download.oracle.com/javase/8/docs/api/index.html>

packages: declaração

58

- Para organizar as classes numa package `poo`, declaramos essa localização usando a palavra reservada **`package`**, que é sempre a primeira instrução dum ficheiro Java.

```
package poo;
```

```
public class Person {  
    // etc  
}
```

- Se a classe **`Person`** não estiver contida numa pasta **`poo`**, o compilador assinala um erro.

Visibilidade package

59

- Os **package** podem determinar a visibilidade dos membros das classes que não têm um classificador de visibilidade explícito.

```
package poo;
```

```
public class Person {  
    private int value; //Ninguém me vê fora da classe  
    int restricted;    //Ninguém me vê fora do package  
    public int sharedByAll; //Todos me podem ver  
    // etc  
}
```

- A visibilidade **package** é a mais restrictiva a seguir a **private**. Os elementos com essa visibilidade apenas são vistos (e usados) dentro da **package**.

De volta ao StupidFriendsBook

-Como tirar partido das interfaces na resolução deste problema?

Esqueleto da classe Main

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

61

```
public class Main {  
    // Comandos do utilizador  
    private static final String QUIT = "Sair";  
    // ...  
  
    // Feedback dado pelo programa  
    private static final String ADEUS= "Adeus";  
    // ...  
  
    public static void main(String[] args) {  
        Main.commands();  
    }  
  
    // ...  
}
```


Esqueleto da classe Main

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

62

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

Interface StupidFriendsBook

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

63

```
public interface StupidFriendsBook {  
    // Constantes de perfis de personalidade  
    static final int INTELLIGENT=0, ANGEL=1, BANDIT=2, STUPID=3;  
  
    void reset();  
    boolean hasFriend(String name);  
    void addFriend(String name);  
    void removeFriend(String name);  
    boolean hasAction(String name, String description);  
    void addAction(String name, String description);  
    void vote(String name, String description,  
              boolean goodForPerson, boolean goodForOthers);  
    // continua...
```

Interface StupidFriendsBook

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

64

```
// Continuação
int numberOfFriends();
String mostBoringFriend();
void initialize(int kind);
boolean hasNext();
Person next();
}
```

Esqueleto da classe

StupidFriendsBookClass

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

65

```
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 hasFriend(String name) {...}
    public void addFriend(String name) {...}
    public void removeFriend(String name) {...}
    public boolean hasAction(String name, String description){..}
    // continua...
```

Esqueleto da classe StupidFriendsBookClass

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

66

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

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

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

Interface Person

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

67

```
public interface Person {  
  
    boolean hasAction(String description);  
    void addAction(String description);  
    void vote(String description, boolean goodForPerson,  
              boolean goodForOthers);  
    String getName();  
    int getPersonality();  
    double getBoredom();  
}
```

Esqueleto da classe PersonClass

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

68

```
public class PersonClass implements Person{
    private static final int DEFAULT = 20;
    private Action[] myActions;
    private int counter;
    private String name;

    public PersonClass(String name) {...}
    public boolean hasAction(String description) {...}
    public void addAction(String description) {...}
    public void vote(String description,
        boolean goodForPerson, boolean goodForOthers) {...}
    public String getName() {...}
    public int getPersonality() {...}
    public double getBoredom() {...}
    // ...
}
```

Interface Action

Documento de partida

Fase 1
Modelação

Fase 2
Construção e Validação

Entregáveis

69

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


Esqueleto da classe ActionClass

70

```
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 getOtherBenefit () {...}  
}
```

71

Iterador com filtro

Interface StupidFriendsBook

72

```
public interface StupidFriendsBook {  
    // Constantes de perfis de personalidade  
    static final int INTELLIGENT =0, ANGEL =1, BANDIT =2, STUPID =3;  
  
    ...  
  
    // Métodos para a iteração  
    void initialize(int kind);  
    boolean hasNext();  
    Person next();  
  
    ...  
}
```

Esqueleto da classe StupidFriendsBookClass

73

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

    public StupidFriendsBookClass() {
        friends = new Person[DEFAULT];
        counter = 0;
        current = 0;
        kind = -1;
    }
    ...
    // Métodos para a interação
    public void initialize(int kind) {...}
    public boolean hasNext() {...}
    public Person next() {...}
}
```

Métodos para iteração (simples)

74

```
public void initialize() {  
    current = 0;  
}  
  
public boolean hasNext() {  
    return current < counter;  
}  
  
public Person next() {  
    return friends[current++];  
}
```

Métodos para iteração (filtro)

75

```
public void initialize(int kind) {
    current = 0;
    this.kind = kind;
    while (current < counter &&
           friends[current].getPersonality() != kind)
        current++;
}

public boolean hasNext() {
    return current < counter;
}

public Person next() {
    Person res = friends[current++];
    while (current < counter &&
           friends[current].getPersonality() != kind)
        current++;
    return res;
}
```

Refatorização do código

76

```
public void initialize(int kind) {
    current = 0;
    this.kind = kind;
    searchNext();
}

public boolean hasNext() {
    return current < counter;
}

public Person next() {
    Person res = friends[current++];
    searchNext();
    return res;
}

private void searchNext() {
    while (current < counter &&
           friends[current].getPersonality() != kind)
        current++;
}
```

Métodos para iteração

77

```
// Método na classe Main que usa os métodos de iteração para  
// percorrer os amigos da rede social
```

```
private static void list(int kind, StupidFriendsBook fb) {  
    fb.initialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
current = -1
```

friends =	B	S	I	S	I	A	A	counter = 7
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	kind = -1

Métodos para iteração

78

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.current < fb.friends.length) {  
        System.out.println(fb.friends[fb.current]);  
        fb.searchNext();  
    }  
}
```

```
public void initialize(int kind) {  
    current = 0; this.kind = kind;  
    searchNext();  
}
```

current = -1

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7
kind = -1

Métodos para iteração

79

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.current < fb.friends.length) {  
        System.out.println(fb.friends[fb.current]);  
        fb.searchNext();  
    }  
}
```

```
public void initialize(int kind) {  
    current = 0; this.kind = kind;  
    searchNext();  
}
```

current = 0

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

80

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.current < fb.friends.length) {  
        System.out.println(fb.friends[fb.current]);  
        fb.searchNext();  
    }  
}
```

```
public void initialize(int kind) {  
    current = 0; this.kind = kind;  
    searchNext();  
}
```

current = 0

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

81

```
void list(int kind, StupidFriendsBook fb) {
```

```
    fb.inicialize(kind);
```

```
    while (f
```

```
        Syste
```

```
    }
```

```
}
```

```
public void initialize(int kind) {
```

```
    current = 0; this.kind = kind;
```

```
private void searchNext() {
```

```
    → while (current < counter &&  
        friends[current].getPersonality() != kind)  
        current++;
```

```
}
```

current = 0

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

82

```
void list(int kind, StupidFriendsBook fb) {
```

```
    fb.inicialize(kind);
```

```
    while (f
```

```
        Syste
```

```
    }
```

```
}
```


```
public void initialize(int kind) {
```

```
    current = 0; this.kind = kind;
```

```
private void searchNext() {
```

```
    while (current < counter &&
```

```
        friends[current].getPersonality() != kind)
```

```
         current++;
```

```
}
```

current = 1

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

83

```
void list(int kind, StupidFriendsBook fb) {
```

```
    fb.inicialize(kind);
```

```
    while (f
```

```
        Syste
```

```
    }
```

```
}
```

```
public void initialize(int kind) {
```

```
    current = 0; this.kind = kind;
```

```
private void searchNext() {
```

```
    → while (current < counter &&  
        friends[current].getPersonality() != kind)  
        current++;
```

```
}
```

current = 1

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

84

```
void list(int kind, StupidFriendsBook fb) {
```

```
    fb.inicialize(kind);
```

```
    while (f
```

```
        Syste
```

```
    }
```

```
}
```

```
public void initialize(int kind) {
```

```
    current = 0; this.kind = kind;
```

```
private void searchNext() {
```

```
    while (current < counter &&
```

```
        friends[current].getPersonality() != kind)
```

```
        current++;
```

```
}
```

current = 2

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

85

```
void list(int kind, StupidFriendsBook fb) {
```

```
    fb.inicialize(kind);
```

```
    while (f
```

```
        System
```

```
    }
```

```
}
```

```
public void initialize(int kind) {
```

```
    current = 0; this.kind = kind;
```

```
private void searchNext() {
```

```
    → while (current < counter &&  
        friends[current].getPersonality() != kind)  
        current++;
```

```
}
```

current = 2

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

86

```
void list(int kind, StupidFriendsBook fb) {
```

```
    fb.inicialize(kind);
```

```
    while (f
```

```
        System
```

```
    }
```

```
}
```

```
public void initialize(int kind) {
```

```
    current = 0; this.kind = kind;
```

```
private void searchNext() {
```

```
    while (current < counter &&
```

```
        friends[current].getPersonality() != kind)
```

```
        current++;
```

```
    }
```



current = 2

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7


kind = I

Métodos para iteração

87

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.current < fb.friends.length) {  
        System.out.println(fb.friends[fb.current]);  
        fb.searchNext();  
    }  
}
```

```
public void initialize(int kind) {  
    current = 0; this.kind = kind;  
    searchNext();  
}
```



current = 2

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

88

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next());  
    }  
}
```

```
public boolean hasNext() {  
    return current < counter;  
}
```

current = 2

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

89

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}  
  
public Person next() {  
    Person res = friends[current++];  
    searchNext();  
    return res;  
}
```

current = 2

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

90

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {  
    Person res = friends[current++];  
    searchNext();  
    return res;  
}
```

current = 3

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

91

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {  
    Person res = friends[current++];  
    searchNext();  
    return res;  
}
```

current = 3

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

92

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {
```

```
    Person p = friends[current];  
    searchNext();  
    return p;  
}
```

```
private void searchNext() {  
    while (current < counter &&  
           friends[current].getPersonality() != kind)  
        current++;  
}
```

current = 3

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

93

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {
```

```
    Person p = friends[current];  
    searchNext();  
    return p;  
}
```

```
private void searchNext() {  
    while (current < counter &&  
           friends[current].getPersonality() != kind)  
        current++;  
}
```

current = 4

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

94

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {
```

```
    Person p = friends[current];  
    searchNext();  
    return p;  
}
```

```
private void searchNext() {  
    while (current < counter &&  
           friends[current].getPersonality() != kind)  
        current++;  
}
```

current = 4

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

95

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
System.out.println(fb.next().getName());
```

```
public Person next() {
```

```
    Person p = friends[current++];
```

```
    private void searchNext() {
```

```
        while (current < counter &&  
            friends[current].getPersonality() != kind)  
            current++;
```



current = 4

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

96

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {  
    Person res = friends[current++];  
    searchNext();  
    return res;  
}
```

current = 4

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

97

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next());  
    }  
}
```

```
public boolean hasNext() {  
    return current < counter;  
}
```

current = 4

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

98

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {  
    Person res = friends[current++];  
    searchNext();  
    return res;  
}
```

current = 5

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

99

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {  
    Person res = friends[current++];  
    searchNext();  
    return res;  
}
```

current = 5

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

100

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {
```

```
    Person p = friends[current];  
    searchNext();  
    return p;  
}
```

```
private void searchNext() {  
    while (current < counter &&  
           friends[current].getPersonality() != kind)  
        current++;  
}
```

current = 5

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

101

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {
```

```
    Person p = friends[current];  
    searchNext();  
    return p;  
}
```

```
private void searchNext() {  
    while (current < counter &&  
           friends[current].getPersonality() != kind)  
        current++;  
}
```

current = 6

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

102

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {
```

```
    Person p = friends[current];  
    searchNext();  
    return p;  
}
```

```
private void searchNext() {  
    while (current < counter &&  
           friends[current].getPersonality() != kind)  
        current++;  
}
```

current = 6

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

103

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {
```

```
    Person p = friends[current];  
    searchNext();  
    return p;  
}
```

```
private void searchNext() {  
    while (current < counter &&  
           friends[current].getPersonality() != kind)  
        current++;  
}
```

current = 7

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

104

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {
```

```
    Person p = friends[current];  
    searchNext();  
    return p;  
}
```

```
private void searchNext() {  
    while (current < counter &&  
           friends[current].getPersonality() != kind)  
        current++;  
}
```

current = 7

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

105

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

```
public Person next() {
```

```
    Person p = friends[current];  
    searchNext();  
    return p;  
}
```

```
private void searchNext() {  
    while (current < counter &&  
           friends[current].getPersonality() != kind)  
        current++;  
}
```

current = 7

friends =


B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

106

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}  
  
public Person next() {  
    Person res = friends[current++];  
    searchNext();  
     return res;  
}
```

current = 7

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

107

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next());  
    }  
}
```

```
public boolean hasNext() {  
    return current < counter;  
}
```

current = 7

friends =

B	S	I	S	I	A	A
[0]	[1]	[2]	[3]	[4]	[5]	[6]

counter = 7

kind = I

Métodos para iteração

108

```
void list(int kind, StupidFriendsBook fb) {  
    fb.inicialize(kind);  
    while (fb.hasNext()) {  
        System.out.println(fb.next().getName());  
    }  
}
```

