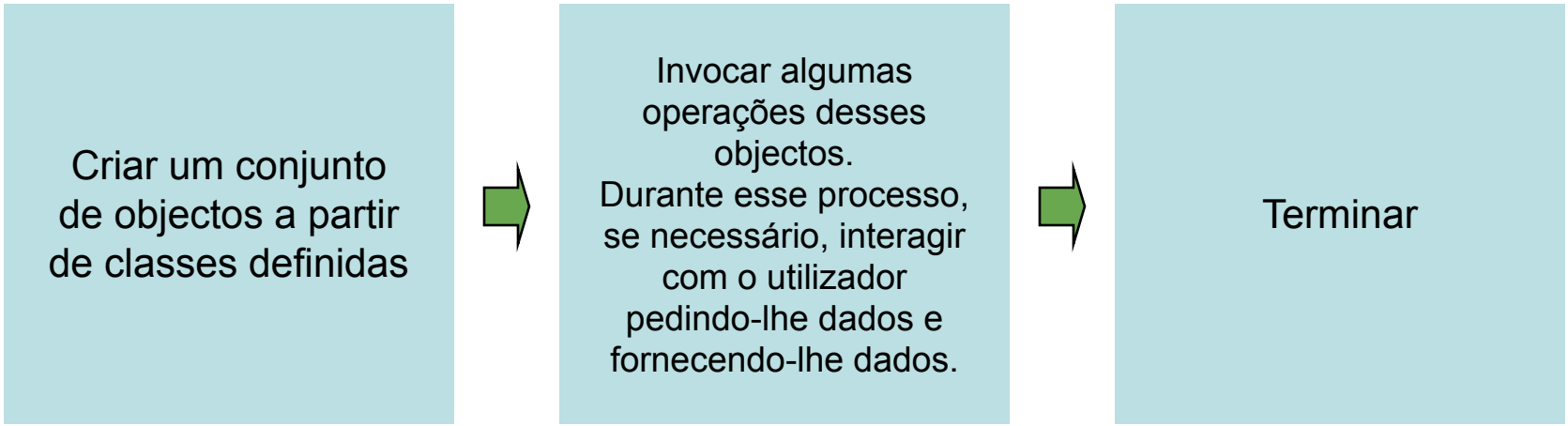
Programa Principal
Leitura e Escrita de dados

Mestrado Integrado em Engenharia Informática FCT UNL
http://ctp.di.fct.unl.pt/miei/ip/

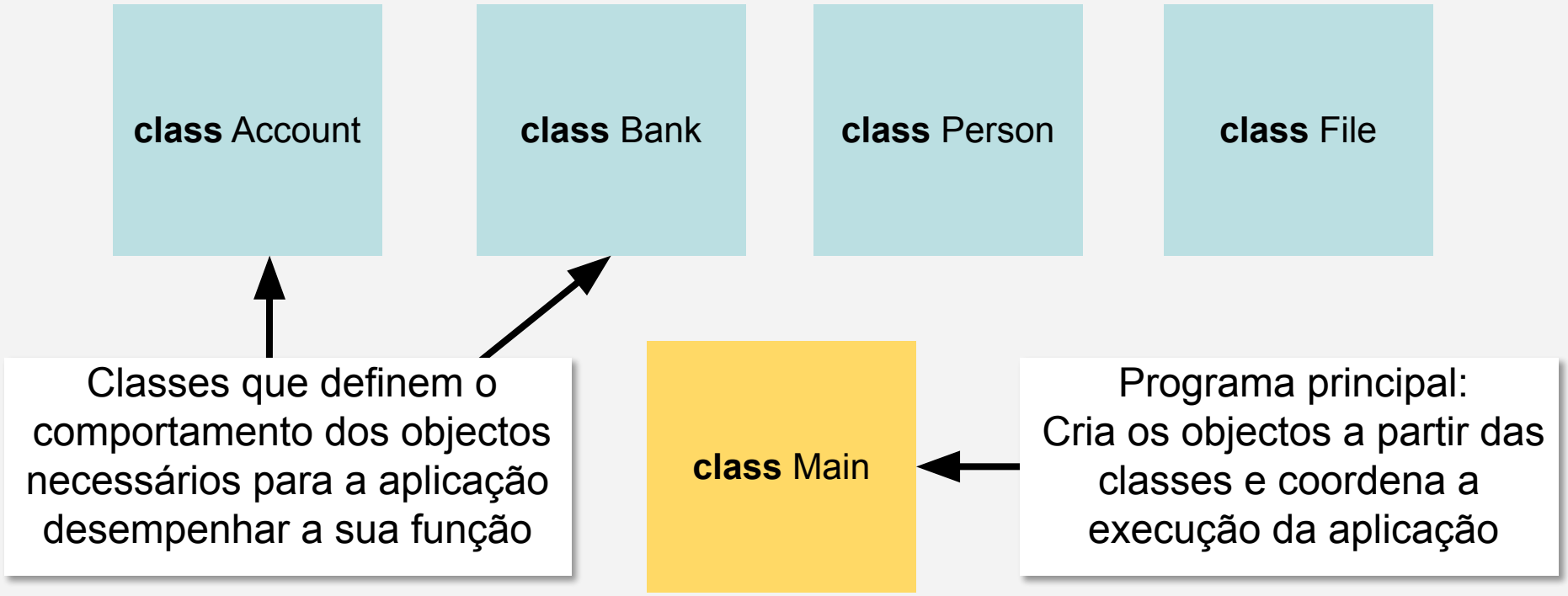
Corpo Docente 2020/2021
António Ravara, Artur Miguel Dias, Bernardo Toninho,
Ema Vieira, Inês Fernandes, Margarida Mamede,
Miguel Monteiro, Rui Nóbrega

Programa Principal Simples

- Os programas mais simples têm sempre a seguinte estrutura de tarefas em sequência



Estrutura de uma Aplicação Simples



Os componentes de qualquer aplicação são sempre **um conjunto de classes** e um **programa principal**, que coordena a execução do sistema de software

Programa Principal Simples

- Em Java, o programa principal de uma aplicação define-se numa classe especial (a que chamamos `Main`), definida à parte das outras classes
- Na classe `Main` define-se um método especial, *estático*, chamado `main`, com o programa principal

Modificador de acesso `static`
Num método indica que este é da classe e não dos objectos.

```
public class Main {
    public static void main(String[] args) {

    }
}
```

Programa principal
Quando a aplicação iniciar a sua execução, são as instruções aqui colocadas que irão ser executadas

Programa Principal Simples

- Exemplo (Lâmpadas)
  - O programa principal de uma aplicação simples que cria duas lâmpadas e realiza algumas operações

```
public class Main {
    public static void main(String[] args) {
        Lamp l1 = new Lamp();
        Lamp l2 = new Lamp();
        l1.turnOn();
        l2.turnOn();
        l1.turnOff();
    }
}
```

Programa Principal Simples

- Exemplo (Conta)
  - O programa principal de uma aplicação simples que cria uma conta bancária segura e realiza algumas operações

```
public class Main {
    public static void main(String[] args) {
        SafeBankAccount b = new SafeBankAccount(1000);
        b.deposit(20);
        b.applyInterest();
    }
}
```

Programa Principal Simples

- Exemplo (Contas)
  - O programa principal de uma aplicação simples que cria duas contas e realiza uma transferência entre elas

```
public class Main {
    public static void main(String[] args) {
        SafeBankAccount b1 = new SafeBankAccount(1000);
        SafeBankAccount b2 = new SafeBankAccount(2000);
        int someAmount = 20;
        b1.withdraw(someAmount);
        b2.deposit(someAmount);
    }
}
```

Note que os programas principais de exemplo apresentados até aqui não produzem resultados visíveis para o utilizador.

Output de texto

- Exemplo (Output)

  A operação `System.out.println` pode ser usada no programa principal para escrever mensagens ao utilizador

```
public class Main {
    public static void main(String[] args) {
        String name = "Luis";
        int age = 20;
        System.out.println(name);
        System.out.println("The age is "+age+", not 42");
        System.out.println("Pi="+Math.PI);
    }
}
```

Programa Principal Simples

- Exemplo (Contas e Output)
  - O programa principal de uma aplicação simples que cria duas contas e realiza uma transferência entre elas

```
public class Main {
    public static void main(String[] args) {
        SafeBankAccount b1 = new SafeBankAccount(1000);
        SafeBankAccount b2 = new SafeBankAccount(2000);
        int someAmount = 20;
        b1.withdraw(someAmount);
        b2.deposit(someAmount);
        System.out.println( b1.getBalance() );
        System.out.println( b2.getBalance() );
    }
}
```

Princípio arquitectural

- Em qualquer aplicação que se programe é muito importante separar claramente
  - As partes responsáveis pela interacção com o utilizador
  - As partes responsáveis pela funcionalidade dos vários objectos (definidas em várias classes)

- Assim, o uso de operações de output (ou input, como veremos mais tarde) é apenas permitido no interior da classe `Main` ou nas classes responsáveis pela interacção.
  Esta regra é **muito importante**.
- Por exemplo, o uso de `System.out.println` numa classe de aplicação, como `SafeBankAccount` ou `Calculator` será **sempre proibido** nas disciplinas de programação do MIEI.

Conta Bancária Segura

- **Objectivo**
  - Simular uma conta bancária segura.
- **Descrição**
  - Uma conta bancária é um “depósito” de dinheiro (valor inteiro em cêntimos). O saldo pode ser positivo (credor ou nulo) ou negativo (devedor), e é sempre um valor inteiro em cêntimos.
- **Funcionalidades**
  - Numa conta pode-se depositar e levantar dinheiro. Deve ser sempre possível consultar o saldo da conta e verificar se a conta tem um saldo devedor.
  - O levantamento é seguro, pois só pode ser efectuado caso o valor a levantar seja inferior ou igual ao valor do saldo. Qualquer tentativa de levantamento de um valor sem provisão na conta leva à aplicação de uma multa de 2 Euros.
  - Usualmente, o banco credita um certo juro nas contas dos seus clientes, numa base periódica (por exemplo, uma vez por ano). O valor do juro é calculado por aplicação de uma taxa ao valor do saldo, ou seja, a taxa de juro é determinada com base no valor do saldo, através de um sistema de escalões (quanto maior o saldo, maior a taxa).

Conta Bancária Segura

- Deve ser sempre possível calcular o valor de juro anual a aplicar ao saldo da conta. Existem três taxas possíveis:

Valor do Saldo	Taxa
≤ 2.000 €	1%
]2.000 €,10.000 €]	2%
> 10.000 €	3%

- Deve ser sempre possível creditar os juros no saldo da conta.
- Se não indicarmos nada, a conta é criada com saldo zero. Em alternativa, podemos indicar um valor inicial para o saldo.

- **Interacção com o utilizador**
  - Após criar uma conta bancária segura, pode invocar as operações da conta.

Conta Bancária Segura

Interface de SafeBankAccount:

`void` `deposit(`**`int`** `amount)`
Depositar a importância `amount` na conta
`Pre: amount > 0`

`void` `withdraw(`**`int`** `amount)`
Levantar a importância `amount` na conta ou aplicar a multa (**`FINE`**)
`Pre: amount > 0`

`int` `getBalance()`
Consultar o saldo da conta

`boolean` `isInRedZone()`
Indica se a conta está devedora

`int` `computeInterest()`
Calcular qual o valor do juro anual a aplicar
`Pre:` `!isInRedZone()`

`void` `applyInterest()`
Creditar o juro anual ao saldo da conta
`Pre:` `!isInRedZone()`

Nova interacção com o utilizador

Desenvolver um programa que:

- Cria uma conta bancária:
  - Pede ao utilizador o saldo inicial da conta a criar.
  - Cria uma conta bancária com o saldo indicado, **se e só se** este for válido (**>=0**);
- Efetua um levantamento na conta:
  - Pede o valor a levantar na conta e efectua o levantamento se este valor for válido;
  - Informa caso não seja possível efectuar o levantamento.
    - O levantamento só será efectuado com sucesso, se o saldo da conta for igual ou superior ao valor a levantar;
    - Note que, caso não seja efectuado o levantamento será sempre aplicada uma multa (recorde a definição de conta segura).
- Lista o saldo corrente da conta.

Construção da classe `Main`

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {
      //Cria uma conta e tenta fazer levantamento.
      SafeBankAccount account; //Conta Bancária
      int balance; // Saldo da conta
      int amount; // Valor a levantar

      Scanner in = new Scanner(System.in);
      System.out.print("Saldo inicial: ");
      balance = in.nextInt();
      in.nextLine();
      //Criar conta bancaria com saldo inicial = balance
      if (balance >= 0) { //Verificar a pre-condicao de SafeBankAccount
         account = new SafeBankAccount(balance);
         //tentar levantar dinheiro
         System.out.print("Valor a levantar em centimos:");
         amount = in.nextInt();
         in.nextLine();
         ...
      }
      ...
      in.close();
} }
```

Temos que indicar onde está definida a classe `Scanner`

Temos que ler tudo o que está para além do inteiro (nomeadamente a mudança de linha)

Devemos encerrar o `Scanner in`, libertando assim os seus recursos, quando terminamos as leituras de dados.

Construção da classe `Main`

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
      ...
      //Criar conta bancaria com saldo inicial = balance
      //Verificar a pré-condição de SafeBankAccount
      if (balance >= 0) {
         ...// slide anterior
         if (amount > 0){ //Verificar a pre-condicao de withdraw
            if (account.getBalance()< amount)//O saldo nao e suficiente
              System.out.println("Levantamento nao efectuado.");
           // Chama-se withdraw mesmo quando o saldo nao e suficiente
           // para a multa ser aplicada
           account.withdraw(amount);
           System.out.println("Saldo da conta=" +
                                account.getBalance() + " centimos");
         }
         else System.out.println("Levantamento negativo: nao efetuado.");
      }
      else System.out.println("Saldo inicial negativo: Conta nao criada.");
      in.close();
    }
}
```

Nova interacção com o utilizador (transferência bancária)

- Desenvolver um programa que:
  - Pede ao utilizador o saldo inicial de duas contas bancárias.
    - Se um dos saldos iniciais não for positivo, cria a conta com saldo igual a 0;
  - Pede um valor a transferir e tenta executar uma transferência bancária de uma conta para a outra.
    - A transferência só será efectuada com sucesso, se o saldo da conta de origem for igual ou superior ao valor a transferir;
    - Note que, caso não seja efectuada a transferência NÃO será aplicada uma multa.
  - Lista o saldo corrente das contas.

Primeira tentativa de construção da classe `Main`

Cria duas contas, com os saldos iniciais dados pelo utilizador...

```
import java.util.Scanner;
public class Main {
	public static void main(String[] args) {
		Scanner in = new Scanner(System.in);
		SafeBankAccount accountOrigin =new SafeBankAccount(); //Comeca a 0
		System.out.println("Conta Origem");
		System.out.println("Saldo inicial em centimos:");
		int initialOrigin = in.nextInt();   // Lê saldo inicial
		in.nextLine();
		if (initialOrigin > 0)              // Se for positivo
			accountOrigin.deposit(initialOrigin);// Deposita o saldo inicial
		SafeBankAccount accountDestiny =new SafeBankAccount(); //Comeca a 0
		System.out.println("Conta Destino");
		System.out.println("Saldo inicial em centimos:");
		int initialDestiny = in.nextInt();  // Le saldo inicial
		in.nextLine();
		if (initialDestiny > 0)             // Se for positivo
			accountDestiny.deposit(initialDestiny);// Deposita o saldo inicial
		System.out.println("Valor a transferir em centimos:");
		int value = in.nextInt();
		in.nextLine();
		if (accountOrigin.getBalance() >= value && value >= 0){
			accountOrigin.withdraw(value);
			accountDestiny.deposit(value);
			System.out.println("Transferencia efectuada com sucesso");
		} else
			System.out.println("Transferencia nao efectuada");
		System.out.print("Saldo conta origem: ");
		System.out.println(accountOrigin.getBalance()+ " Centimos");
		System.out.print("Saldo conta destino: ");
		System.out.println(accountDestiny.getBalance()+ " Centimos");
		in.close();
	}
}
```

… e depois lê o valor a transferir e transfere-o de `accountOrigin` para `accountDestiny`

Funciona, mas não é uma boa solução...

```
import java.util.Scanner;
public class Main {
      public static void main(String[] args) {
            Scanner in = new Scanner(System.in);
            SafeBankAccount accountOrigin =new SafeBankAccount(); //Comeca a 0
            System.out.println("Conta Origem");
            System.out.println("Saldo inicial em centimos:");
            int initialOrigin = in.nextInt();   // Le saldo inicial
            in.nextLine();
            if (initialOrigin > 0)              // Se for positivo
                  accountOrigin.deposit(initialOrigin);// Deposita o saldo inicial
            SafeBankAccount accountDestiny =new SafeBankAccount(); //Comeca a 0
            System.out.println("Conta Destino");
            System.out.println("Saldo inicial em centimos:");
            int initialDestiny = in.nextInt();  // Le saldo inicial
            in.nextLine();
            if (initialDestiny > 0)             // Se for positivo
                  accountDestiny.deposit(initialDestiny);// Deposita o saldo inicial
            System.out.println("Valor a transferir em centimos:");
            int value = in.nextInt();
            in.nextLine();
            if (accountOrigin.getBalance() >= value && value >= 0){
                  accountOrigin.withdraw(value);
                  accountDestiny.deposit(value);
                  System.out.println("Transferencia efectuada com sucesso");
            } else
                  System.out.println("Transferencia nao efectuada");
            System.out.print("Saldo conta origem: ");
            System.out.println(accountOrigin.getBalance()+ " Centimos");
            System.out.print("Saldo conta destino: ");
            System.out.println(accountDestiny.getBalance()+ " Centimos");
            in.close();
      }
}
```

Cria duas contas, com os saldos iniciais dados pelo utilizador...

Este método está a ficar complicado. Já viu bem o tamanho de fonte que tivemos de usar para ele caber num único slide? :-(
O problema é que este método faz demasiadas tarefas...

... e depois lê o valor a transferir e transfere-o de `accountOrigin` para `accountDestiny`

Vamos usar uma lupa...

Cria duas contas, com os saldos iniciais dados pelo utilizador...

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        SafeBankAccount accountOrigin = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta Origem");
        System.out.println("Saldo inicial em centimos:");
        int initialOrigin = in.nextInt();   // Le saldo inicial
        in.nextLine();
        if (initialOrigin > 0)              // Se for positivo
            accountOrigin.deposit(initialOrigin);// Deposita o saldo inicial
        SafeBankAccount accountDestiny = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta Destino");
        System.out.println("Saldo inicial em centimos:");
        int initialDestiny = in.nextInt();  // Le saldo inicial
        in.nextLine();
        if (initialDestiny > 0)             // Se for positivo
            accountDestiny.deposit(initialDestiny);// Deposita o saldo inicial
        ...
    }
```

... e depois lê o valor a transferir e transfere-o de `accountOrigin` para `accountDestiny`

Temos bastante código repetido

O código de criação das duas contas é bastante semelhante.

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        SafeBankAccount accountOrigin = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta Origem");
        System.out.println("Saldo inicial em centimos:");
        int initialOrigin = in.nextInt();   // Le saldo inicial
        in.nextLine();
        if (initialOrigin > 0)              // Se for positivo
            accountOrigin.deposit(initialOrigin);// Deposita o saldo inicial
        SafeBankAccount accountDestiny = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta Destino");
        System.out.println("Saldo inicial em centimos:");
        int initialDestiny = in.nextInt();  // Le saldo inicial
        in.nextLine();
        if (initialDestiny > 0)             // Se for positivo
            accountDestiny.deposit(initialDestiny);//Deposita o saldo inicial
        ...
    }
```

Podemos e devemos evitar código desnecessariamente repetido.

Código repetido é mesmo necessário?

O código de criação das duas contas é bastante semelhante.

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        SafeBankAccount accountOrigin = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta Origem");
        System.out.println("Saldo inicial em centimos:");
        int initialOrigin = in.nextInt();   // Le saldo inicial
        in.nextLine();
        if (initialOrigin > 0)              // Se for positivo
            accountOrigin.deposit(initialOrigin);// Deposita o saldo inicial
        SafeBankAccount accountDestiny = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta Destino");
        System.out.println("Saldo inicial em centimos:");
        int initialDestiny = in.nextInt();  // Le saldo inicial
        in.nextLine();
        if (initialDestiny > 0)             // Se for positivo
            accountDestiny.deposit(initialDestiny);//Deposita o saldo inicial
        ...
    }
```

E se programarmos o essencial dos dois casos **num método separado** e chamarmos esse método duas vezes? Alguns `detalhes` terão de ser tratados, claro.

Podemos criar um **método auxiliar**

Podemos declarar um método `createAccount(...)` que construa a conta

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        SafeBankAccount origin = createAccount(in, "origem");
        SafeBankAccount destiny = createAccount(in, "destino");
        ...
        in.close();
    }

    private static SafeBankAccount createAccount(Scanner in,String name){
        SafeBankAccount account = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta " + name);
        System.out.println("Saldo inicial em centimos:");
        int initial = in.nextInt(); // Le saldo inicial
        in.nextLine();
        if (initial > 0)              // Se for positivo
            account.deposit(initial);// Deposita o saldo inicial
        return account;
    }
}
```

Este método auxiliar deve ser **privado** (e estático). Necessitamos de um parâmetro para o Scanner (***in***) e outro com o nome da conta (**name**).

Invocação do método auxiliar

Invocamos `createAccount(...)` para construir duas novas contas.

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        SafeBankAccount origin = createAccount(in, "origem");
        SafeBankAccount destiny = createAccount(in, "destino");
        ...
        in.close();
    }

    private static SafeBankAccount createAccount(Scanner in,String name){
        SafeBankAccount account = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta " + name);
        System.out.println("Saldo inicial em centimos:");
        int initial = in.nextInt(); // Le saldo inicial
        in.nextLine();
        if (initial > 0)              // Se for positivo
            account.deposit(initial);// Deposita o saldo inicial
        return account;
    }
}
```

Em vez de um método "gigante" `main`, podemos definir métodos auxiliares privados, como ***`createAccount(...)`***

Continuando a olhar o `main` à lupa...

Pede o valor a transferir, efectua a transferência e dá feedback ao utilizador.

```
public class Main {
    public static void main(String[] args) {
        …
        System.out.println("Valor a transferir em centimos:");
        int amount = in.nextInt();
        in.nextLine();
        if (accountOrigin.getBalance() >= amount && amount >= 0) {
            accountOrigin.withdraw(amount);
            accountDestiny.deposit(amount);
            System.out.println("Transferencia efectuada com sucesso");
        } else {
            System.out.println("Transferencia nao efectuada");
        }
        System.out.print("Saldo conta origem: ");
        System.out.println(accountOrigin.getBalance()+ " Centimos");
        System.out.print("Saldo conta destino: ");
        System.out.println(accountDestiny.getBalance()+ " Centimos");
        in.close();
    }
}
```

Continuamos a ver várias tarefas misturadas e algum código repetido

Várias oportunidades de melhoria

Pede o valor a transferir, efectua a transferência e dá feedback ao utilizador.

```
public class Main {
    public static void main(String[] args) {
        …
        System.out.println("Valor a transferir em centimos:");
        int amount = in.nextInt();
        in.nextLine();
        if (accountOrigin.getBalance() >= amount && amount >= 0) {
            accountOrigin.withdraw(amount);
            accountDestiny.deposit(amount);
            System.out.println("Transferencia efectuada com sucesso");
        } else {
            System.out.println("Transferencia nao efectuada");
        }
        System.out.print("Saldo conta origem: ");
        System.out.println(accountOrigin.getBalance()+ " centimos");
        System.out.print("Saldo conta destino: ");
        System.out.println(accountDestiny.getBalance()+ " centimos");
        in.close();
    }
}
```

Continuamos a ver várias tarefas misturadas e algum código repetido

A leitura de um valor merece um método

Pede o valor a transferir, efectua a transferência e dá feedback ao utilizador.

```
public class Main {
    public static void main(String[] args) {
        …
        System.out.println("Valor a transferir em centimos:");
        int amount = in.nextInt();
        in.nextLine();
        if (accountOrigin.getBalance() >= amount && amount >= 0) {
            accountOrigin.withdraw(amount);
            accountDestiny.deposit(amount);
            System.out.println("Transferencia efectuada com sucesso");
        } else {
            System.out.println("Transferencia nao efectuada");
        }
        System.out.print("Saldo conta origem: ");
        System.out.println(accountOrigin.getBalance()+ " Centimos");
        System.out.print("Saldo conta destino: ");
        System.out.println(accountDestiny.getBalance()+ " Centimos");
        in.close();
    }
```

Podemos ter um método privado para pedir o valor a transferir...

O método `getIntValue` lê um inteiro

Pede o valor a transferir, efectua a transferência e dá feedback ao utilizador.

```
public class Main {
    public static void main(String[] args) {
        …
        int amount = getIntValue(in,"Valor a transferir em centimos:");
        …
        in.close();
    }

    private static int getIntValue(Scanner in, String msg){
        System.out.println(msg);
        int value = in.nextInt();
        in.nextLine();
        return value;
    }
}
```

Podemos ter um método privado para pedir o valor a transferir...

O método `transferMoney` transfere o valor da origem para o destino

Pede o valor a transferir, **efectua a transferência** e dá feedback ao utilizador.

```
public class Main {
    public static void main(String[] args) {
        …
        int amount = getIntValue(in,"Valor a transferir em centimos:");
        transferMoney(accountOrigin, accountDestiny, amount);
        …
        in.close();
    }
    private static void transferMoney(SafeBankAccount accountOrigin,
                                      SafeBankAccount accountDestiny,
                                      int value) {
        if (accountOrigin.getBalance() >= value && value >= 0){
            accountOrigin.withdraw(value);
            accountDestiny.deposit(value);
            System.out.println("Transferencia efectuada com sucesso");
        } else
            System.out.println("Transferencia nao efectuada");
    }
}
```

Podemos ter um método privado para transferir o valor `value`...

O método `printAccount` escreve o estado de uma conta na consola

Pede o valor a transferir, efectua a transferência e dá feedback ao utilizador.

```
public class Main {
    public static void main(String[] args) {
        …
        int amount = getIntValue(in,"Valor a transferir em centimos:");
        transferMoney(accountOrigin, accountDestiny, amount);
        printAccount(accountOrigin, "origem");
        printAccount(accountDestiny, "destino");
        in.close();
    }

    private static void printAccount(SafeBankAccount account,
                                     String name) {
        System.out.print("Saldo conta " + name + ": ");
        System.out.println(account.getBalance()+ " centimos");
    }
}
```

Podemos ter um método privado para escrever os saldos finais.

Podemos ainda melhorar...

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        SafeBankAccount origin = createAccount(in, "origem");
        SafeBankAccount destiny = createAccount(in, "destino");
        ...
        in.close();
    }

    private static SafeBankAccount createAccount(Scanner in,String name){
        SafeBankAccount account = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta " + name);
        System.out.println("Saldo inicial em centimos:");
        int initial = in.nextInt(); // Le saldo inicial
        in.nextLine();
        if (initial > 0)              // Se for positivo
            account.deposit(initial);// Deposita o saldo inicial
        return account;
    }
}
```

Re-utilização de `getIntValue()`

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        SafeBankAccount origin = createAccount(in, "origem");
        SafeBankAccount destiny = createAccount(in, "destino");
        ...
        in.close();
    }
```

Dentro do `createAccount(...)` necessitamos de introduzir um `int`.

```
    private static SafeBankAccount createAccount(Scanner in,String name){
        SafeBankAccount account = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta " + name);
        System.out.println("Saldo inicial em centimos:");
        int initial = in.nextInt(); // Le saldo inicial      Padrão típico
        in.nextLine();
        if (initial > 0)                // Se for positivo
            account.deposit(initial);// Deposita o saldo inicial
        return account;
    }
}
```

Re-utilização de `getIntValue()`

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        SafeBankAccount origin = createAccount(in, "origem");
        SafeBankAccount destiny = createAccount(in, "destino");
        ...
        in.close();
    }

    private static SafeBankAccount createAccount(Scanner in,String name){
        SafeBankAccount account = new SafeBankAccount(); //Comeca a 0
        System.out.println("Conta " + name);
        int initial = getIntValue(in,"Saldo inicial em centimos:");
        if (initial > 0)                // Se for positivo
            account.deposit(initial);// Deposita o saldo inicial
        return account;
    }
}
```

Dentro do `createAccount(...)` necessitamos de introduzir um `int`.

A classe `Main` completa.

Cria duas contas bancárias, pede o valor a transferir, efectua a transferência e dá feedback ao utilizador. Cada tarefa no seu respectivo método auxiliar.

```
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        SafeBankAccount accountOrigin = createAccount(in, "origem");
        SafeBankAccount accountDestiny = createAccount(in, "destino");
        int amount = getIntValue(in,"Valor a transferir ...");
        transferMoney(accountOrigin, accountDestiny, amount);
        printAccount(accountOrigin, "origem");
        printAccount(accountDestiny, "destino");
        in.close();
    }
    // Metodos auxiliares:
    private static SafeBankAccount createAccount(...) {...}
    private static int getIntValue(...) {...}
    private static void transferMoney(...) {...}
    private static void printAccount(...) {...}
}
```