

Programa Principal

Leitura e Escrita de dados

**Material didáctico elaborado pelas diferentes equipas de
Introdução à Programação**

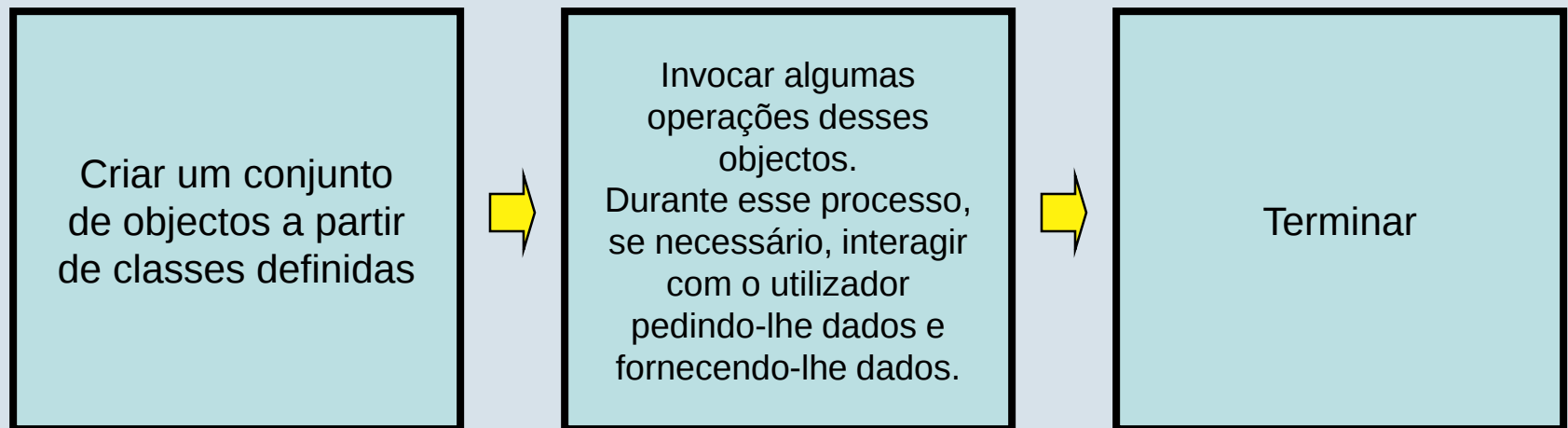
Luís Caires (Responsável), Armanda Rodrigues, António Ravara, Carla Ferreira, Fernanda Barbosa, Fernando Birra, Jácome Cunha, João Araújo, Miguel Goulão, Miguel Pessoa Monteiro, e Sofia Cavaco.

Mestrado Integrado em Engenharia Informática FCT UNL

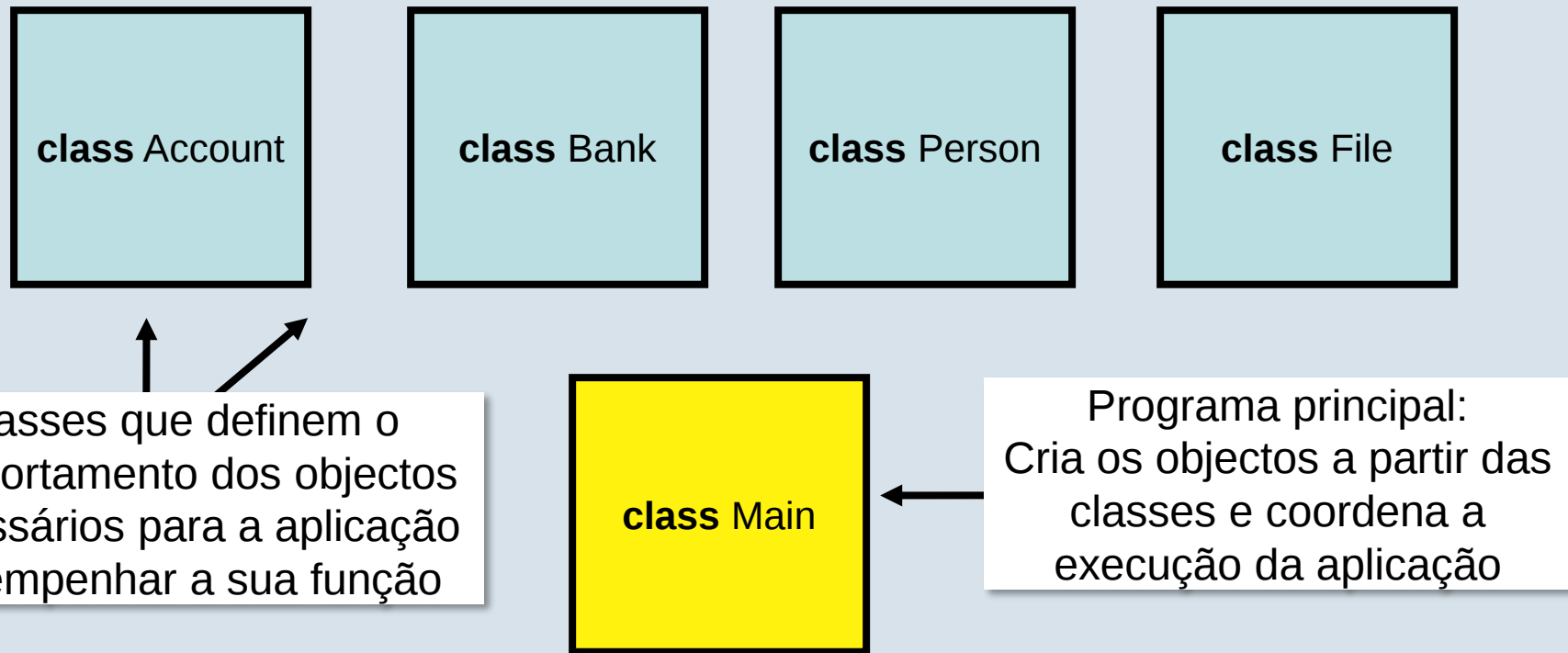
Programa Principal Simples

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

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 e input é apenas permitido no interior da classe `Main` ou nas classes responsáveis pela interacção.**
- Esta regra é mesmo muito importante.
- Por exemplo, o uso de `System.out.println` numa classe de aplicação, como `SafeBankAccount` ou `TwitterHappy` será **sempre proibido** nas disciplinas de programação do MIEI.

Programa Principal Simples

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

```
public class Main {
```

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

```
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.on();  
        l2.on();  
        l1.off();  
    }  
}
```

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 some_amount = 20 ;  
        b1.withdraw(some_amount) ;  
        b2.deposit(some_amount) ;  
    }  
}
```

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 some_amount = 20 ;  
        b1.withdraw(some_amount) ;  
        b2.deposit(some_amount) ;  
        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**.

Recorde a 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).

Recorde a 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
$\leq 2000 \text{ €}$	1%
$]2000 \text{ €, } 10.000 \text{ €}]$	2%
$> 10.000 \text{ €}$	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.

Recorde a 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 redZone()

Indica se a conta está devedora

int computeInterest()

Calcular qual o valor do juro anual a aplicar

void applyInterest()

Creditar o juro anual ao saldo da conta

Nova interacção com o utilizador

- Desenvolver um programa que:
 - Pede ao utilizador o saldo inicial duma conta bancária.
 - Pede o valor a levantar na conta. 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.

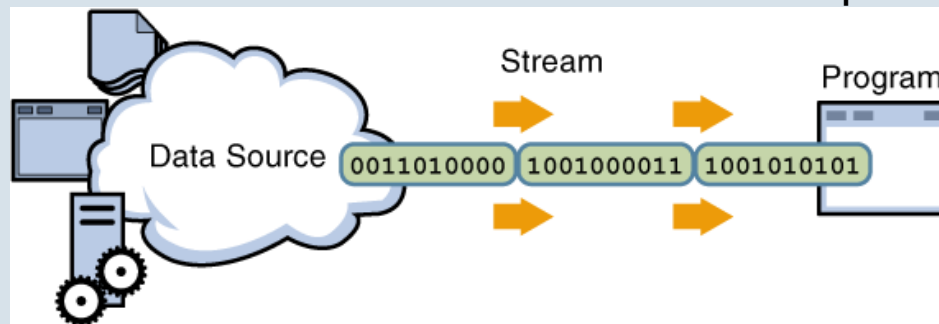
Leitura de dados

- `System.in` é o nome de um objecto pré-definido utilizado para fazer leitura de dados a partir do *standard input*
 - Por omissão, o *standard input* é o teclado
- No entanto, o `System.in` só lê um carácter de cada vez
- No Java, a classe `Scanner` foi criada para fazer a leitura a partir do teclado de uma forma simples
- O `Scanner` é inicializado estabelecendo uma ligação a uma *stream* de entrada de dados (neste caso, o `System.in`)
- Quando já não necessitamos de continuar a ler caracteres, devemos encerrar a ligação à *stream* de entrada de dados que foi estabelecida no construtor de `Scanner`.

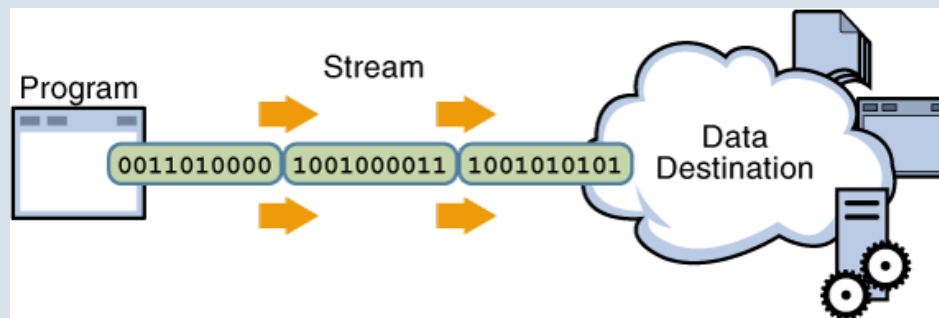
```
Scanner in = new Scanner(System.in);  
System.out.println("Quantidade: ");  
int value = in.nextInt();  
in.close();
```

System.in e System.out

- Os objectos `System.in` e `System.out` representam duas *streams* de entrada e saída de dados do programa, respectivamente
- Uma *stream* é um fluxo de dados de onde podemos ler ou onde podemos escrever informação.
 - `System.in` é a stream de entrada de dados usada por omissão no sistema



- `System.out` é a stream de saída de dados usada por omissão no sistema



Disciplina de uso do Scanner

- Antes de ler ou escrever dados numa *stream*, temos de lhe conseguir aceder
- Depois de ler ou escrever na *stream*, devemos “libertar” o acesso
- O objecto da classe `Scanner` não é uma *stream*, mas usa uma *stream* (no exemplo, `System.in`)
 - Indicamos a *stream* a que queremos aceder no construtor
 - Quando já não necessitamos do `Scanner`, devemos libertar a *stream* usando o método `close()` do `Scanner`, que encerra o acesso desse `Scanner` à *stream* indicada no construtor

```
Scanner in = new Scanner(System.in); // Scanner acede a System.in
System.out.println("Quantidade: ");
int value = in.nextInt(); //inteiro lido fica guardado em value
in.close(); // Ja nao necessitamos do Scanner, devemos liberta-lo
```

Operações da classe Scanner

- `int nextInt()` lê um inteiro
- `float nextFloat()` lê um real
- `double nextDouble()` lê um real
- `String next()` lê uma `String` sem espaços
- `String nextLine()` lê uma linha até ao fim
- `void close()` liberta os recursos do `Scanner`, desactivando-o

Estude a classe `Scanner` no pacote `java.util`

Main.java

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        //Cria uma conta e tenta fazer levantamento.
        Scanner in = new Scanner(System.in);
        SafeBankAccount account;
        int balance;
        int value; //podia ser so 1 variavel

        System.out.print("Saldo inicial: ");
        balance = in.nextInt();
        in.nextLine();

        //Criar conta bancaria com saldo inicial - balance
        account = new SafeBankAccount(balance);
        //tentar levantar dinheiro
        System.out.print("Valor a levantar em centimos:");
        value = in.nextInt();
        in.nextLine();
        in.close();
        // Slide seguinte...
    }
}
```

← Temos que indicar onde está definida a classe Scanner

← Temos que ler tudo o que está para além do inteiro

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

Main.java

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        // ... Slide anterior...
        if (account.getBalance() < value) // 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(value);
        System.out.println("Saldo da conta=" + account.getBalance()
                           + " centimos");
    }
}
```

Nova interacção com o utilizador

- Desenvolver um programa que:
 - Pedir ao utilizador, a informação referente a duas contas bancárias.
 - Para cada conta pedir o saldo inicial da conta
 - Pedir o valor a transferir da primeira conta dada para a segunda conta e efectuar a transferência. E informar se a transferência foi ou não efectuada com sucesso.
 - O processo de transferência consiste em levantar dinheiro na primeira conta e depositar esse mesmo dinheiro na segunda conta;
 - Uma transferência só será efectuada com sucesso, se for possível efectuar o levantamento na primeira conta.
 - Neste caso, se o saldo for inferior ao valor a transferir, não há lugar a débito de multa.
 - Listar a informação referente a cada conta:
 - Para cada conta deve escrever o saldo da conta.

Main.java

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int amount, value;
        SafeBankAccount accountOrigin, accountDestination;
        System.out.println("Conta Origem");
        System.out.println("Saldo inicial em centimos:");
        amount = in.nextInt();
        in.nextLine();
        accountOrigin = new SafeBankAccount(amount);
        System.out.println("Conta Destino");
        System.out.println("Saldo inicial em centimos:");
        amount = in.nextInt();
        in.nextLine();
        accountDestination = new SafeBankAccount(amount);
        ...
    }
}
```

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

Main.java

```
public class Main {
```

Pede o valor a transferir e efectua transferência, se conseguir

```
    public static void main(String[] args) {  
        ...  
        System.out.println("Valor a transferir em centimos:");  
        value = in.nextInt();  
        in.nextLine();  
        if (accountO.getBalance() >= value) {  
            accountO.withdraw(value);  
            accountD.deposit(value);  
            System.out.println("Transferencia efectuada com sucesso");  
        }  
        else {  
            System.out.println("Transferencia nao efectuada");  
            System.out.println("Conta Origem");  
            System.out.println(accountO.getBalance() + " Centimos");  
            System.out.println("Conta Destino");  
            System.out.println(accountD.getBalance() + " Centimos");  
        }  
        in.close();  
    }
```

Nova interacção com o utilizador (menú de opções)

- Desenvolver um programa que:
 - Pede ao utilizador o saldo inicial duma conta bancária.
 - Apresenta em forma de menú todas as operações possíveis de realizar sobre a conta, da seguinte forma:

- 1- Depositar
 - 2- Levantar
 - 3- Consultar saldo
 - 4- Consultar juro anual
 - 5- Creditar juro anual
 - 6- Sair
 - Pede ao utilizador a opção do menú que deseja realizar. Em caso de levantamento e depósito deve pedir também o valor a levantar ou depositar, respectivamente. Nos casos de consulta deve escrever na consola a informação respectiva.
 - Escrever na consola o saldo da conta.

Main.java

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int amount, option;
        SafeBankAccount account;
        System.out.println("Saldo inicial em centimos:");
        amount = in.nextInt();
        in.nextLine();
        account = new SafeBankAccount(amount);

        //Mostrar menu e ler opcao
        ...

        switch (option){
            //processar as varias opcoes
            ...
        }
        System.out.println("Saldo Conta:");
        System.out.println(account.getBalance()+ " Centimos");
        in.close();
    }
}
```

```

import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int amount, option;
        SafeBankAccount account;
        System.out.println("Saldo inicial em centimos:");
        amount = in.nextInt();
        in.nextLine();
        account = new SafeBankAccount(amount);

        //Mostrar menu e ler opcao
        System.out.println("1- Levantar");
        ...
        System.out.println("6- Sair");
        System.out.println("Escolha opcao:");
        option = in.nextInt();
        in.nextLine();

        switch (option){
            ...
        }
        System.out.println("Saldo Conta:");
        System.out.println(account.getBalance()+ " Centimos");
        in.close();
    }
}

```

Este método começa a ficar grande demais, e a combinar várias tarefas distintas, tais como inicialização de variáveis, apresentação de menus e tratamento de comandos do utilizador...

```

import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int amount, option;
        SafeBankAccount account;
        System.out.println("Saldo inicial em centimos:");
        amount = in.nextInt();
        in.nextLine();
        account = new SafeBankAccount(amount);

        //Mostrar menu e ler opcao
        System.out.println("1- Levantar");
        ...
        System.out.println("6- Sair");
        System.out.println("Escolha opcao:")
        option = in.nextInt();
        in.nextLine();

        switch (option) {
            ...
        }
        System.out.println("Saldo Conta:");
        System.out.println(account.getBalance()+ " Centimos");
        in.close();
    }
}

```

Podemos identificar algumas dessas tarefas autónomas. Por exemplo, mostrar o menu e ler uma opção poderiam ser colocadas em métodos auxiliares.

```

import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int amount, option;
        SafeBankAccount account;
        System.out.println("Saldo inicial em centimos:");
        amount = in.nextInt();
        in.nextLine();
        account = new SafeBankAccount(amount);

        //Mostrar menu e ler opcao
        option = readCommand(in);

        switch (option) {
            ...
        }
        System.out.println("Saldo Conta:");
        System.out.println(account.getBalance()+ " Centimos");
        in.close();
    }
}

```

Em vez de colocar o código todo num método “gigante”, devemos criar métodos auxiliares, de modo a facilitar a compreensão e futura evolução do código. Neste caso, `readCommand` mostra o menu e lê uma opção (`option`) que será usada no `switch`.

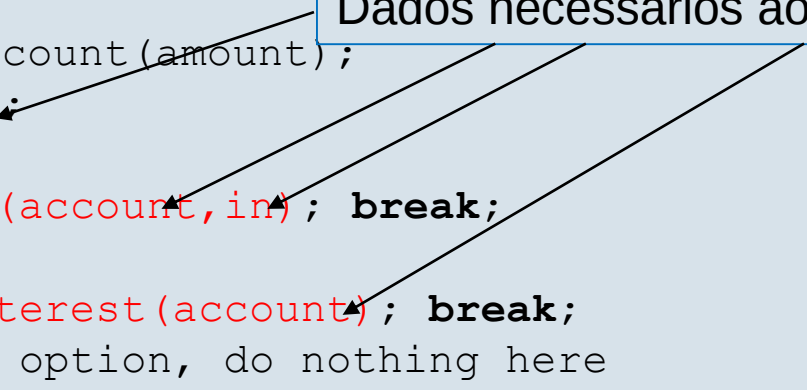
Main.java

```
import java.util.Scanner;  
public class Main {
```

Métodos privados da classe Main

```
    public static void main(String[] args) {  
        Scanner in = new Scanner(System.in);  
        int amount, option;  
        SafeBankAccount account;  
        System.out.println("Saldo inicial em centimos:");  
        amount = in.nextInt();  
        in.nextLine();  
        account = new SafeBankAccount(amount);  
        option = readCommand(in);  
        switch (option) {  
            case 1: processDeposit(account, in); break;  
            ...  
            case 5: processApplyInterest(account); break;  
            case 6: break; // exit option, do nothing here  
        }  
        System.out.println("Saldo Conta:");  
        System.out.println(account.getBalance()+ " Centimos");  
        in.close();  
    }  
}
```

Dados necessários aos métodos



The diagram consists of a blue-bordered box containing the text 'Dados necessários aos métodos'. Three black arrows originate from this box and point to specific method calls in the code: 'readCommand(in)' on line 15, 'processDeposit(account, in)' on line 18, and 'processApplyInterest(account)' on line 23.

Main.java

Métodos privados da classe Main

```
import java.util.Scanner;

public class Main {
    private static int readCommand(Scanner in) {
        int op;
        System.out.println("1- Levantar");
        ...
        System.out.println("6- Sair");
        System.out.println("Escolha opcao:")
        op = in.nextInt();
        in.nextLine();
        return op;
    }
    ...

    public static void main(String[] args) {
        ...
        option = readCommand(in);
        ...
    }
}
```

Main.java

```
import java.util.Scanner;  
public class Main {
```

Métodos privados da classe Main

```
    private static void processDeposit(SafeBankAccount b, Scanner in) {  
        int amount;  
        System.out.println("Montante em centimos(valor nao negativo):");  
        amount = in.nextInt();  
        in.nextLine();  
        if (amount >= 0){  
            b.deposit(amount);  
            System.out.println("Deposito efectuado com sucesso");  
        }  
        else  
            System.out.println("Montante deve ser um valor nao negativo");  
    }  
  
    public static void main(String[] args) {  
        ...  
        case 1: processDeposit(account, in); break;  
        ...  
    }  
}
```

Main.java

Métodos privados da classe Main

```
import java.util.Scanner;

public class Main {
    private static int readCommand(Scanner in) {
        ...
    }

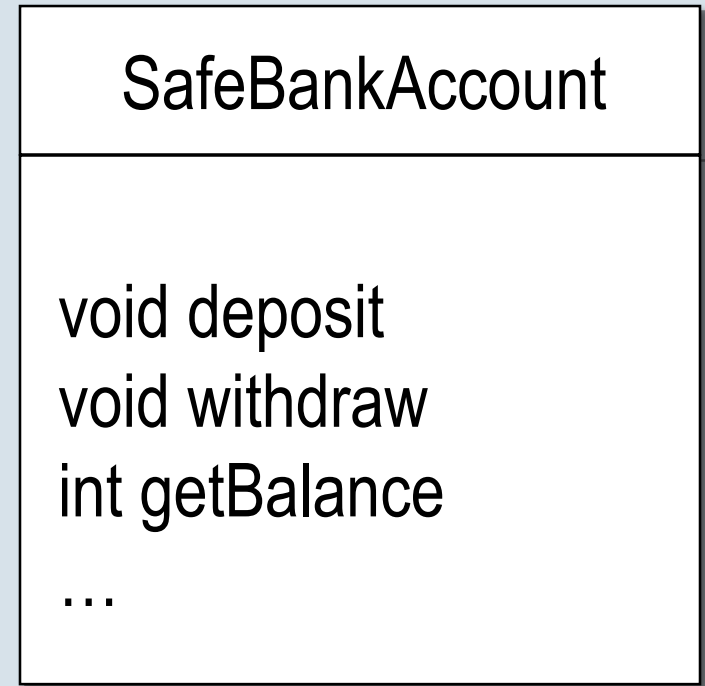
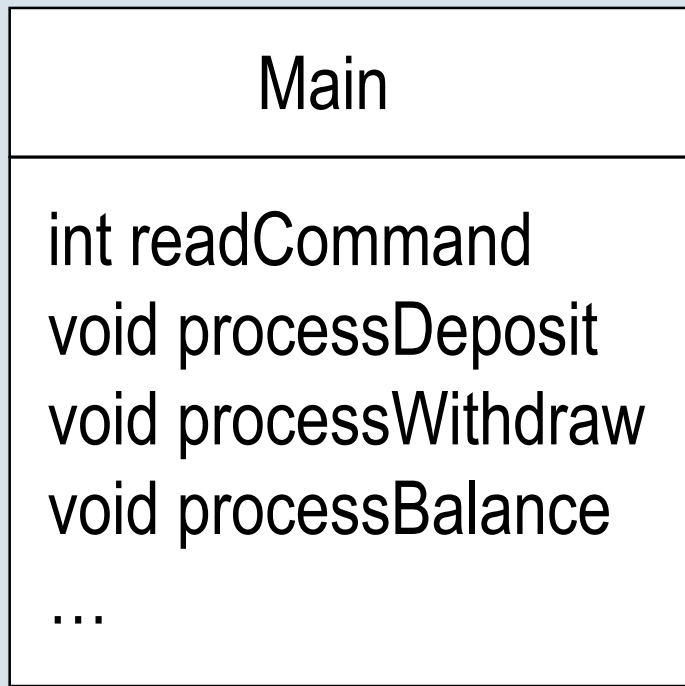
    private static void processDeposit(SafeBankAccount account,
                                       Scanner in) {
        ...
    }

    ...

    public static void main(String[] args) {
        ...
        ...
    }
}
```

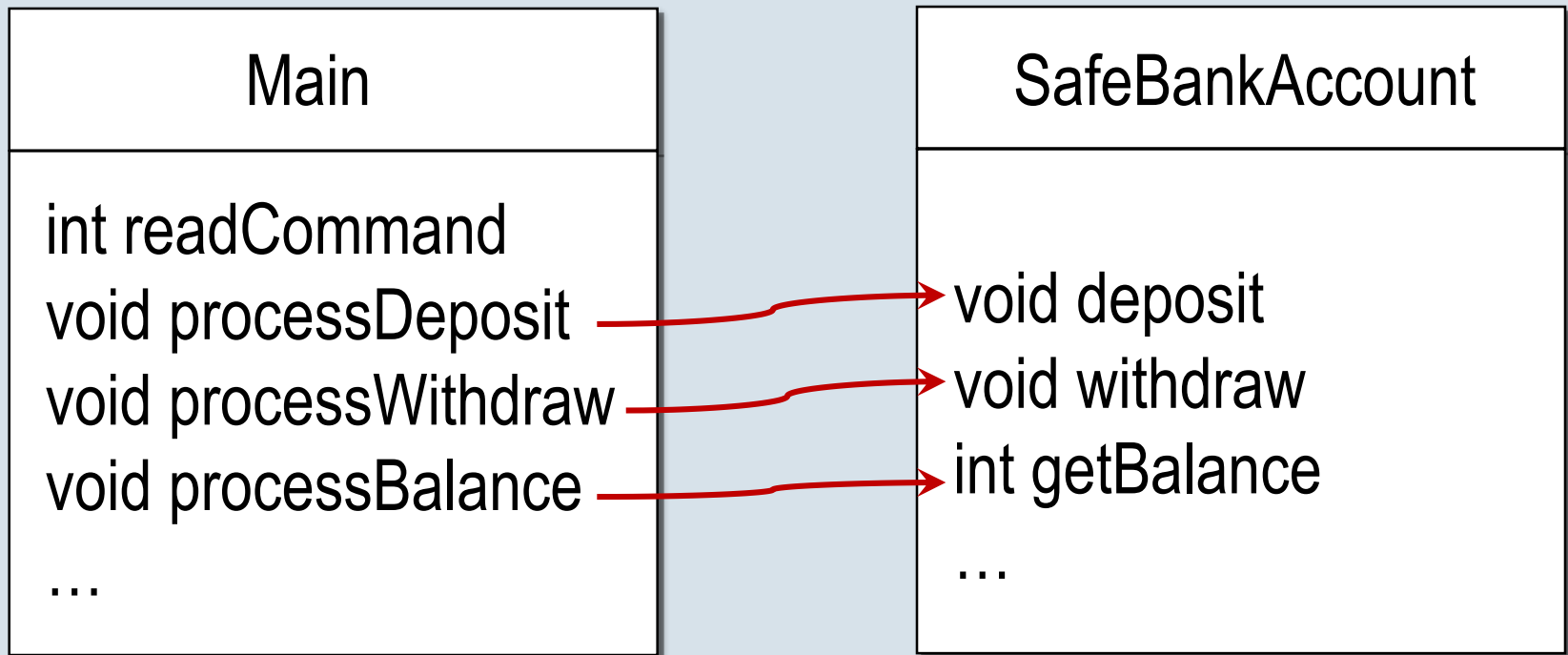
Main.java

- Por ser responsável pelo I/O, a classe `Main` estabelece uma “ponte” entre o *utilizador final* (que usa o programa e interage com ele por meio do teclado, consola, etc.) e as restantes classes do programa, ditas específicas do domínio (e.g., `Semaphore`, `SafeBankAccount`, `Rect`, etc.)



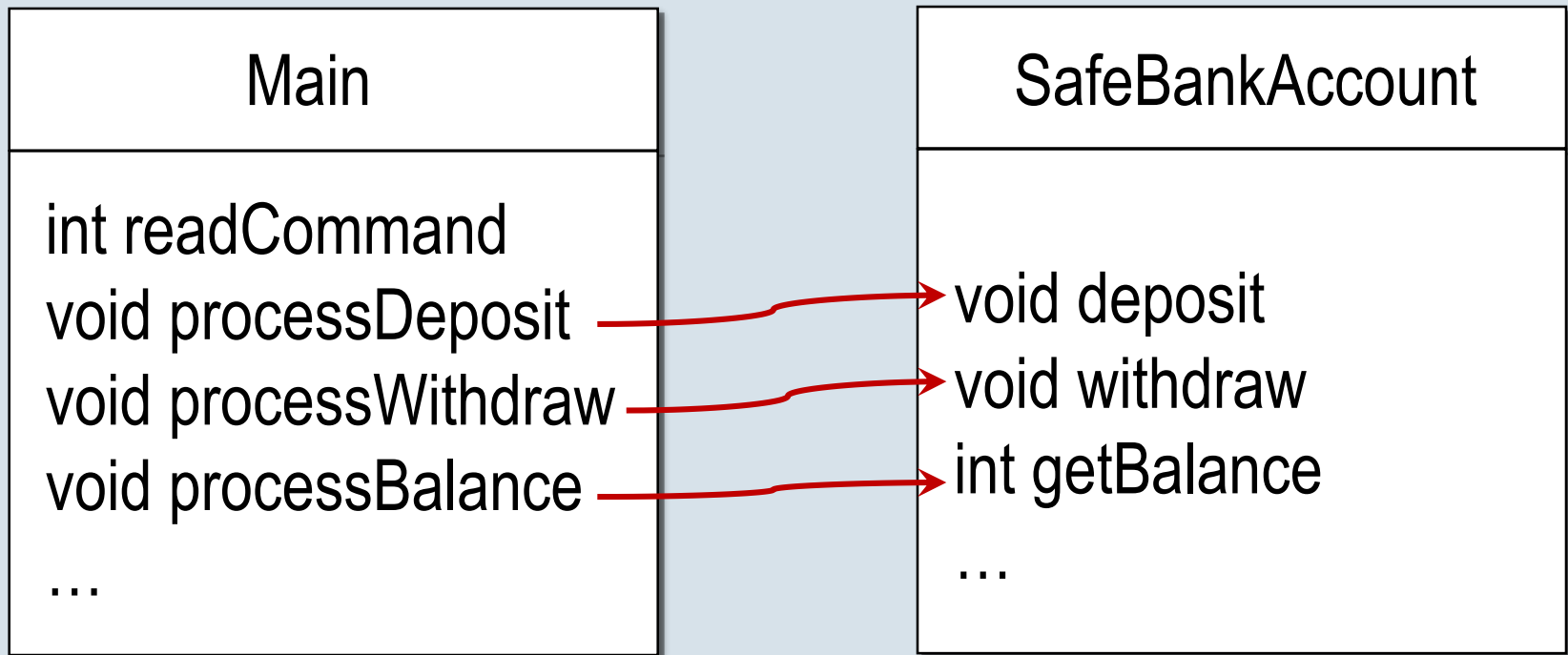
Main.java

- Porque estabelece a ponte entre `SafeBankAccount` e o utilizador final, é de esperar que haja alguma correspondência entre alguns dos métodos de `Main` e métodos de `SafeBankAccount`.



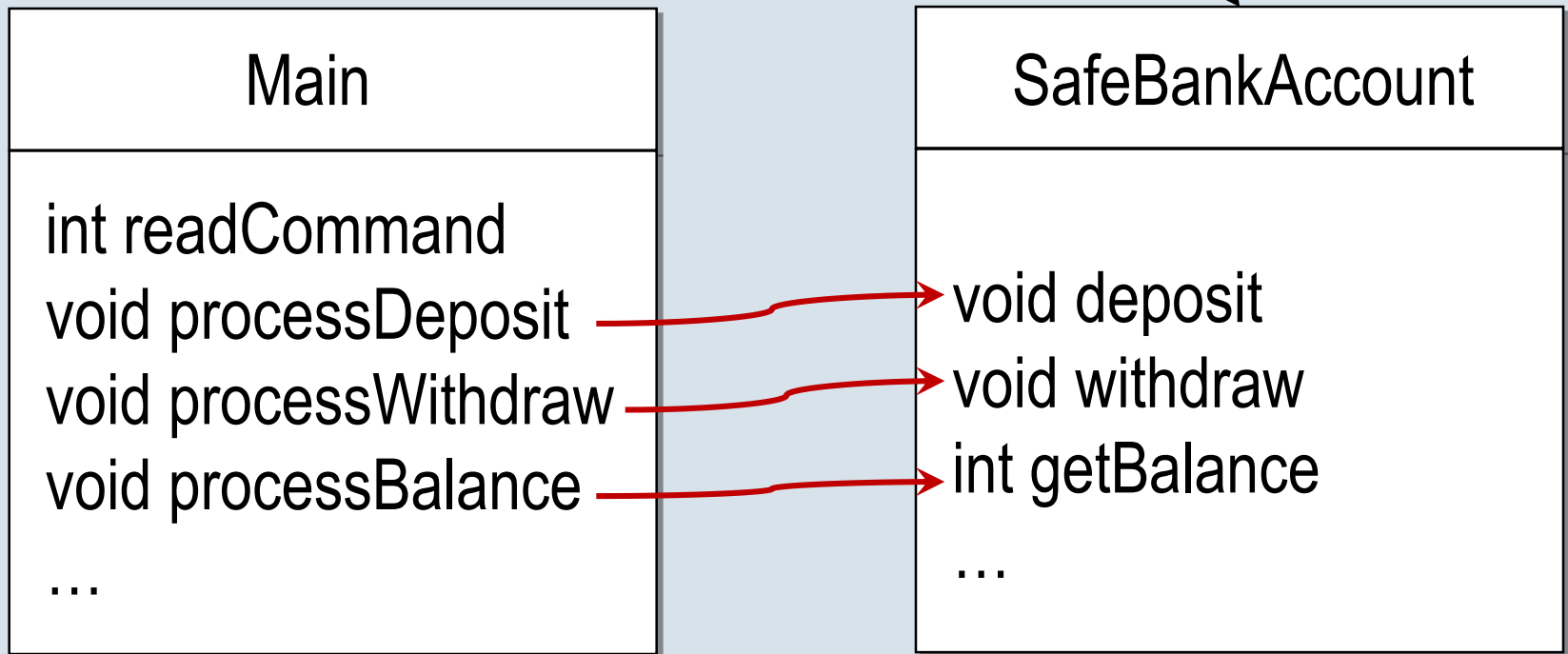
Main.java

- Assim, encontramos em cada classe – `Main` e `SafeBankAccount` – um método relacionado com depósitos, um método relacionado com levantamentos, etc. Porém, o propósito dos métodos nas duas classes é bastante diferente.



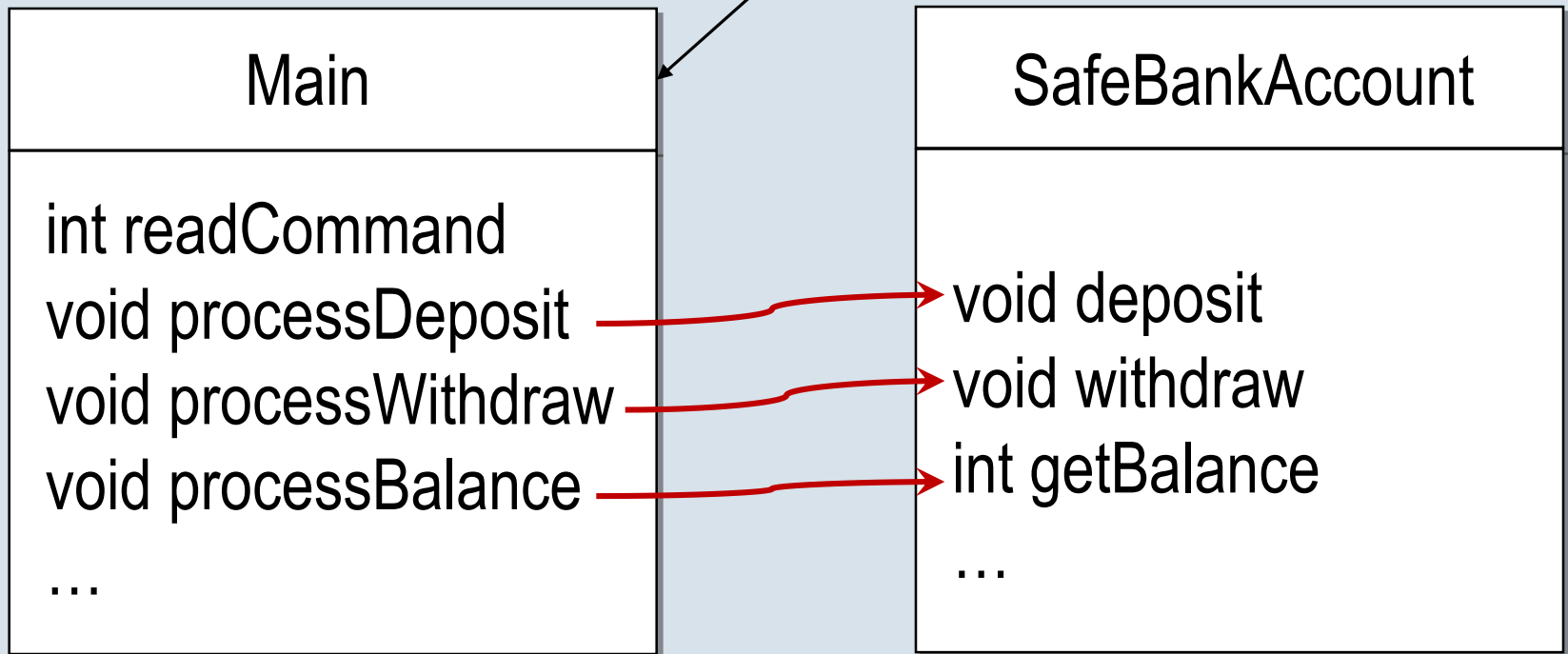
Main.java

A classe `SafeBankAccount` não sofre qualquer alteração: os seus métodos continuam dedicados a gerir o estado interno duma conta bancária.



Main.java

Os métodos – `private` e `static` – da classe `Main` destinam-se a realizar os comandos do interpretador: ler novo comando, determinar a tarefa a desempenhar, ler argumentos adicionais (e.g., saldo inicial, montante a depositar) e mostrar resultados na consola (e.g., saldo presente na conta, confirmação de que a operação teve sucesso, etc).



Porque tudo na `Main` deve ser privado?

- Só devemos dar visibilidade pública aos membros duma classe que se destinam ao seu uso externo
 - Essas partes públicas constituem a *interface* da classe
 - Usualmente, são operações – é por isso que as variáveis de instância costumam ser privadas
- Porém, os membros da classe `Main` nunca são chamados de outra classe. É função da `Main` chamar as outras classes, nunca o contrário. Esta situação faz com que não faça sentido dar visibilidade pública aos membros de `Main`.
- A exceção é o método `main()`, porque é obrigatório que seja público, pelas regras do Java.

Nova interação com o utilizador (interpretador de comando)

- Desenvolver um programa que:
 - Pede ao utilizador o saldo inicial duma conta bancária.
 - Fica à espera dum comando do utilizador. Os comandos permitidos são:

```
L <montante_levantar> → levantar  
D <montante_depositar > → depositar  
CS → consultar saldo  
CJA → consultar juro anual  
AJA → aplicar juro anual  
S → sair
```
 - Processa um comando. Para todos os casos deve escrever na consola se a acção foi bem sucedida ou não, e a informação pedida, caso exista.
 - Escrever na consola o saldo da conta.

Main.java

Nesta versão, a única modificação é que o comando é representado por meio duma `String` e não dum `int`. Esta é a prática mais habitual. Em relação à versão anterior de `Main`, as adaptações devem-se a esse aperfeiçoamento.

