# Exercício 5: Generalização/Especialização de classes

**Professor Miguel Melo**

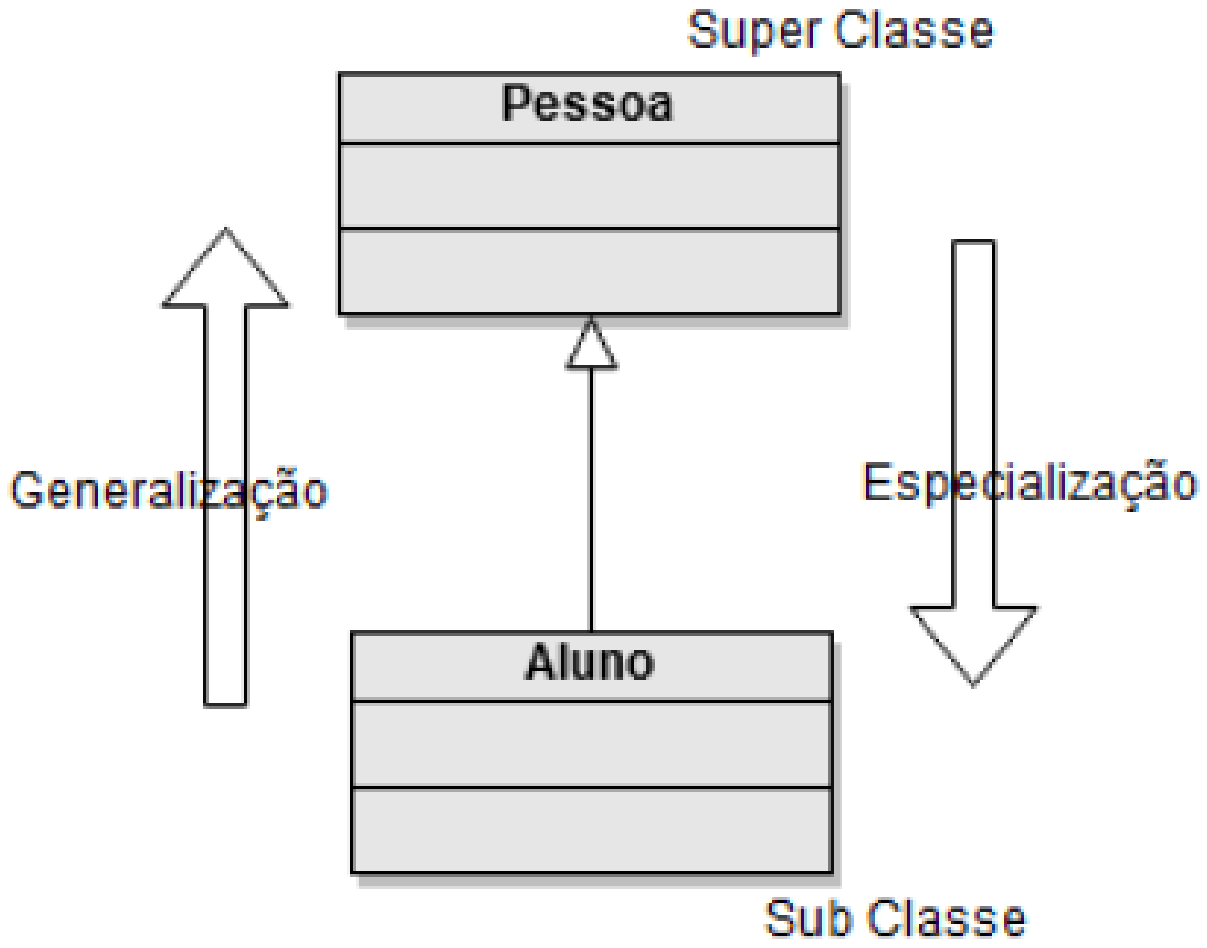
Programação Orientada a Objetos

**Engenharia Informática**

utad UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

# Associação entre classes - Generalização/Especialização

- A **Generalização** permite a especialização, partindo de uma classe mais genérica para uma mais específica
- Atributos e métodos definidos na classe base (superclasse) são herdados pelas classes derivadas (subclasse)
- É representado por um Triângulo **aberto** no lado da **classe base**



# Como declarar uma subclasse

```
class Funcionario : public Pessoa {
    private:
        string setor;
        int num;
```

# Como invocar métodos da super-classe

```
void Funcionario::Show(void)
{
            Show();
    cout << "Num:" << num << ";Setor:" << setor << "\n";
}
```

# Como invocar métodos da super-classe

```
void Funcionario::Show(void)
{
    Pessoa::Show();
    cout << "Num:" << num << ";Setor:" << setor << "\n";
}
```

# Algoritmos de ordenação (sorting algorithms)

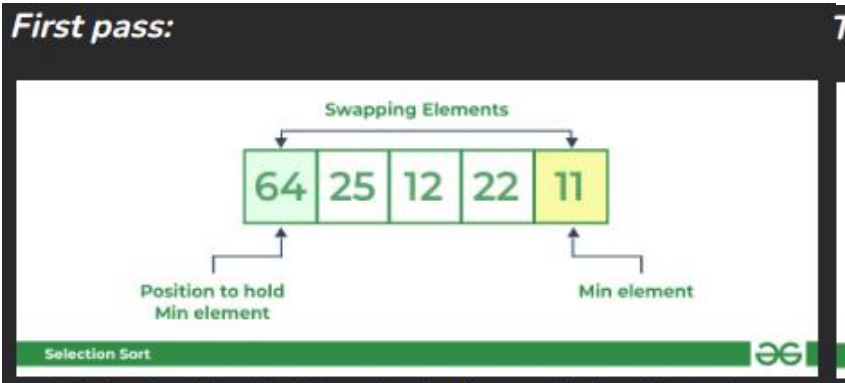
Usados para ordenar um array, vetor ou lista de elementos

Três abordagens populares:

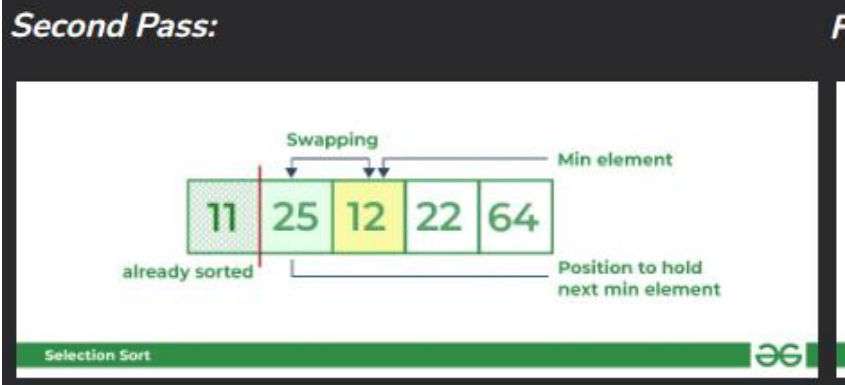
- ***Insertion sort:*** algoritmo simples, verificando os valores 2 a 2 e pondo-os por ordem
- ***Selection sort:*** simples e eficiente, pesquisa o valor mais baixo e troca-o com o valor da posição a ser processada
- ***Bubble sort:*** pesquisa o valor mais alto e coloca-o em primeiro de forma sucessiva

# Algoritmos de ordenação - Selection sort

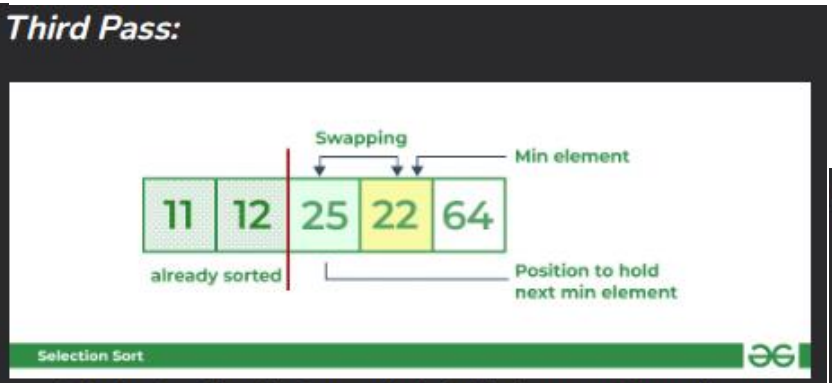
***Selection sort:*** simples e eficiente, pesquisa o valor mais baixo e troca-o com o valor da posição a ser processada



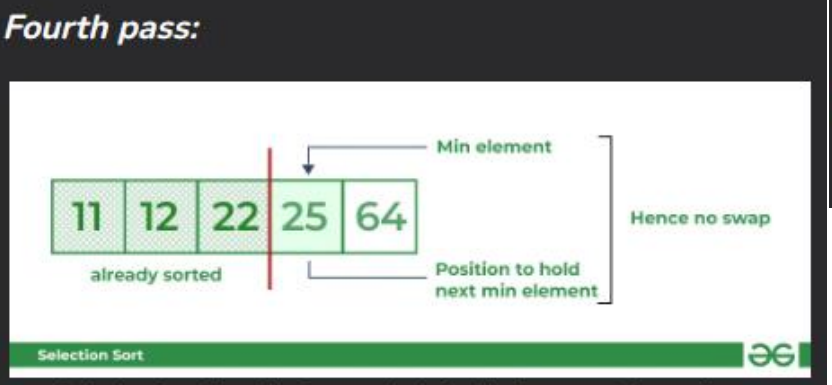

*Selection Sort Algorithm | Swapping 1st element with the minimum in array*



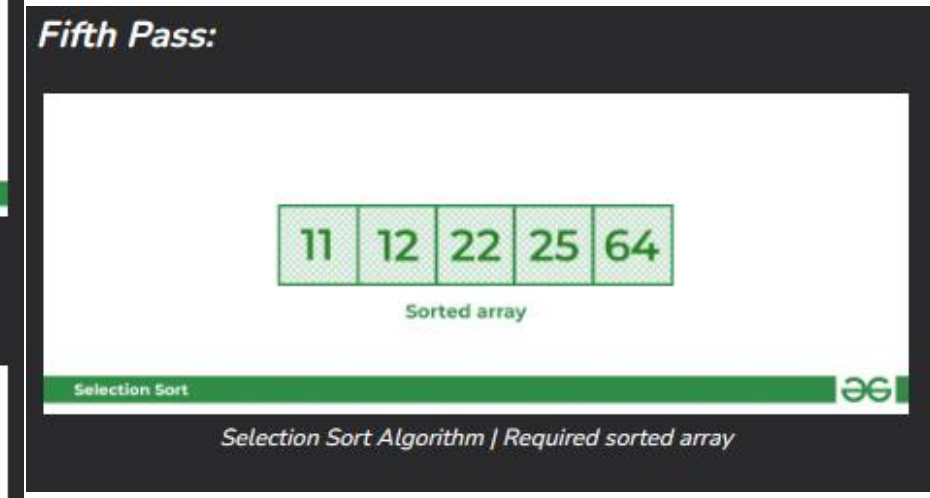

*Selection Sort Algorithm | swapping i=1 with the next minimum element*




*Selection Sort Algorithm | swapping i=2 with the next minimum element*



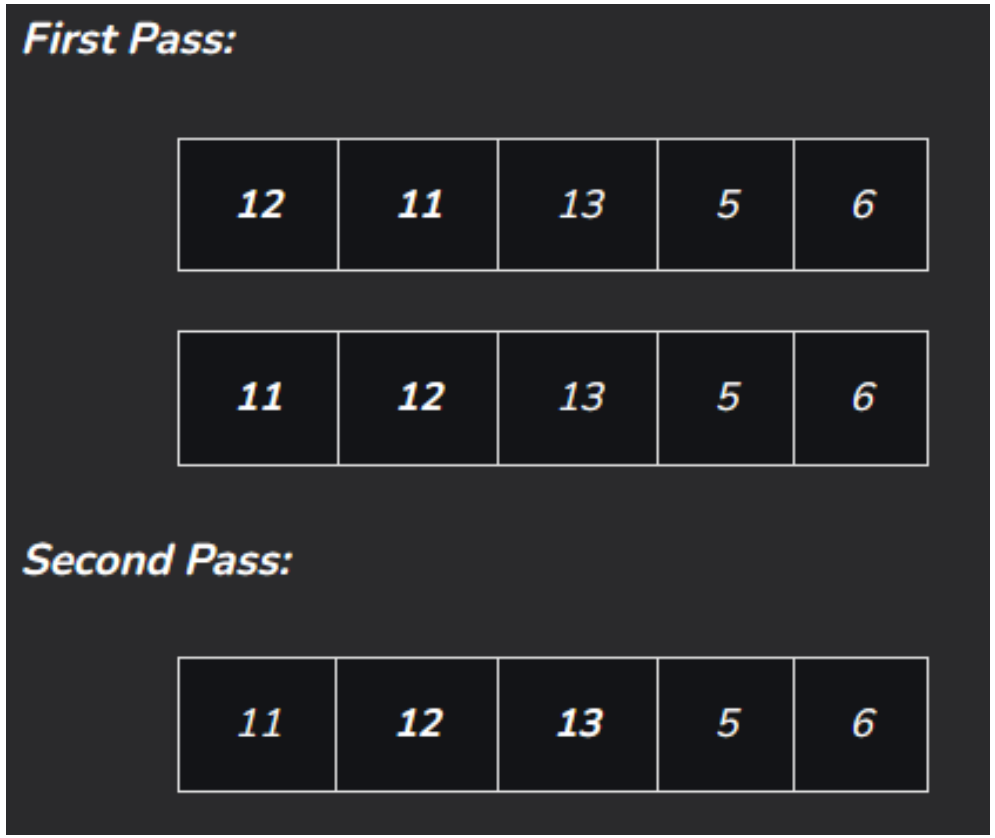

*Selection Sort Algorithm | swapping i=3 with the next minimum element*

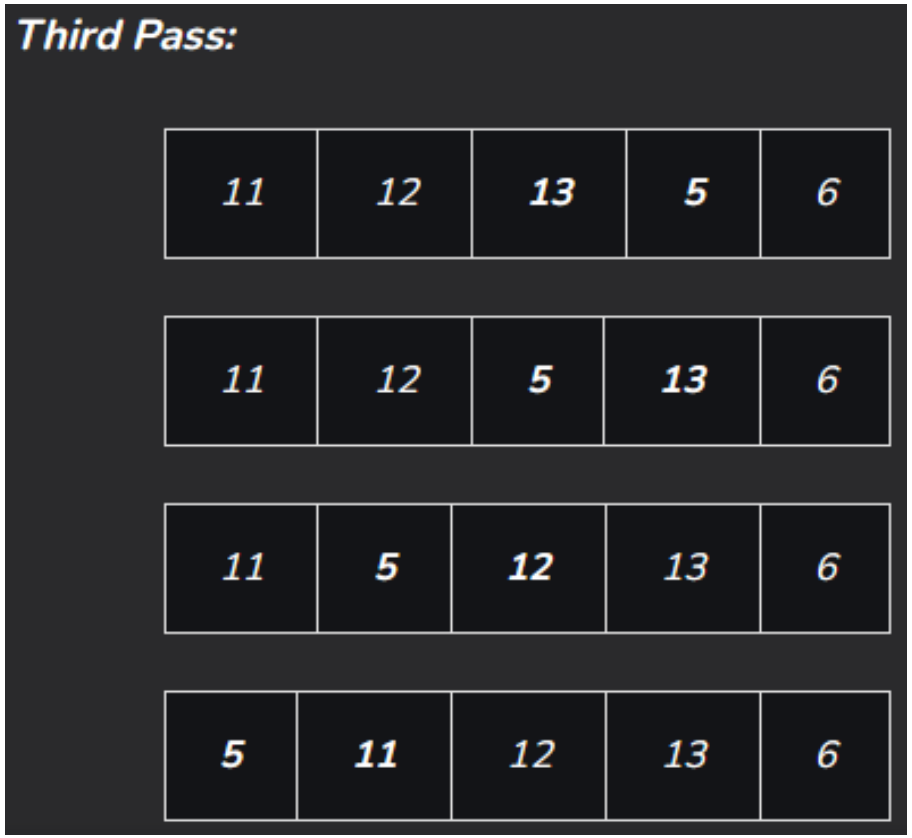



*Selection Sort Algorithm | Required sorted array*

# Algoritmos de ordenação - Insertion Sort

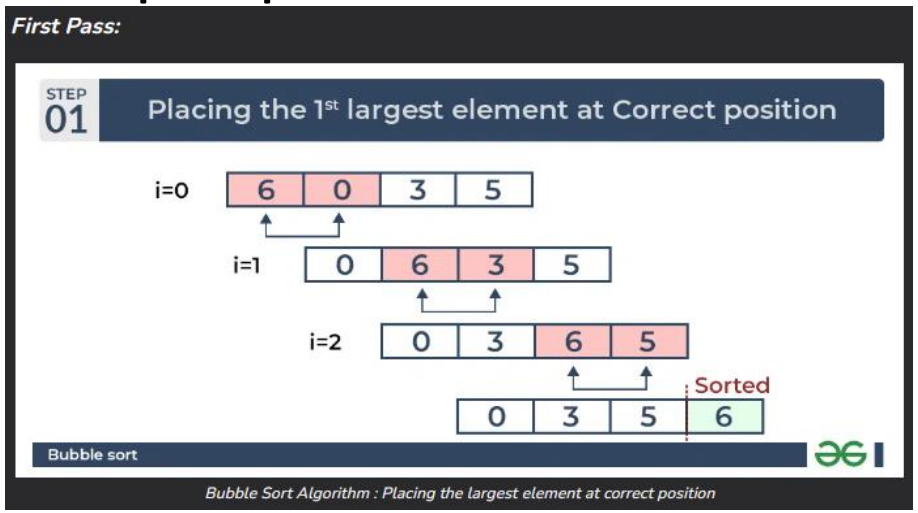
- ***Insertion sort:*** algoritmo simples, verificando os valores 2 a 2 e pondo-os por ordem



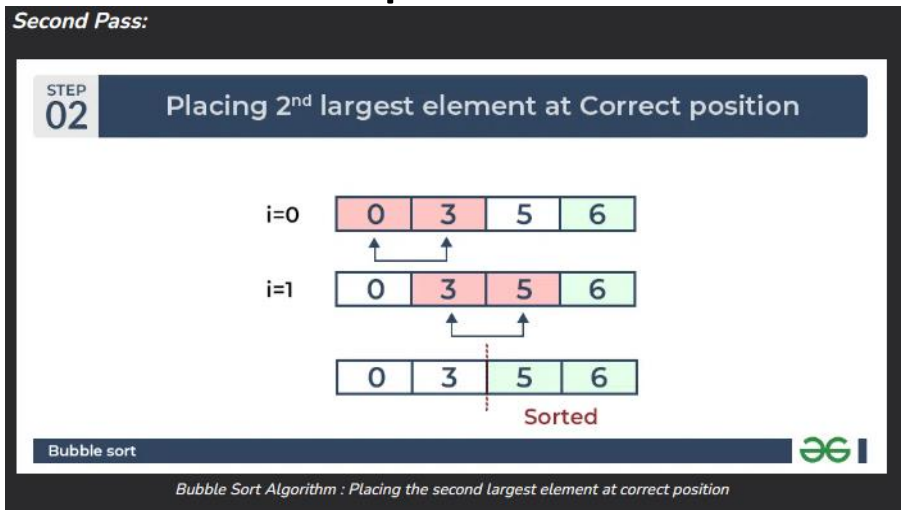



# Algoritmos de ordenação - Bubble sort

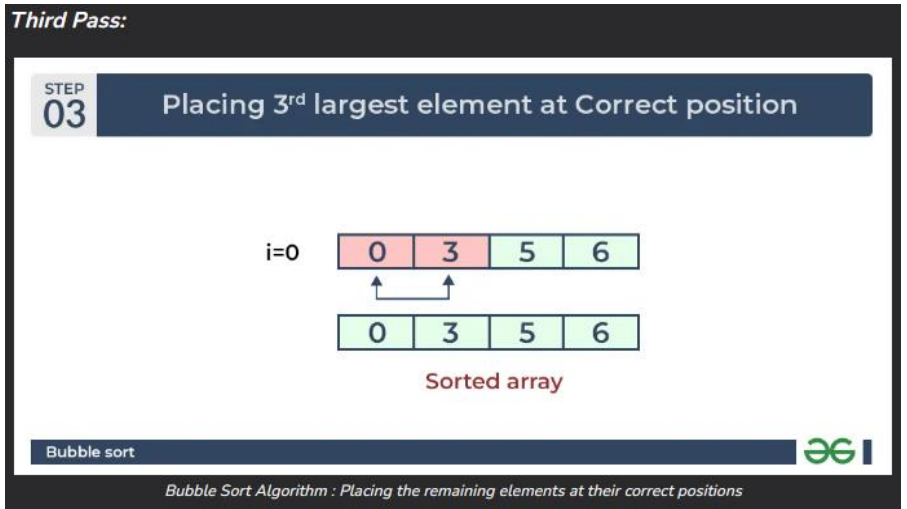
***Bubble sort:*** pesquisa o valor mais alto e coloca-o em primeiro de forma sucessiva



Bubble Sort Algorithm : Placing the largest element at correct position



Bubble Sort Algorithm : Placing the second largest element at correct position



Bubble Sort Algorithm : Placing the remaining elements at their correct positions

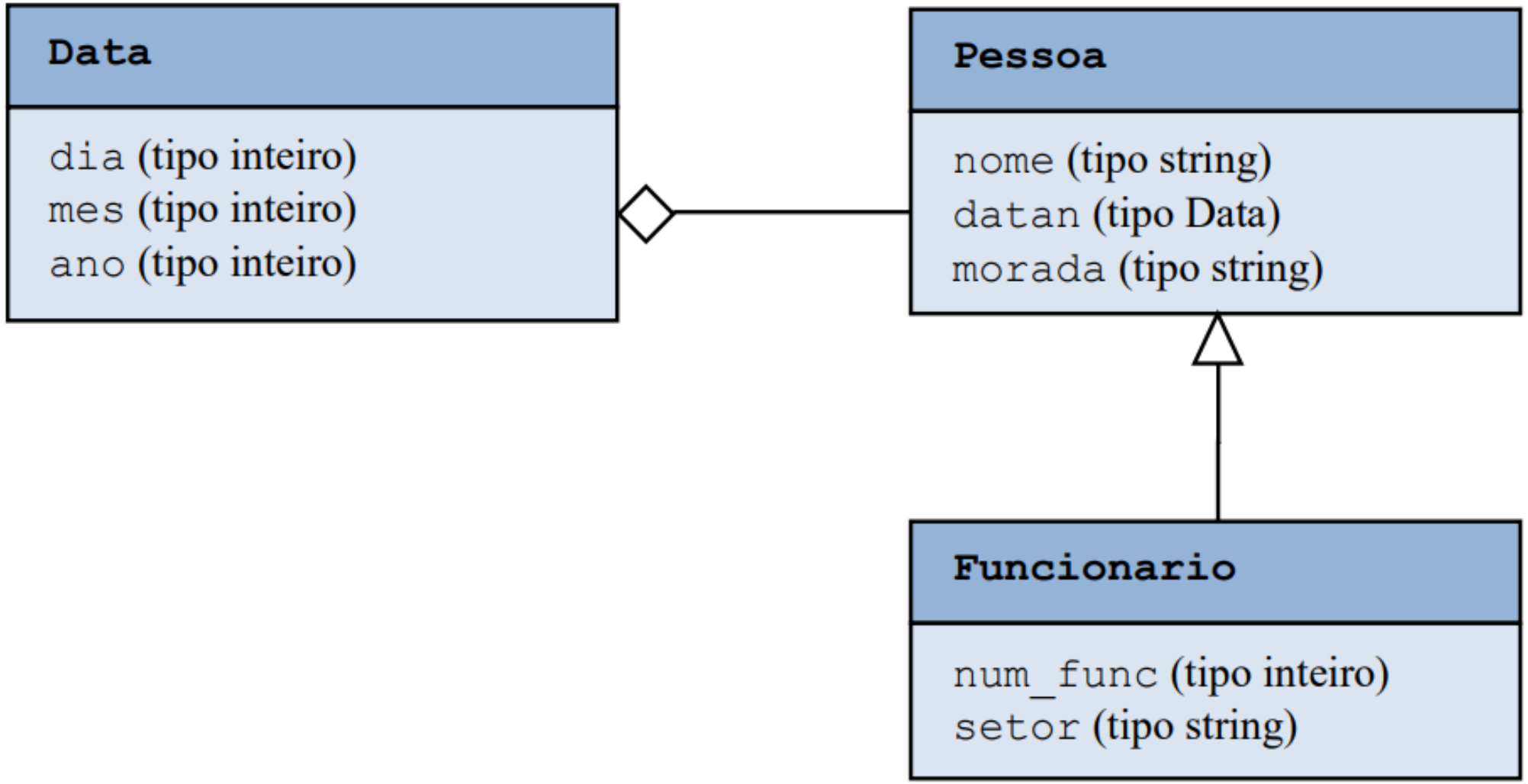
# Exercício 5 - Preparação do Projeto

- Tendo como ponto de partida o projeto do Exercício 4, podes:
  - Duplicar a pasta do Exercício 4 e alterar o nome para Exercício 5

  **Ou**

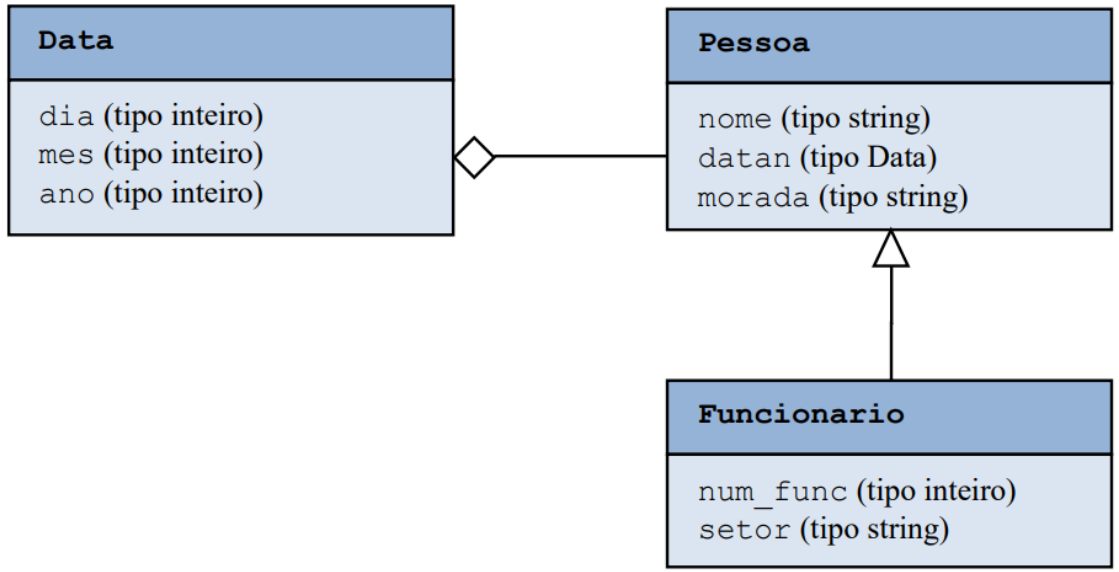
  - Salvar o projeto relativo ao Exercício 4 como Exercício 5:
    - "File" -> "Save Project As..." -> Salvar como Exercício 5
    - Isto vai-te permitir manter o Exercício 4 intacto e começar o novo Exercício 5 com tudo o que tens no Exercício 4

# Exercício 5


Data
dia (tipo inteiro)
mes (tipo inteiro)
ano (tipo inteiro)
Pessoa
nome (tipo string)
datan (tipo Data)
morada (tipo string)
Funcionario
num_func (tipo inteiro)
setor (tipo string)

# Exercício 5.1 a 5.3

**5.** Considere que uma fábrica de automóveis pretende desenvolver uma aplicação para fazer a gestão dos seus funcionários. São modeladas 3 classes: *Data*, *Pessoa* e *Funcionario*.




5.1. Defina a classe `Funcionario`, de acordo com o diagrama apresentado.
5.2. Implemente os construtores por defeito e com parâmetros da classe `Funcionario`.
5.3. Implemente os métodos assessores da classe.

# Exercício 5.1 a 5.3 – Proposta de Resolução

Funcionario.h

```h
#include "Pessoa.h"
#include <string>

class Funcionario : public Pessoa {
    private:
        string setor;
        int num;

    public:
        Funcionario();
        Funcionario(int _num, string _setor, string _nome, string _morada, Data _data);

        void SetNum(int _num) {
            num = _num;
        }

        void SetSetor(string _setor) {
            setor = _setor;
        }

        int GetNum(void) {
            return num;
        }

        string GetSetor(void) {
            return setor;
        }
};
```

Funcionario.cpp

```cpp
#include "Funcionario.h"

Funcionario::Funcionario()
{
}

Funcionario::Funcionario(int _num, string _setor, string _nome, string _morada, Data _data) :Pessoa(_nome, _morada, _data)
{
    setor = _setor;
    num = _num;
}
```

# Exercício 5.4 e 5.5

5.4. Num programa, defina 2 objectos do tipo Funcionario com os seguintes atributos:

a) Joca Gaio; 20/9/1987; Rua da direita n 2; 25023; Pintura

b) Ana Rola; 4/2/1990; Rua da esquerda n 3; 25024; Comercial

5.5 Mostre no ecrã os dados dos funcionários criados na alínea anterior.

# Exercício 5.4 e 5.5 – Proposta de Resolução

Método main()

```
Funcionario a(25023, "Pintura", "Joca Gaio", "Rua da direita n2", Data(20,9,1987));
Funcionario b(25024, "Comercial", "Ana Rola", "Rua da esquerda n3", Data(4,2,1990));

a.Show();
b.Show();
```

Funcionario.cpp

```
void Funcionario::Show(void)
{
    Pessoa::Show();
    cout << "Num:" << num << ";Setor:" << setor << "\n";
}
```

*não te esqueças de especificar o método no ficheiro .h

# Exercício 5.6 e 5.7

5.6. Atualize a morada do “Joca Gaio” para “Rua do meio n 4” e o setor para “Ferragem”.

5.7. Na data de nascimento da “Ana Rola”, altere o dia para “23”

**Proposta de resolução:** usa os getters e setters adequados para o efeito e chama o método Show() de seguida para mostrar os dados atualizados dos funcionários

# Exercício 5.8 e 5.9

5.8. Implemente a sobrecarga do método >> para ler a informação de um funcionário, inserida pelo teclado

5.9. Defina mais 2 objectos do tipo Funcionario e leia os seus dados através do teclado.

# Exercício 5.8 e 5.9 - Proposta de resolução

Ficheiro .h

```
friend istream & operator >> (istream &is, Funcionario &F);
```

Método *main*

```
Funcionario c,d;
cin >> c;
fflush(stdin);
cin >> d;
```

** o fflush permite libertar a stream*
*de forma a usar uma nova para o cin seguinte*

Ficheiro .cpp

```
istream & operator >> (istream &is, Funcionario &F)
{
    char aux[100];

    is >> (Pessoa &)F;
    cout << "Num:";
    is >> F.num;
    cout << "Setor:";
    is >> aux;
    F.setor = aux;

    return is;
}
```

# Exercício 5.10 e 5.11

5.10. Implemente o método SaveFile()para guardar os dados de um funcionário em ficheiro.

5.11.No programa principal, crie um ficheiro, cujo nome seja lido através do teclado e guarde a informação dos 4 funcionários, um por linha, com os campos separados por ‘;’ .

# Exercício 5.10  e 5.11 - Proposta de resolução

## Ficheiro .h

```
void SaveFile(ofstream &os);
```

## Ficheiro .cpp

```
void Funcionario::SaveFile(ofstream &os)
{
    Pessoa::SaveFile(os);
    os << num << ";" << setor << "\n";
}
```

## Método *main*

```
ofstream ofile;
string str = "funcionarios.txt";

ofile.open(str.c_str());

if (ofile)
{
    a.SaveFile(ofile);
    b.SaveFile(ofile);
    ofile.close();
    cout << "Ficheiro " << str << " criado com sucesso!" << endl;
}
else
{
    cout << "ERRO: não é possível abrir o ficheiro " << str << '\n';
}
```

# Exercício 5.12 e 5.13

5.12. Implemente o método ReadFile()para ler os dados de um funcionário a partir de um ficheiro.

5.13.No programa principal, leia do ficheiro, criado na alínea 5.10, os dados de todos os funcionários para um vector de objectos.

# Exercício 5.12  e 5.13 - Proposta de resolução

Ficheiro .h

```
void ReadFile(ifstream &is);
```

Ficheiro .cpp

```
void Funcionario::ReadFile(ifstream &is)
{
    char aux[100];

    Pessoa::ReadFile(is);
    is.getline(aux, 100, ';');
    num = atoi(aux);
    is.getline(aux, 100, '\n');
    setor = aux;
}
```

Método *main*

```
ifstream ifile;
ifile.open(str.c_str());

Funcionario funcionarios[4];

if (ifile)
{
    int i =0;
    while (ifile.peek() != EOF)
    {
        funcionarios[i].ReadFile(ifile);
        i++;
    }

    ifile.close();
    cout << "Ficheiro " << str << " lido com sucesso!" << endl;
}
else
{

    cout << "ERRO: não é possível abrir o ficheiro " << str << '\n';
}
```

# Exercício 5.14

5.14.Mostre, no ecrã, os dados de todos os Funcionários, ordenados pela data de nascimento

# Exercício 5.14

Método main():

```
SelectionSort(funcionarios,2);

cout << "Vetor ordenado: \n";
int contador = 2;
for (int i = 0; i < contador; i++)
    cout << funcionarios[i] << "\n";
```

Funções de *sort*:

```
void BubbleSort(Funcionario *A, int size);
void SelectionSort(Funcionario *A, int size);
void InsertionSort(Funcionario *A, int size);
void Swap(Funcionario *x, Funcionario *y);

void BubbleSort(Funcionario *A, int size)
{
    bool swaped;
    int i, j;

    i = size;
    do {
        swaped = false;
        for (j = 1; j<i; j++)
            if ((A + j - 1)->GetNome() >(A + j)->GetNome())
            {
                Swap(A + j - 1, A + j);
                swaped = true;
            }
        i--;
    } while (i>0 && swaped);
}

void SelectionSort(Funcionario *A, int size)
{
    int i, j, aux;

    for (i = 0; i<size - 1; i++)
    {
        aux = i;
        for (j = i + 1; j<size; j++)
            if ((A + j)->GetData() < (A + aux)->GetData())
                aux = j;
        Swap(A + aux, A + i);
    }
}

void InsertionSort(Funcionario *A, int size)
{
    int i, j;
    Funcionario aux;

    for (i = 1; i<size; i++)
    {
        aux = *(A + i);
        j = i;
        while (j>0 && (A + j - 1)->GetNome() > aux.GetNome())
        {
            *(A + j) = *(A + j - 1);
            j = j - 1;
        }
        *(A + j) = aux;
    }
}

void Swap(Funcionario *x, Funcionario *y)
{
    Funcionario aux;

    aux = *x;
    *x = *y;
    *y = aux;
}
```

# Exercício 5: Generalização/Especialização de classes

**Professor Miguel Melo**

Programação Orientada a Objetos

**Engenharia Informática**

utad UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO