Programação Orientada a Objectos
22/23

- Breve introdução aos princípios de orientação a objetos

Programação orientada a objetos – Breve introdução aos princípios

- 3 princípios principais
  - Abstração de dados e Encapsulamento
  - Herança
  - Polimorfismo

Programação orientada a objetos – Breve introdução aos conceitos

- **Abstração de dados e Encapsulamento**
  - Os conceitos com que o programa trabalha passam a ser o bloco constituinte do programa a partir do qual tudo o resto é contruído
  - Cada conceito é materializado numa classe que contém não só os dados, mas também as funções para lhes aceder e manipular
  - A classe oculta os dados internos, tornando o resto do programa independente dos pormenores de implementação do conceito.

O encapsulamento é um aspeto muito importante pois permite fazer programas mais robustos e menos propensos a erros

Abstração de dados e Encapsulamento

Exemplo – programa que lida com dados de pessoas

Em vez de se ter
- Variáveis do tipo estrutura “Pessoa” e funções que recebem essas variáveis para as trabalhar

Passa a ter-se
- Objetos (espécie de variável) de uma classe (tipo de dados) pessoa.
- Esses objetos agem como se as funções (“métodos”) para os manipular estivessem dentro do próprio objeto (funções “dentro” de variáveis), apesar de, na verdade, o código não estar dentro de dados.
- O resto do programa **não tem acesso direto** aos dados internos que definem o conceito. A preocupação do programador (do resto do programa) passa a ser “**como é que se usa** aquilo” em vez de “como é que aquilo está feito por dentro”

Programação orientada a objetos – Breve introdução aos princípios

- Herança
  - O mecanismo de herança permite **estender um conceito**, construindo novos a partir de um conceito base já existente, reutilizar todos os mecanismos nele já existentes (reaproveitamento de dados e funções)
  - Significa que se podem definir classes aproveitando dados e funções já existentes noutra.
  - Dependendo do contexto do programa, poder-se-á utilizar objetos da classe estendida onde for válida a utilização dos objetos da classe original.

A herança permite evitar código repetido e resulta em programas mais pequenos que os seus equivalentes de linguagens não orientadas a objetos

Herança

- Exemplo
  - Se o programa já souber lidar com o conceito "Pessoa", pode-se facilmente **estender** esse conceito para construir o conceito "Estudante" (pessoa com a característica adicional de ser estudante numa escola)
  - Não será necessário repetir em "Estudante" toda a lógica (definições, dados e funções) relativos à parte do conceito de "Estudante" que corresponde (é comum) ao conceito de "Pessoa"
  - Poderá haver partes do programa que consigam lidar tanto com variáveis (objetos) Pessoa como Estudante sem alteração (polimorfismo)

Programação orientada a objetos – Breve introdução aos princípios

- Polimorfismo
  - Permite trabalhar objetos ("variáveis") de tipos de dados diferentes através do mesmo código, aumentando ainda mais a poupança de código e evitando repetição
    - de lógica no programa – os métodos são partilhados
    - de estruturas de dados – a mesma coleção aceita guardar vários tipos de dados
  - Os objetos **terão que ter uma base comum** e é através dessa base que é feita a sua manipulação
    - Essa base comum significa que está a ser usado o mecanismo de herança

Polimorfismo

- Exemplo

Um programa que tem dois conceitos (tipos de dados: estruturas, classes): **Pessoa** e **Estudante**, e um mecanismo (função) para cumprimentar a pessoa e o estudante.

Programa não orientado a objetos:
- Existe um mecanismo (função) para cumprimentar a pessoa **e outro** para cumprimentar o estudante porque Pessoa e Estudante são dois tipos de dados diferentes e incompatíveis, obrigando a duplicação de código

Programa orientado a objetos:
- Pessoa e Estudante são classes, sendo que Estudante estende a classe Pessoa. A função para cumprimentar pessoas, se estiver bem programada, também aceitará objetos de estudante porque estudante é um tipo particular do conceito mais geral pessoa

O polimorfismo é uma das maiores vantagens das linguagens orientadas a objetos

Programação orientada a objetos – Breve introdução aos princípios

-> Encapsulamento e abstração + Herança + Polimorfismo

- Os princípios de programação orientada a objetos são por vezes referidos como *princípios SOLID*

Princípios SOLID

- ***S*** ***-> Single responsibility***: Cada classe representa um único conceito e lida com uma única funcionalidade (responsabilidade) especializada
- ***O*** ***-> Open-closed***: open for extension, closed for modification
- ***L*** ***-> Liskov Substitution***: outro termo para polimorfismo. Uma função que lida com um objeto de uma classe base deve ser capaz de lidar, de forma transparente com um objeto da uma classe derivada sem
- ***I*** –> **Interface Segregation** – É melhor ter **muitas *interfaces* específicas** (especializadas) do que uma interface que faz tudo. Neste contexto, *interface* significa a funcionalidade que a classe expõe (relacionado com o princípio "S")
- ***D*** **– *Dependency Inversion*** – O programa deve depender de conceitos e não de implementações (**interessa o que as classes dizem que fazem, não como o fazem**)

Programação orientada a objetos – Breve introdução aos princípios

- Encapsulamento, Herança, Polimorfismo
- Princípios SOLID

- O que aqui está neste momento é uma breve enumeração: pouco mais do que "*estes conceitos existem e têm a ver com esta disciplina*"
  - O verdadeiro significado e alcance destes conceitos só se entenderão do meio do semestre em diante
  - Mas é importante ficar claro desde já que **a avaliação não incide só em funcionalidade (o programa funciona) mas sim como é que está feito por dentro e como é que estes princípios foram aplicados**