UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BCC35A - Linguagens de Programação

Prof. Dr. Rodrigo Hübner

Aula 04: Polimorfismo; Herança e composição; Classes abstratas e interfaces; Templates.

Herança

- **Principal mecanismo** de OO:
  - Permite criar novos tipos baseado em tipos existentes
- **Reuso**
  - **Subtipo** herda os dados e funcionalidades de **supertipo**
  - É possível **adicionar novos dados e funcionalidades**
  - É possível **alterar as funcionalidades** existentes: sobrescrita de métodos
- **Organização**
  - É possível criar hierarquia de classes

Herança simples e múltipla

- **Herança múltipla** permite que uma subclasse derive de mais de uma classe
  - Vantagem: às vezes a herança múltipla é útil (quando?)
  - Desvantagem: complexidade na implementação (colisão de nomes)
  - Desvantagem: ineficiência, herança múltipla custa mais que herança simples
  - Desvantagem: o projeto das classes é mais difícil (aumento na manutenção)

Herança em `JavaScript`

- O suporte a classes em `JavaScript` foi adicionado na versão `ECMAScript 6` (2015)
- Antes, `JavaScript` dava suporte a orientação a objetos por meio dos protótipos (até mesmo com construtores).
- Ver (e alterar) `herança.js`

Herança (`Java` vs `C#`)

- `Java`
  - Similar a `C++`, todas as classes são descendentes de `Object`
  - `Autoboxing`
  - Método declarado com `final` não pode ser **sobrescrito**
  - **Herança múltipla** por meio de **Interfaces**

```
Integer integer = 9;
...
int in = 0;
in = new Integer(9);
```

Herança (`Java` vs `C#`)

- `C#`
  - Em **sobrescrita de método**, classe pai possui `virtual` e filho possui `override`
  - Uma classe com **pelo menos um método abstrato**, deve ser marcado com `abstract`

Polimorfismo

- Uma **variável polimórfica**, declarada como sendo de uma determinada **classe**, é capaz de **referenciar** ou **apontar** objetos da **mesma classe** ou de qualquer uma de suas **subclasses**.
- Podemos dizer que **polimorfismo** é a capacidade de **tratar objetos por seus supertipos**.
- Analisar: `herança.cpp` e `polimorfismo.cpp`

Classes e métodos abstratos

- **Método abstrato** não contém implementação, apenas definição
- **Classe abstrata** contém ao menos um método abstrato
- **Classe abstrata pura** ou **interface** contém somente métodos abstratos
- **Classe abstrata** e **interface** não podem ser instanciadas

Classes e métodos abstratos em Java

- Podemos escrever classes abstratas da seguinte forma:

```
abstract class Tipo {
  public int op1() {
     ...
  }
  public abstract double op2();
}
```

Classes e métodos abstratos em `Java`

- Podemos escrever classes abstratas puras ou interface da seguinte forma:

```
interface Tipo {
  int op1();
  double op2();
}
```

- Tanto para **classes abstratas** quanto para **interfaces**, no uso (**instância**) é necessário anotar com `@override`.
- Ver `Shape.java`

Herança múltipla em `C#` e `Java`

- Em ambas as linguagens é necessário utilizar de `interface`, para que seja possível **utilizar atributos** e **somente implementar métodos** das classes herdadas.
- Ver e executar `herança_multipla.cs`

Templates

- Realiza a **conversão** de **tipos da assinatura** de um método em **tempo de execução**
- É uma **alternativa** à **sobrecarga de funções**, quando estas envolvem lógicas de programas e operações idênticas para vários tipos de dados
- Ver e executar:
  - `templates.cpp`
  - `template_class.cpp`

Próxima aula

- **Prática**: trabalhando com o conteúdo visto em suporte OO para as LPs dos trabalhos.