

Criação de Classes

Sérgio Lopes

- Saber criar uma classe com todos os seus componentes

Objectivos

Atributos

Definem as características dos objectos.

```
public String nome;  
private int idade;  
String morada;
```

<visibilidade> <tipo de dados> <nome> [= <valor inicial>];

Criação de atributos em classes...

Exemplo

Métodos

Definem o comportamento dos objectos.

```
public String getNome() {  
    //...  
}
```

```
<visibilidade> <tipo de dados> <nome> ([<parametros>]) {  
    <implementação>  
}
```

Criação de métodos em classes...

Exemplo

Um construtor permite a criação de objectos, e caso necessário, a definição de valores iniciais.

The slide features two solid green rectangular bars at the bottom. The first bar is a small square on the left, and the second is a long horizontal bar extending across the bottom right.

Construtores

Objectivos dos Construtores

Permite obter as referências para a memória.

São responsáveis por criar o objecto, transformando a definição da classe em algo que podemos usar (instanciação).

Permitem definir valores iniciais para o estado do objecto.

Regras

Podem ter qualquer acesso (atenção ao private!).

Não têm valor de retorno.

Têm sempre o nome da classe.

Podem existir vários, desde que tenham parâmetros diferentes.

Se o construtor cria, o destrutor destrói!

Destruítores

Objectivos dos Destrutores

Remover algum recurso que não seja imediatamente removido pelo Garbage Collector.

São invocados automaticamente pelo Garbage Collector (logo não servem para remover a própria classe!).

Devem ser usados com cuidado.

Método *finalize*

Os destrutores são implementados através do método *finalize*, existente em todas as classes.

É a única forma de se usar um destrutor.

Construire classes...

Exemplo

Facilitam a organização do código e a resolução de conflitos de nomenclatura.

The word "Packages" is centered between two solid green rectangular bars. The bar on the left is short and vertical, while the bar on the right is long and horizontal.

Packages

O que são

Pastas em disco onde os nossos ficheiros de código estão localizados.

Organizam os ficheiros e permitem que os nomes das classes sejam únicos, evitando conflitos no carregamento das classes pela plataforma.

São apenas pastas!

Organizar o código...

Exemplo

Permitem a aplicação prática da teoria de encapsulamento...

Modificadores de Acesso

Propósito

Tornar o conceito de encapsulamento aplicável.

Afectam o acesso aos atributos, métodos e classes.

Devem ser usados!

Modificador **public**

Permite o acesso completo ao recurso (classe, método ou atributo) que estão a controlar.

Todas as classes terão acesso completo ao recurso.

Não é possível controlar o acesso e impedir estragos!

Modificador **protected**

Permite o acesso a classes descendentes e a outras classes que estejam no mesmo package.

Úteis para métodos que devem ser redefinidos pelas sub-classes mas que não devem ser usados indiscriminadamente.

Não fazem o que é suposto!

Sem Modificador

Também chamado de acesso por omissão. Permite o acesso a todas as classes no mesmo package.

Não permite o acesso a descendentes que estejam em packages diferentes.

Modificador **private**

Remove todo e qualquer acesso, ficando o recurso visível apenas à própria classe/objecto que o define.

Deve ser usado em todos os atributos e em métodos que digam respeito ao comportamento interno do objecto.

Começar sempre por este!

Tabela de Acesso

	Própria	Package	Descendentes	Outros
public	X	X	X	X
protected	X	X	X	
sem modificador	X	X		
private	X			

Encapsular correctamente...

Exemplo

Questões

- Criação de atributos segue regras que já nos são familiares
- Criação de métodos também, (Sem **void** dentro de parêntesis)
- Construtores úteis para criar objectos
- Modificadores de acesso para controlar quem usa os recursos

Resumindo

A linguagem Java tenta que a implementação dos conceitos de POO seja o mais próxima possível da definição dos mesmos.

Conclusão