Programação por Objectos

Java

Parte 2: Classes e objectos

Classes (1)

Sintaxe

Qualif* class Ident

[extends IdentC] [implements IdentI [,IdentI]*] {

[Atributos | Métodos]*

}

- Qualif: qualificador (visibilidade, entre outros)
- Ident: identificador da classe
- extends IdentC: especialização de superclasse
- implements IdentI: realização de interfaces

Classes (2)

- **Qualificadores de classe**:
  - **public**: a classe pode ser publicamente acedida (os membros públicos da classe são disponibilizados em todo o programa).
  - **abstract**: a classe não pode ser instanciada (normalmente, porque alguns método são apenas protótipos).
  - **final**: a classe não pode ter subclasses.
- Na omissão do qualificador **public**, uma classe é apenas acessível no pacote onde está definida.
- Uma classe pode ter mais do que um qualificador. Contudo, uma classe não pode ser ao mesmo tempo **abstract** e **final**.

Classes (3)

```
public class Conta {
        /* atributos */
        /* métodos */
}
```

Atributos (1)

Sintaxe

Qualif* Tipo Ident [= Expr] [, Ident = Expr]* ;

- Qualif: qualificador (visibilidade, entre outros)
- Ident: identificador do atributo
- Tipo: tipo do atributo
- Expr: inicialização do atributo

Atributos (2)

- Tipos possíveis do atributo:
  - Primitivos:
    - `boolean`
    - `char`
    - `byte`
    - `short`
    - `int`
    - `long`
    - `float`
    - `double`
  - Referências: classes e interfaces definidas pelo Java, por exemplo, classe `String`, e classes e interfaces definidas pelo programador.

Atributos (3)

- **Qualificadores de atributo**:
  - Visibilidade:
    - **public**: atributo acessível onde quer que a classe seja acessível.
    - **private**: atributo acessível apenas na classe.
    - **protected**: atributo acessível na classe, subclasses e classes no mesmo pacote.
  - **static**: atributo de classe.
  - **final**: atributo constante.
  - **transient**: atributo que não vai ser serializado.
- No caso de omissão de um qualificador de visibilidade, o atributo é acessível na classe e classes no mesmo pacote.
- Com excepção dos qualificadores de visibilidade, um atributo pode ter mais do que um qualificador.

Atributos (4)

- **Princípio de encapsulamento da informação**:
  - Os atributos não devem ser acedidos fora do objecto a que pertencem, devendo ser apenas alterados por métodos (modificadores).
  - A visibilidade dos atributos deve ser **private** ou **protected**. Evitar, o mais possível, o qualificador **public**.

Atributos (5)

- **Inicialização dos atributos**:
  - Expr pode ser uma constante, um outro atributo, a chamada a um método, ou uma expressão envolvendo qualquer destes.
  - Por omissão, quando um atributo não é inicializado é-lhe atribuído um valor inical, dependente do seu tipo:
    - **boolean** – false
    - **char** – '\u0000'
    - **byte**, **short**, **int**, **long** – 0
    - **float**, **double** – +0.0
    - referênca para um objecto – null
  - Um atributo pode ser (explicitamente) inicializado:
    - Directamente quando é declarado na classe.
    - Na inicialização da respectiva classe (no caso de atributos **static**), ou na construção do respectivo objecto (no caso de atributos de instância).

Atributos (6)

- Uma constante possui os qualificadores **static final**.

```
public static final double PI = 3.141592;
```

- Um atributo **final** tem de ser sempre explicitamente inicializado. Quando não é inicializado directamente quando é declarado é dito **atributo final em branco**.

Atributos (7)

```
public class Conta{
        /* atributos */
        private static long numProxConta = 0;
        protected long numConta; // número da conta
        protected String dono;  // proprietário da conta
        protected float quantia; // saldo actual
        /* métodos */
}
```

Atributos (8)

- Um atributo de uma classe é acedido pelo operador ponto (“`.`”) na forma `referência.atributo`.
- A `referência` é um identificador de:
  - objecto, se o atributo não tiver qualificador **static**.
  - classe, se o atributo tiver qualificador **static**.

```
System.out.println(Conta.numProxConta);
```

Objectos

Sintaxe

Ident = new Classe ([Expr [, Expr]*]);

- Ident: referência para o objecto a ser criado
- Classe: classe a que pertence o dito objecto
- Expr: parâmetros a passar ao construtor

Garbage collector

- No Java, um objecto existe enquanto for referenciado.
- O ***garbage collector*** destrói objectos não referenciados.
- Se programador pretender destruir explicitamente um objecto deve:
  1. Remover todas as referências ao objecto a eliminar.
  2. Invocar o método `System.gc()`.

Construtores (1)

- Um **construtor** é um método executado na criação de objectos.
  - Têm o mesmo identificador da classe e não podem ser chamados.
  - Parâmetros são os da instrução `new`.
  - Nunca devolvem tipos, nem mesmo `void`.
  - Normalmente usados para inicializar os atributos de instância.
- Uma classe pode ter mais de um construtor.
  - O tipo e o número de argumentos passados a um construtor determinam o construtor a usar.

Construtores (2)

```
public class Conta{
        /* atributos */
        private static long numProxConta = 0;
        protected long numConta; // número da conta
        protected String dono;   // proprietário da conta
        protected float quantia; // saldo actual
        /* construtores */
        Conta() {
                numConta = numProxConta++;
        }
        Conta(String s, float q) {
                numConta = numProxConta++;
                dono = s;
                quantia = q;
        }
        /* métodos */
}
```

Construtores (3)

- Quando uma classe não define nenhum construtor (e só neste caso), o Java providencia uma **construtor por omissão** (sem argumentos).
- Um **construtor por cópia** é um construtor que recebe como argumento um objecto do mesmo tipo que o objecto que vai construir, e constrói o novo objecto com o mesmo estado do objecto recebido.
  - Normalmente, um construtor por cópia apenas atribui o valor dos atributos do objecto recebido ao objecto a ser cirado.

```
/* construtor por cópia */
Conta(Conta c) {
        numConta = c.numConta;
        dono = c.dono;
        quantia = c.quantia;
}
```

Construtores (4)

- Um construtor pode fazer uma **chamada explícita** de um outro construtor da classe através do `this()`.
- Se o construtor a chamar tiver `N` parâmetros, estes devem ser passados na chamada explícita `this(param1,...,paramN)`.
- Se existir, a chamada explícita deve ser a primeira instrução no construtor.
- Qualquer expressão que é passada como argumento ao construtor explícito não deve incluir nem atributos nem métodos do objecto a ser criado.

Construtores (5)

```
public class Conta{
        /* atributos */
        private static long numProxConta = 0;
        protected long numConta; // número da conta
        protected String dono;   // proprietário da conta
        protected float quantia; // saldo actual
        /* construtores */
        Conta() {
                numConta = numProxConta++;
        }
        Conta(String s, float q) {
                this(); /* chamada explícita */
                dono = s;
                quantia = q;
        }
        /* métodos */
}
```

Inicialização de atributos de instância (1)

- Um objecto acabado de criar tem um estado inicial:
  - Inicialização por omissão.
  - Inicialização dos atributos na declaração dos mesmos.
  - Quando é necessário mais do que uma inicialização simples:
    - **Construtores**: usados para inicializar um objecto antes da referência para o objecto ser retornada pelo `new`.
    - **Blocos de inicialização**: executados como se estivessem presentes no início dos construtores da classe.
      - Vistos como código de construção de objectos (garantia de correcção no tratamento de atributos finais em branco).
      - Úteis para definir pedaços comuns de código a executar por todos os construtores da classe.

Inicialização de atributos de instância (2)

- O construtor é chamado após:
  - Inicialização por omissão dos atributos de instância da classe.
  - Inicialização dos atributos de instância na declaração dos mesmos.

Inicialização de atributos de instância (3)

```
public class Conta{
        /* atributos */
        private static long numProxConta = 0;
        protected long numConta; // número da conta
        protected String dono;   // proprietário da conta
        protected float quantia; // saldo actual
        /* construtores */
        Conta() {
                numConta = numProxConta++;
        }
        Conta(String s, float q) {
                this();
                dono = s;
                quantia = q;
        }
        /* métodos */
}
```

Inicialização de atributos de instância (4)

```
public class Conta{
        /* atributos */
        private static long numProxConta = 0;
        { /* bloco de inicialização */
                numConta = numProxConta++;
        }
        protected long numConta; // número da conta
        protected String dono;  // proprietário da conta
        protected float quantia; // saldo actual
        /* construtores */
        Conta() {}
        Conta(String s, float q) {
                dono = s;
                quantia = q;
        }
        /* métodos */
}
```

Inicialização de atributos estáticos

- Os atributos estáticos de uma classe podem ser inicializados:
  - Na declaração dos mesmo.
  - Em **blocos de inicialização estáticos.**
    - Declarados como **static**.
    - Só podem manipular membros estáticos da classe.
- A inicialização dos atributos estáticos de uma classe é feita depois da classe ser carregada, mas antes de ser de facto usada.