Princípios de Orientação por Objectos

Objectivos

Ser capaz de:

- Descrever 'abstracção'
- Descrever conceitos de orientação por objectos: herança, encapsulamento, polimorfismo
- Identificar objectos e classes simples

Orientação por Objectos

- OO é um paradigma diferente para desenho e programação de software
- OO baseia-se na construção de modelos de objectos reais
- OO cria programas que são reutilizáveis e facilmente adaptáveis
- Os objects são autónomos e incluem informação e comportamento

O que é um Objecto?

- Definição filosófica: uma entidade que pode ser identificada
- Na terminologia
  - OO: uma abstracção de um objecto real
  - empresarial: uma entidade relevante para o domínio de aplicação
  - software: uma estrutura de dados e as funções associadas

Os Objectos executam Operações

- Um objecto existe para contribuir com funcionalidade (comportamento) a um sistema.
- Cada comportamento distinto é dado o nome de **operação**.

Objecto:
A minha caneta azul

Operação:
escrever

Objecto:
Caixa Multibanco

Operação:
levantamento

Os Objectos memorizam Valores

- Os objects têm conhecimento (informação) sobre o seu estado actual.
- Cada elemento de informação é dado o nome de **atributo**.

Objecto:
A minha caneta azul

Atributo:
Volume de tinta

Objecto:
Caixa Multibanco

Atributo:
Dinheiro levantado

Os Objectos são Abstracções

- No modelo de um objecto, apenas é necessário incluir **operações** e atributos que são importantes para o problema em questão.

Exemplo de uma operação que não interessa incluir:
- **apontar-a**

Exemplos de atributos que não interessam incluir:
- **comprimento do bico**
- **fabricante da tinta**
- **idade**

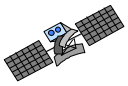
Encapsulamento

- O encapsulamento permite ocultar como as coisas funcionam e o que se sabe para além da interface—as operações de um objecto.
- Uma caixa Multibanco é um objecto que entrega dinheiro aos seus utilizadores:
  - A caixa MB encapsula isto para os seus utilizadores.
  - Violar o encapsulamento é considerado um roubo ao banco.
- Violar o encapsulamento em programação orientada por objectos é impossível.

Exercício: descobrir as operações e atributos





Hierarquias de objectos

- Os objectos são compostos por outros objectos.
- Os objectos podem fazer parte de outros objectos.
- Esta relação entre objectos é conhecida por **agregação**.

Um banco pode ser um objecto.

Uma caixa MB pode ter um teclado, leitor de cartões, dispensador de notas, todos podendo ser objectos.

Um banco pode ter uma caixa MB que também pode ser um objecto.

O que é uma Classe?

- Uma classe é uma especificação de objectos.
- Uma definição de uma classe especifica as operações e atributos para todas as instâncias de uma classe.

Quando se cria a 'minha caneta azul', não é necessário especificar as suas operações e atributos.
Basta simplesmente indicar a classe a que pertence.

Porque necessitamos de classes?

- Uma classe descreve o tipo de um objecto.
- Uma classe define o comportamento (operações) e estrutura (atributos) de um grupo de objectos:
  - Pode-se reduzir a complexidade utilizando classes.
  - No mundo existem imensos objectos, razão pela qual as pessoas os agrupam em tipos.
  - Se se compreender o tipo, pode-se aplicá-lo a vários objectos.

Como identificar uma classe?

- Identificar a estrutura e comportamento comum de um grupo de objectos.
- Identificar um único conceito coerente.
- Ambos objectos pertencem à classe Caneta.

A minha caneta azul

ops: escrever, recarregar
atribs: volume de tinta
côr da tinta

A tua caneta azul

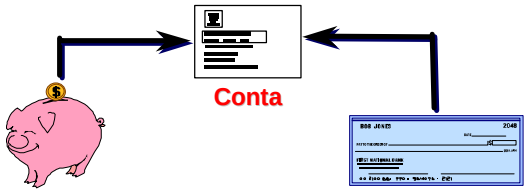
ops: escrever, recarregar
atribs: volume de tinta

Classes versus Objectos

- As classes são definições estáticas que nos permitem compreender todos os objectos de uma classe.
- Os objectos são as entidades dinâmicas que existem no mundo real e em suas simulações.
- Nota—em OO as pessoas frequentemente utilizam ambas as palavras classes e objectos de forma indiferente; é necessário utilizar o contexto para distinguir entre os dois significados possíveis.

Herança

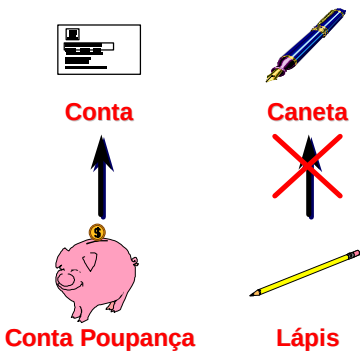
- Podem existir semelhanças entre classes distintas.
- Deve-se definir as propriedades comuns (atributos, operações) entre classes numa superclasse comum.



- As subclasses utilizam herança para incluir as propriedades comuns entre elas.

Relação "Is-a-Kind-of"

- Um objecto de uma subclasse "é-um-tipo-de" objecto de uma superclasse.
- Uma subclasse deve ter todo o comportamento da superclasse.



Polimorfismo

- O polimorfismo é a capacidade de um único nome poder referir objectos de classes diferentes, se relacionadas por uma subclasse comum
- O polimorfismo surge quando a linguagem de programação simultaneamente suporta herança e a associação dinâmica de tipos (*late binding*)

Polimorfismo...

- O polimorfismo permite que uma operação possa existir em diferentes classes.
- Cada operação tem um mesmo significado mas é executada de forma particular.

Transportar passageiros

Resumo

- Um objecto é uma abstracção de objecto real.
- Uma classe é um 'molde' ou 'fôrma' de objectos.
- As classes formam árvores de herança; as operações definidas numa classe são herdadas por todas as suas subclasses.
- O polimorfismo liberta quem invoca uma operação de conhecer a classe exacta do objecto que a irá receber.

Exercício Prático:

- Identificar classes no exemplo do 'WebCrawler II'
- Identificar métodos para as classes
- Identificar atributos para as classes
- Procurar relações de herança entre as classes

Manipulação de Classes e Objectos

Objectivos

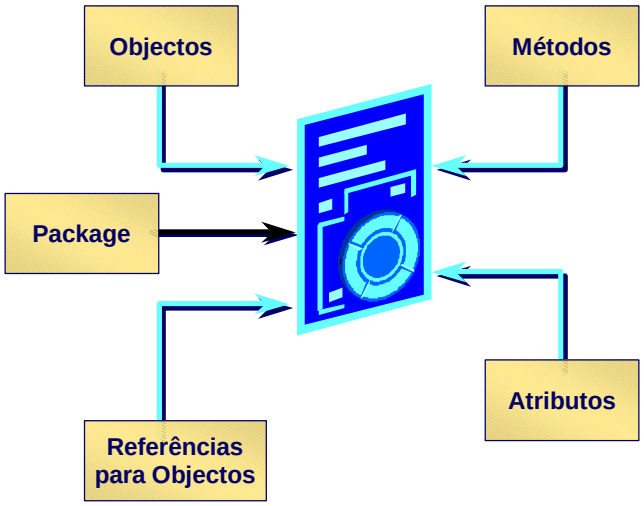
Ser capaz de:

- Utilizar packages para agrupar classes relacionadas
- Definir variáveis e métodos de instâncias
- Criar objectos e invocar métodos
- Utilizar as palavras public, private e protected
- Redefinir métodos de uma classe (overloading)
- Escrever construtores
- Utilizar variáveis e métodos de classes

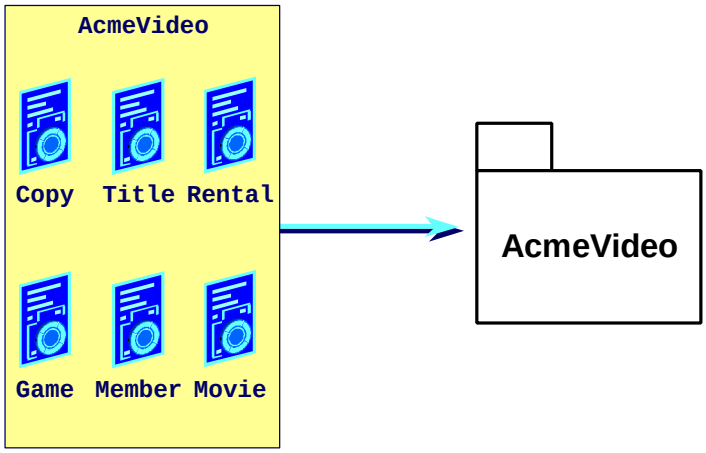
Contéudos

- As classes definem as características, atributos e comportamento dos objectos.
- Todo o código Java reside em classes.
- Toda a informação dos objectos é armazenada em variáveis.
- Os packages auxiliam a controlar o acesso a classes.
- O 'overloading' permite ter interfaces simples.
- Os construtores garantem consistência na criação de objectos.

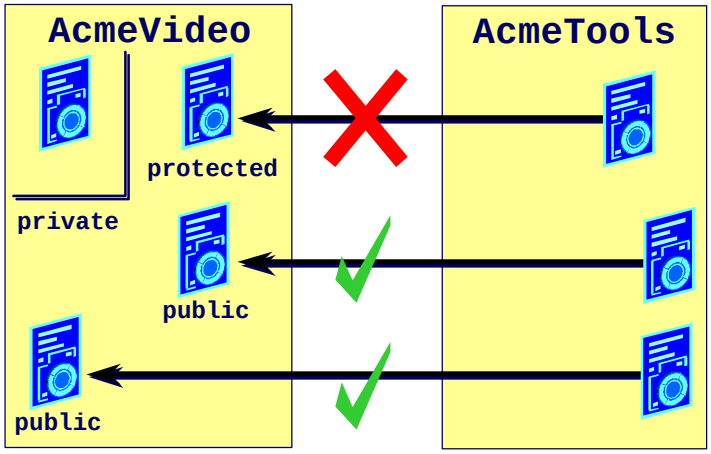
Classes Java



Packages

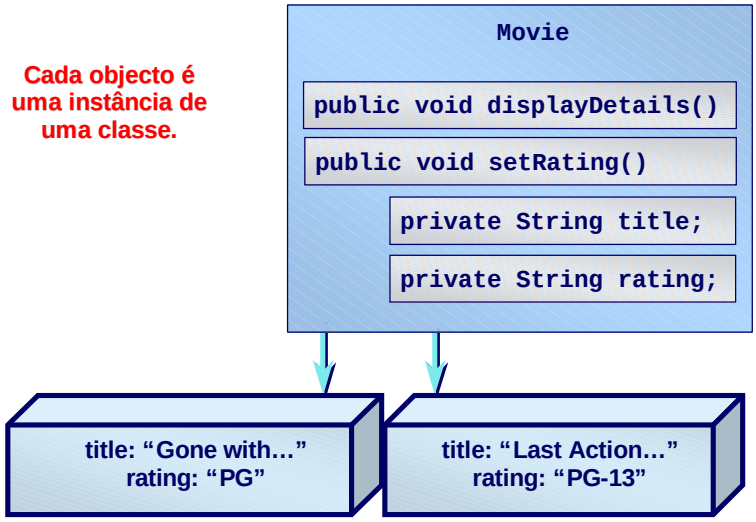


Controlo de Acesso



Classes e Objectos

Cada objecto é uma instância de uma classe.



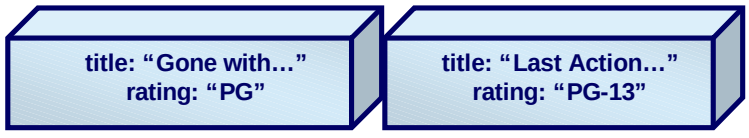
Criação de Objectos

- Os objectos são criados pelo operador new:

```
objectRef = new ClassName();
```

- Por exemplo, para criar dois objectos Movie:

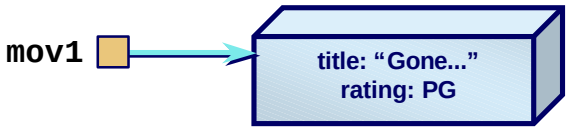
```
Movie mov1 = new Movie("Gone ...");
Movie mov2 = new Movie("Last ...");
```



new

- O operador 'new' realiza o seguinte:
  - Aloca memória para o novo objecto
  - Invoca um método especial da classe para inicialização de objectos, um constructor
  - Retorna uma referência para o novo objecto

```
Movie mov1 = new Movie("Gone...");
```



Objectos e valores primitivos

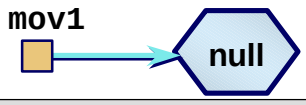
- As variáveis de tipos primitivos armazenam valores.

```
int i;
```


```
int j = 3;
```

- As variáveis de tipos de classes armazenam referências para objectos.

```
Movie mov1;
```




```
Movie mov1 = new Movie();
```

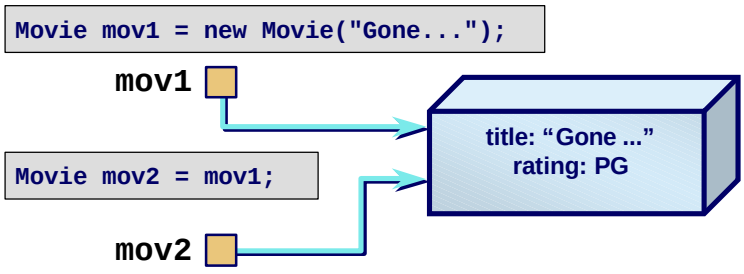
A referência null

- As referências para objectos têm o valor null até serem inicializadas.
- É possível comparar referências de objectos a null.
- Pode-se "eliminar" um objecto pela atribuição do valor null a uma referência.

```
Movie mov1 = null;  //Declare object reference
…
if (mov1 == null)    //Ref not initialized?
 mov1 = new Movie(); //Create a Movie object
…
mov1 = null;         //Forget the Movie object
```


Atribuição de Referências

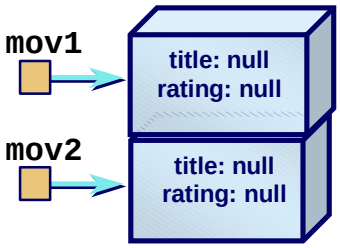
- A atribuição de uma referência a outra resulta em duas referências para o mesmo objecto:



Variáveis de instância

- As variáveis de instância são declaradas na classe:

```
public class Movie {
  public String title;
  public String rating;
…
}
```



- Criação de vários 'movie':

```
Movie mov1 = new Movie();
Movie mov2 = new Movie();
```


Acesso a variáveis de instância

- As variáveis públicas de instância podem ser acedidas através do operador '.' :

```
public class Movie {
  public String title;
  public String rating;
  …
}
```

```
Movie mov1 = new Movie();
mov1.title = "Gone ...";
…
if ( mov1.title.equals("Gone ... ") )
    mov1.rating = "PG";
```

Será isto interessante? NÃO!

Exercício:
Criar e manipular objectos

```
public class Movie {
  public String title;
}
```

```
public class MovieTest {
  public static void main(String[] args) {
    Movie mov1, mov2;
    ?
    mov1.title = "Gone with the Wind";
    mov2 = mov1;
    mov2.title = "Last Action Hero";
    System.out.println("Movie 1 is " +    ?     );
    System.out.println("Movie 2 is " +    ?     );
  }
}
```


Métodos

- Um método é equivalente a uma função ou subrotina de outras linguagens:

```
modifier returnType methodName (argumentList) {
    // method body
    …
}
```

- Um método apenas pode ser definido na definição de uma classe.

Argumentos de Métodos

```
public void setRating(String newRating) {
    rating = newRating;
  }
```

```
public void displayDetails() {
  System.out.println("Title is  " + title);
  System.out.println("Rating is " + rating);
}
```


Retorno de valores dum método

```
public class Movie {
  private String rating;
  …
  public String getRating () {
    return rating;
  }
  public void setRating (String r) {
     this.rating = r;
  }
}
```

Invocar métodos a uma instância

```
public class Movie {
  private String title, rating;
  public  String getRating(){
    return rating;
  }
  public void setRating(String newRating){
    rating = newRating;
  }
}
```

```
Movie mov1 = new Movie();
…
if (mov1.getRating().equals("G"))
  …
```

Operador '.' :

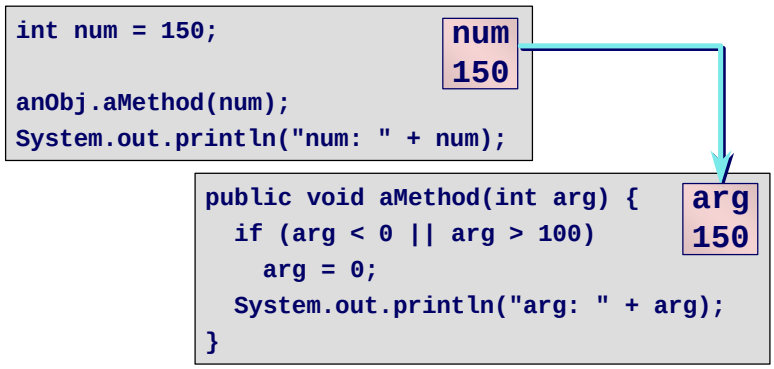
Encapsulamento

- As variáveis de instância devem ser declaradas private.
- Apenas métodos de instância podem ter acesso a variáveis de instância.
- O encapsulamento permite isolar a interface d uma classe da sua implementação interna.

```
Movie mov1 = new Movie();
…
if ( mov1.rating.equals("PG") )    // Error
     mov1.setRating("PG");         // OK
```

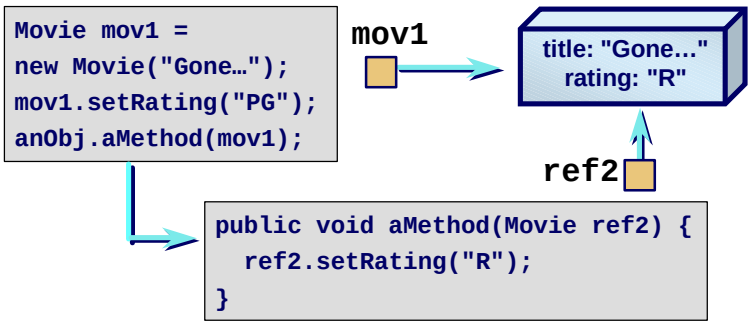
Passagem de valores a métodos

- Quando um valor primitivo é passado a um método, é gerada uma cópia do valor:



Passagem de objectos a métodos

- Quando um objecto é passado a um método, o argumento refere o objecto original:



'Overloading' de métodos

- Diversos métodos de uma classe podem ter o mesmo nome.
- Os métodos devem ter diferentes assinaturas.

```
public class Movie {
  public void setPrice() {
    price = 3.50;
  }
  public void setPrice(float newPrice) {
    price = newPrice;
  } …
}
```

```
Movie mov1 = new Movie();
mov1.setPrice();
mov1.setPrice(3.25);
```


Inicialização de atributos

- As variáveis de instância podem ser inicializadas na sua declaração.

```
public class Movie {
  private String title;
  private String rating = "G";
  private int numOfOscars = 0;
```

- A inicialização é feita na criação do objecto.
- Inicializações mais complexas devem ser colocadas num método construtor.

Construtores

- Para uma inicialização adequada, a classe deve fornecer construtores.
- O construtor é invocado automaticamente quando o objecto é criado:
  - Normalmente declarado 'public'
  - Tem o mesmo nome da classe
  - Não especifica nenhum tipo de retorno
- O compilador automaticamente fornece um construtor por defeito sem argumentos.

Definição de Construtores

```
public class Movie {
  private String title;
  private String rating = "PG";

  public Movie() {
    title = "Last Action …";
  }
  public Movie(String newTitle) {
    title = newTitle;
  }
}
```

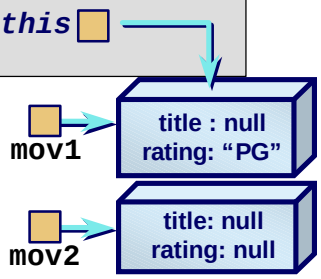
A classe Movie fornece dois construtores

```
Movie mov1 = new Movie();
Movie mov2 = new Movie("Gone …");
Movie mov3 = new Movie("The Good …");
```

A referência 'this'

- Os métodos de instância recebem um argumento com o nome 'this', que refere para o objecto corrente.

```
public class Movie {
  public void setRating(String newRating) {
    this.rating = newRating;
  }
```




```
void anyMethod() {
 Movie mov1 = new Movie();
 Movie mov2 = new Movie();
 mov1.setRating("PG"); …
```


Partilha de código entre construtores

```
public class Movie {
  private String title;
  private String rating;

  public Movie() {
    this("G");
  }
  public Movie(String newRating) {
    rating = newRating;
  }
}
```

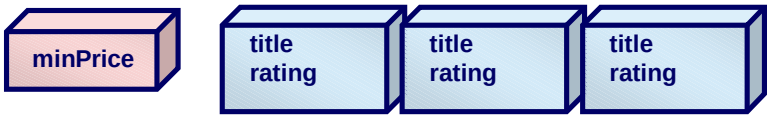
Um construtor pode invocar outro através de this()

```
Movie mov2 = new Movie();
```

Variáveis de Classe

- As variáveis de classe pertencem a uma classe e são comuns a todas as instâncias dessa classe.
- As variáveis de classe são declaradas como 'static' na definição da classe.

```
public class Movie {
  private static double minPrice;   // class var
  private String title, rating;     // inst vars
```



classe Movie objectos Movie

Inicialização de variáveis de classe

- As variáveis de classe podem ser inicializadas na declaração.
- A inicialização é realizada quando a classe é carregada.

```
public class Movie {
  private static double minPrice = 1.29;

  private String title, rating;
  private int length = 0;
```

Métodos de Classe

- Os métodos de classe são partilhados por todas as instâncias.
- São úteis para manipular variáveis de classe:

```
public static void increaseMinPrice(double inc) {
  minPrice += inc;
}
```

- Um método de classe pode ser invocado utilizando o nome da classe ou uma referência para um objecto.

```
Movie.increaseMinPrice(.50);
mov1.increaseMinPrice(.50);
```


Exercício: métodos de classe ou de instância?

```
public class Movie {

 private static float price = 3.50f;
 private String rating;
  …
 public static void setPrice(float newPrice) {
    price = newPrice;
 }
 public float getPrice() {
    return price;
 }
}
```

```
Movie.setPrice(3.98f);
Movie mov1 = new Movie(…);
mov1.setPrice(3.98f);
float a = Movie.getPrice();
float b = mov1.getPrice();
```

Exemplos de Java

- Exemplos de métodos e variáveis 'static':
  - main()
  - Math.sqrt()
  - System.out.println()

```
public class MyClass {

  public static void main(String[] args) {
    double num, root;
    …
    root = Math.sqrt(num);
    System.out.println("Root is " + root);
  } …
```


Variáveis final

- Uma variável declarada 'final' é uma constante.
- Uma variável 'final' não pode ser modificada.
- Uma variável 'final' deve ser inicializada.
- Uma variável 'final' é normalmente pública para permitir acesso externo.

```
public final class Color {

  public final static Color black=new Color(0,0,0);
  …
}
```

Garbage Collection

- Quando todas as referência para um objecto são eliminadas, o objecto é marcado para ser destruído.
  - Garbage collection liberta a memória utilizada pelo objecto.
- Garbage collection é automática.
  - Não existe necessidad de intervenção do programador, mas não possui qualquer controlo sobre quando o objecto é realmente destruído

O método finalize()

- Se um objecto utilizar um outro recurso (p.e. Um ficheiro), o objecto deve libertá-lo.
- Pode ser fornecido um método finalize().
- O método finalize() é invocado antes do objecto ser destruído.

```
public class Movie {
  …
  public void finalize() {
    System.out.println("Goodbye");
  }
}
```

Resumo

- A definição de uma classe especifica as características comuns de um conjunto de objectos.
- Um objecto é uma instância de uma classe particular:
  - Criam-se objectos através do operador 'new'.
  - Manipula-se um objecto através da invocação de métodos públicos de instância.
- Os métodos de instância recebem a referência 'this'
- Os métodos podem ter diferentes implementações
- As classes fornecem um ou mais construtores para inicializar objectos.
- Podem ser definidos variáveis e métodos para implementar comportamentos globais à classe.