

Programação com Objectos

**Material didáctico elaborado pelas diferentes equipas de
Introdução à Programação**

Luís Caires (Responsável), Armanda Rodrigues, António Ravara, Carla Ferreira, Fernanda Barbosa, Fernando Birra, Jácome Cunha, João Araújo, Miguel Goulão, Miguel Pessoa Monteiro, e Sofia Cavaco.

Mestrado Integrado em Engenharia Informática FCT UNL

Como é constituído o software?

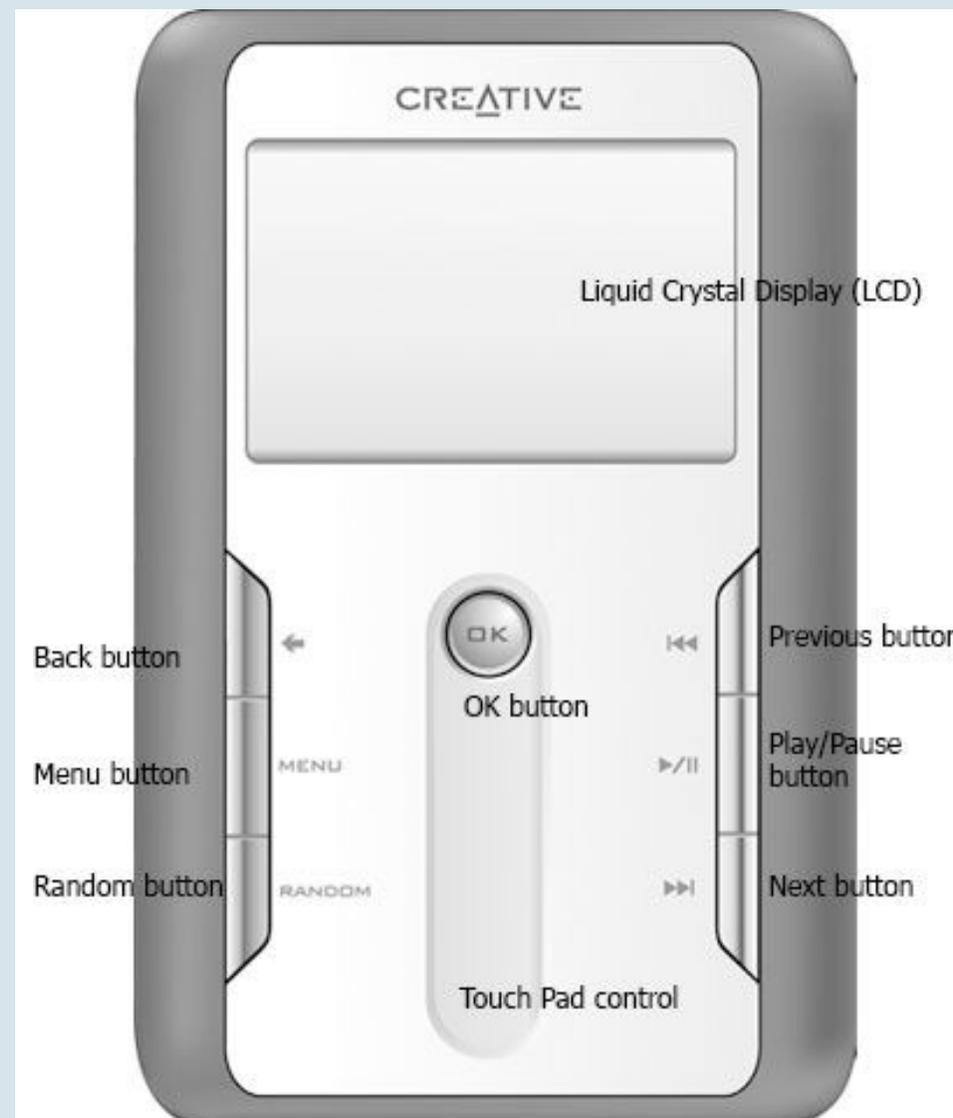
- As aplicações informáticas (software) são, em geral, sistemas bastante complexos.
- Um programa descreve uma tarefa particular
 - ordenar uma lista de números
 - resolver um sistema de equações
 - visualizar uma fotografia
 - ...
- Uma aplicação realiza certas funcionalidades
 - edição de texto
 - sistema de base de dados
 - editor gráfico.

Objectos

- Na sua maioria, os sistemas de software são organizados com base em componentes elementares, chamados **objectos**.
- No mundo real, um objecto é uma entidade física autónoma que desempenha uma determinada finalidade:
 - caneta
 - terminal multibanco
 - televisor
 - leitor MP3
 - automóvel

Objectos Físicos (analogia)

- Qualquer objecto físico suporta um conjunto de operações particulares, que lhe permitem ser utilizado (ou seja manipular o seu estado) pelo utilizador interessado.

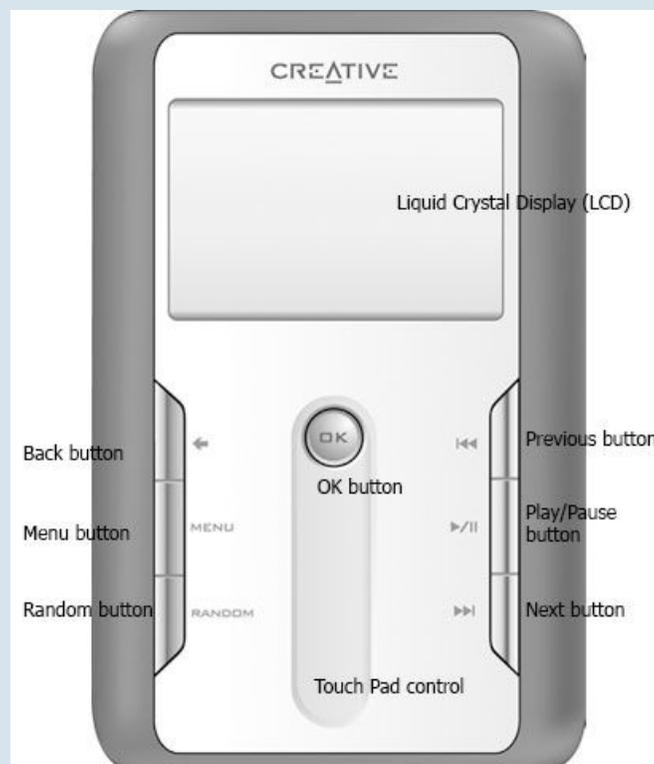


Objectos (Programação)

- Actualmente, os sistemas de software são organizados com base em componentes elementares, chamados **objectos**.
- No **mundo virtual** da programação, um objecto é uma entidade autónoma que desempenha uma determinada finalidade:
 - ficheiro
 - contador
 - conta bancária
 - agenda electrónica
 - base de dados

Interface do MP3

- Os utilizadores interagem com os objectos através da sua interface.
- Tecnicamente, chama-se **interface** à lista das operações suportadas por um objecto :



Interface do MP3:

- ▶ Previous
- ▶ Play
- ▶ Next
- ▶ Back
- ▶ Ok
- ▶ Playing?
- ▶ Memory?

Interface do MP3

- Algumas das operações são acções que podem ser aplicadas ao objecto, alterando o seu estado. Por exemplo,
 - **Next**, muda para a música seguinte;
 - **Stop**, para a música corrente;
- Outras operações permitem apenas consultar o estado do aparelho;
 - **Playing?**, mostra no écran a música corrente;
 - **Memory?**, mostra no écran a memória livre;
- Em geral, todos os objectos apresentam estes dois tipos de operações na sua interface:
 - Operações **Modificadoras** (*mutators*)
 - Operações de **Consulta** (*accessors*)

O primeiro objecto (de software)

O primeiro objecto (de software)

- **Objectivo**

- Simular uma lâmpada.

- **Descrição**

- Uma lâmpada pode estar acesa ou apagada.

- **Funcionalidades**

- O estado da lâmpada pode mudar através das operações de “on” (acesa) e “off” (apagada).
- Deve sempre ser possível verificar se a lâmpada se encontra acesa com a operação “isOn”.
- Quando a lâmpada é criada o seu estado é “apagado”.

- **Interacção com o utilizador**

- Após criar uma lâmpada, pode invocar as operações da lâmpada.

O primeiro objecto (de software)

- **Qual o objecto a definir?**

- Um objecto (de software) que representa uma lâmpada, que tem um **estado** (“aceso” ou “apagado”).



- **Como escrever a interface do objecto em Java?**

- As operações são funções $f: D \rightarrow C$, que em Java se escrevem da forma:

C $f(D)$

- Operação constante (D vazio), em Java escreve-se:

C $f()$

- Operação que não devolve nada (C vazio), em Java escreve-se:

`void f(D)`

O primeiro objecto (de software)

- **Interface do objecto** (três operações):

`void on()`

coloca a lâmpada no **estado** “aceso”

`void off()`

coloca a lâmpada no **estado** “apagado”

`boolean isOn()`

indica se o **estado** corrente da lâmpada é “aceso”

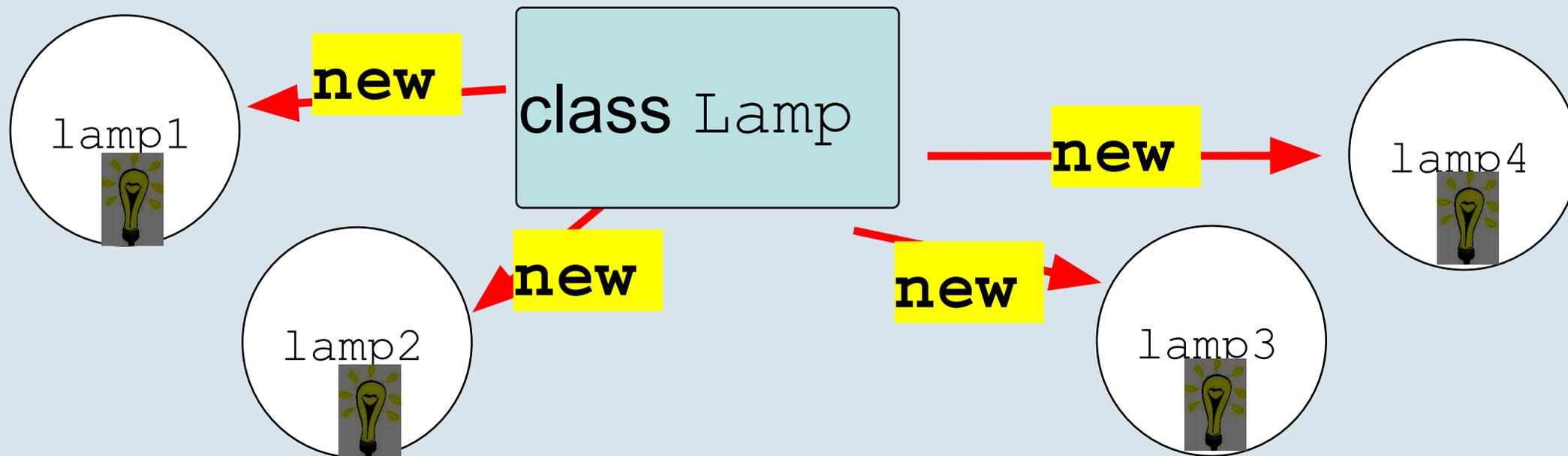


- As operações “on” e “off” são modificadores e não devolvem nada (`void`);
- A operação “isOn” é de consulta (devolve `true`, se estado aceso, e `false`, caso contrário – `boolean`).

A “vida” de um objecto

- Como são fabricados os objectos de software?
 - Os objectos são criados por componentes de software chamados “classes” (veremos adiante)
 - Classes funcionam como “fábricas” de objectos
 - Todos os objectos criados por uma classe são semelhantes à nascença:
 - apresentam a mesma interface
 - executam as mesmas operações
 - podem ter evoluções diferentes
 - as operações de um não “mexem” com o outro

A “vida” de um objecto

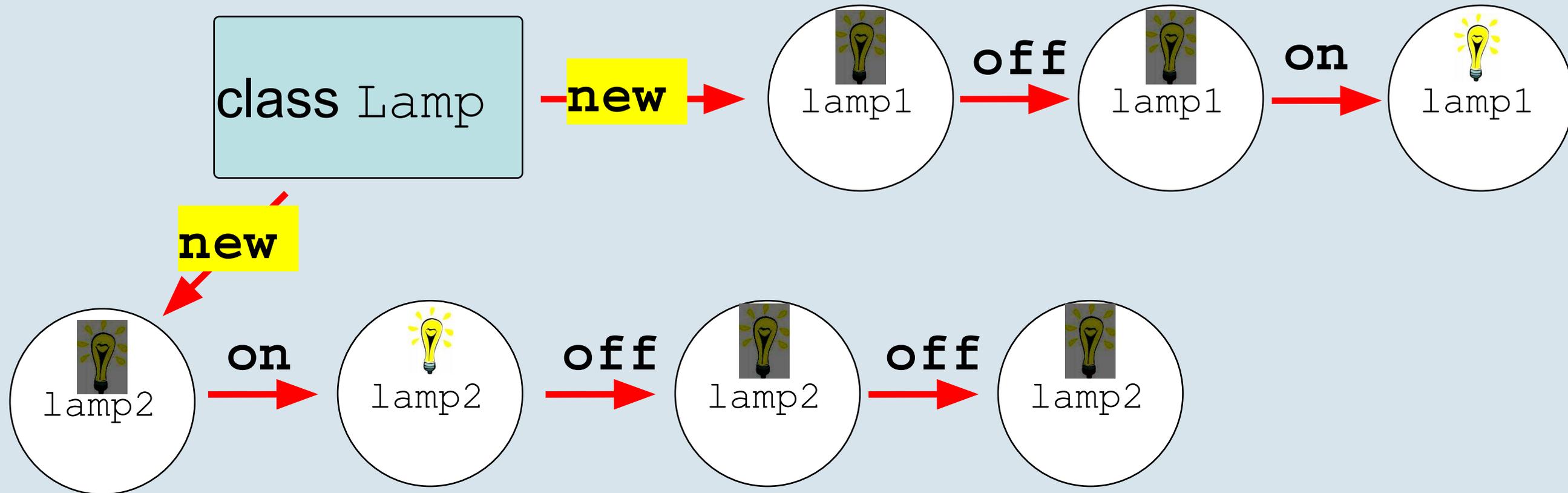


- Em Java, se `Lamp` for o nome de uma classe de objectos, então a expressão “`new Lamp ()`” representa a criação de um novo objecto da classe `Lamp`:

```
Lamp lamp1 = new Lamp ();
```

“`lamp1`” é o nome do novo objecto da classe `Lamp` criado pela expressão `new Lamp ()`, ou como se diz em Java, a sua “referência”. Usa-se este nome para aceder ao objecto no contexto de um programa.

A “vida” de um objecto



- Cada objecto tem a sua **vida própria**
- Cada objecto tem o seu **estado próprio**
- Cada objecto tem uma **identidade própria**:
 - Que é um código único, gerado pelo sistema
- Cada objecto **evolui autonomamente**

Demo

- Interacção com objectos da classe Lamp

Comportamento esperado da lâmpada

```
Lamp lamp1 = new Lamp();  
lamp1.isOn();  
// false (boolean)  
lamp1.on();  
lamp1.isOn();  
// true (boolean)  
lamp1.on();  
lamp1.isOn();  
// true (boolean)  
lamp1.off();  
lamp1.isOn();  
// false (boolean)
```



Criação de uma
nova lâmpada

Comportamento esperado da lâmpada

```
Lamp lamp1 = new Lamp();  
lamp1.isOn();  
// false (boolean)  
lamp1.on();  
lamp1.isOn();  
// true (boolean)  
lamp1.on();  
lamp1.isOn();  
// true (boolean)  
lamp1.off();  
lamp1.isOn();  
// false (boolean)
```

Invocação da operação `isOn`

Resposta da operação `isOn`

Nota: o resultado devolvido pelas operações de consulta é apresentado a **verde**

Comportamento esperado da lâmpada

```
Lamp lamp1 = new Lamp();  
lamp1.isOn();  
// false (boolean)  
lamp1.on();  
lamp1.isOn();  
// true (boolean)  
lamp1.on();  
lamp1.isOn();  
// true (boolean)  
lamp1.off();  
lamp1.isOn();  
// false (boolean)
```

Invocação da operação `on`

A operação `on` não tem resposta (limita-se a alterar o estado do objecto lâmpada)

Nota: as operações `on` e `off` são **modificadores**

Definição de objectos em Java

- A programação em Java consiste, em larga medida, na definição de novos objectos.
- Cada objecto disponibiliza um conjunto de operações, o programador tem que definir com rigor o efeito de cada uma dessas operações.
- Cada operação de um objecto é definida por um “pequeno programa” (em geral, com menos de 10 linhas de texto).
- No contexto da programação, esses pequenos programas, que definem as operações dos objectos, chamam-se **métodos**.
- Para além dos métodos, cada objecto possui uma **memória própria privada**, que estes podem utilizar.

Definição de classes em Java

- A programação em Java consiste, em larga medida, na definição de novos objectos.
- No entanto, não é prático definir cada objecto de forma independente.
- Assim, a linguagem Java permite a definição de **classes**, e não de objectos directamente.
- Uma classe é uma fábrica inesgotável de objectos do mesmo tipo.
- Exemplo: uma vez definida a classe `Lamp`, podemos criar os objectos da classe `Lamp` que quisermos:

```
Lamp mylamp1 = new Lamp ();
```

```
Lamp mylamp2 = new Lamp ();
```

```
...
```

Comportamento esperado da lâmpada

```
Lamp mylamp1 = new Lamp();  
Lamp mylamp2 = new Lamp();  
mylamp1.isOn();  
// false (boolean)  
mylamp2.on();  
mylamp1.isOn();  
// false (boolean)  
mylamp1.on();  
mylamp2.isOn();  
// true (boolean)  
mylamp1.isOn();  
// true (boolean)
```

As operações efectuadas entre objectos diferentes da mesma classe não interferem umas com as outras

Programando a classe Lamp



James Gosling
Criador da linguagem Java

Programando a classe `Lamp` em Java

- Programar em Java consiste em definir uma ou mais classes de objectos.
- Definir uma classe em Java consiste em:
 - Escrever a sua especificação rigorosa, usando um editor de texto (integrado no IDE Eclipse).
- A linguagem Java, como qualquer linguagem de programação obedece a regras muito precisas, que é preciso conhecer com rigor.
- Nos slides seguintes, vamos introduzir passo a passo os vários elementos presentes na definição de uma classe Java usando como exemplo a classe de objectos `Lamp`.

Construção de um Programa

Texto do Programa
Lamp.java

```
public class Lamp{  
    private boolean lampIsOn;  
    public Lamp()    {  
        lampIsOn = false;  
    }  
}
```

criado com um editor de texto

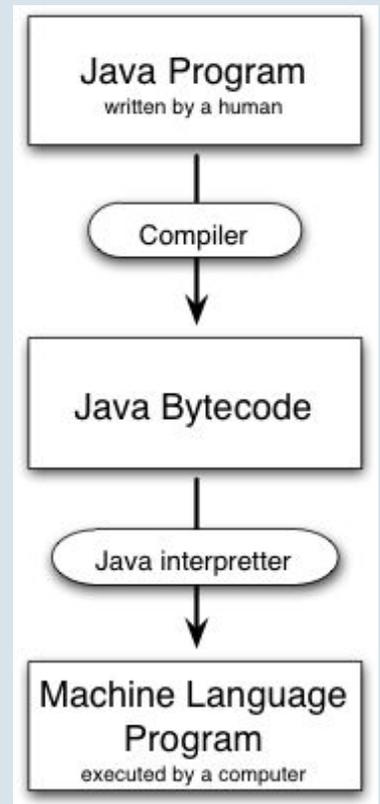
Compilador
(javac)

o compilador **javac** é uma aplicação que traduz o texto do nosso programa em código máquina

computador com uma máquina java instalada

Java Bytecode
Lamp.class

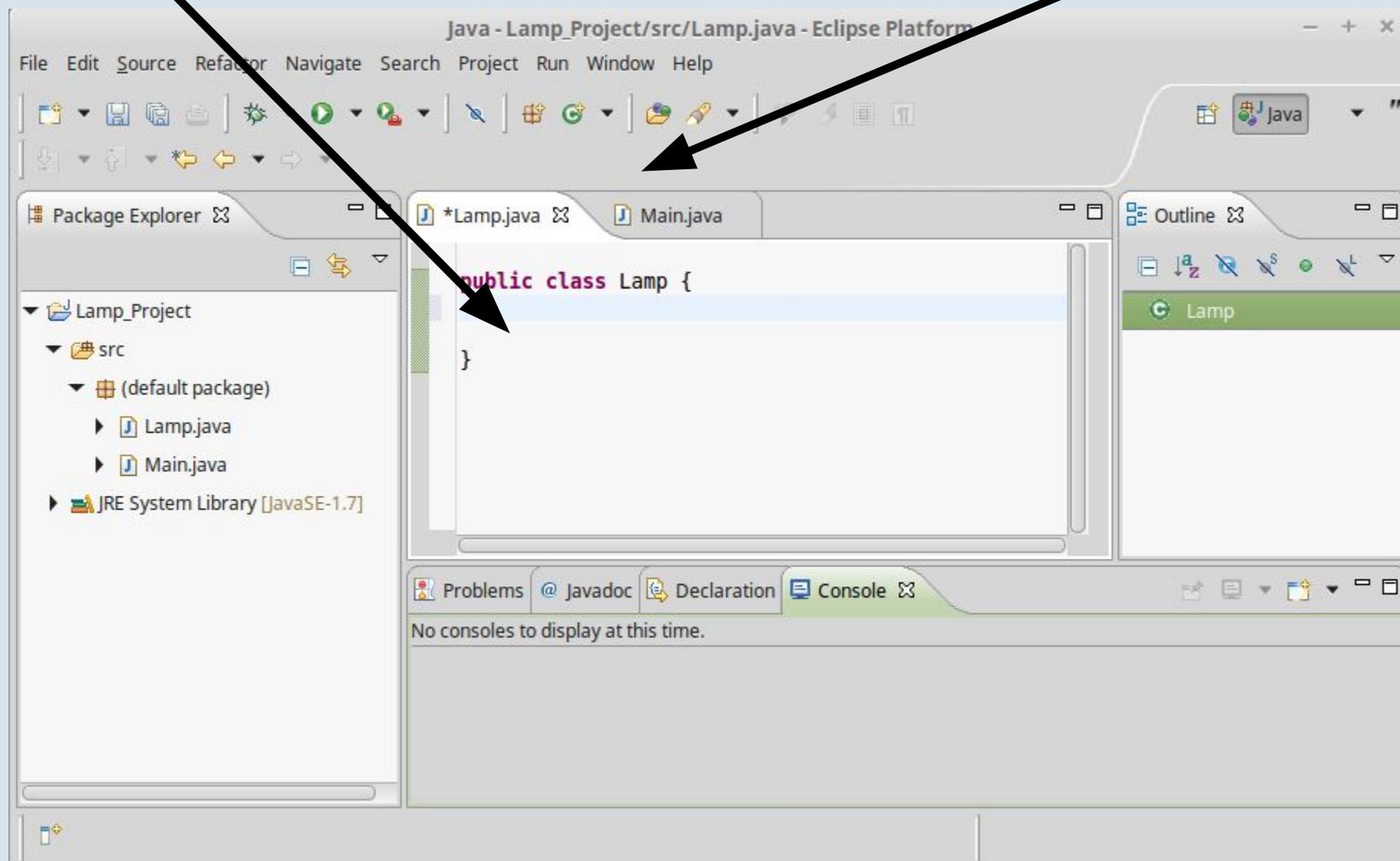
```
1001001001001110  
1001001110100100  
1110100100111010  
0100111010010011  
1010010011101001  
0011101001001110  
1001001110100100  
1110100100111010  
010011101001
```



O IDE Eclipse

Texto do Programa
Lamp.java

Classes do Programa



Classe Lamp

(geradora de objectos “lâmpada”)

Aspecto geral da classe Lamp:

Nome da classe

```
public class Lamp {  
    private boolean lampIsOn ;  
    public Lamp () { lampIsOn = false; }  
    public void on () { lampIsOn = true; }  
    public void off () { lampIsOn = false; }  
    public boolean isOn () { return lampIsOn; }  
}
```

Definição das operações dos objectos da classe Lamp

Classe `Main`

(Onde se criam e testam objectos “lâmpada”)

Relembrando:

- Classes são "moldes" de objectos; "saem" iguais mas cada indivíduo é distinto.
- Como são criados? com um comando específico: `new NomeDaClasse (argumentos)`
- Onde são criados: no corpo de métodos de outra classe.

O início de qualquer aplicação em Java:

- Se a criação de objetos de uma classe é feita (em geral) no corpo de um método de outra classe, onde se criam objectos dessa classe?
- Em Java, todas as aplicações têm um ponto de entrada pré-definido: um método denominado `main`, a partir do qual serão invocados todos os outros métodos necessários à aplicação.
- Normalmente criamos uma classe de topo na nossa aplicação, denominada `Main`, onde definimos o método `main`.

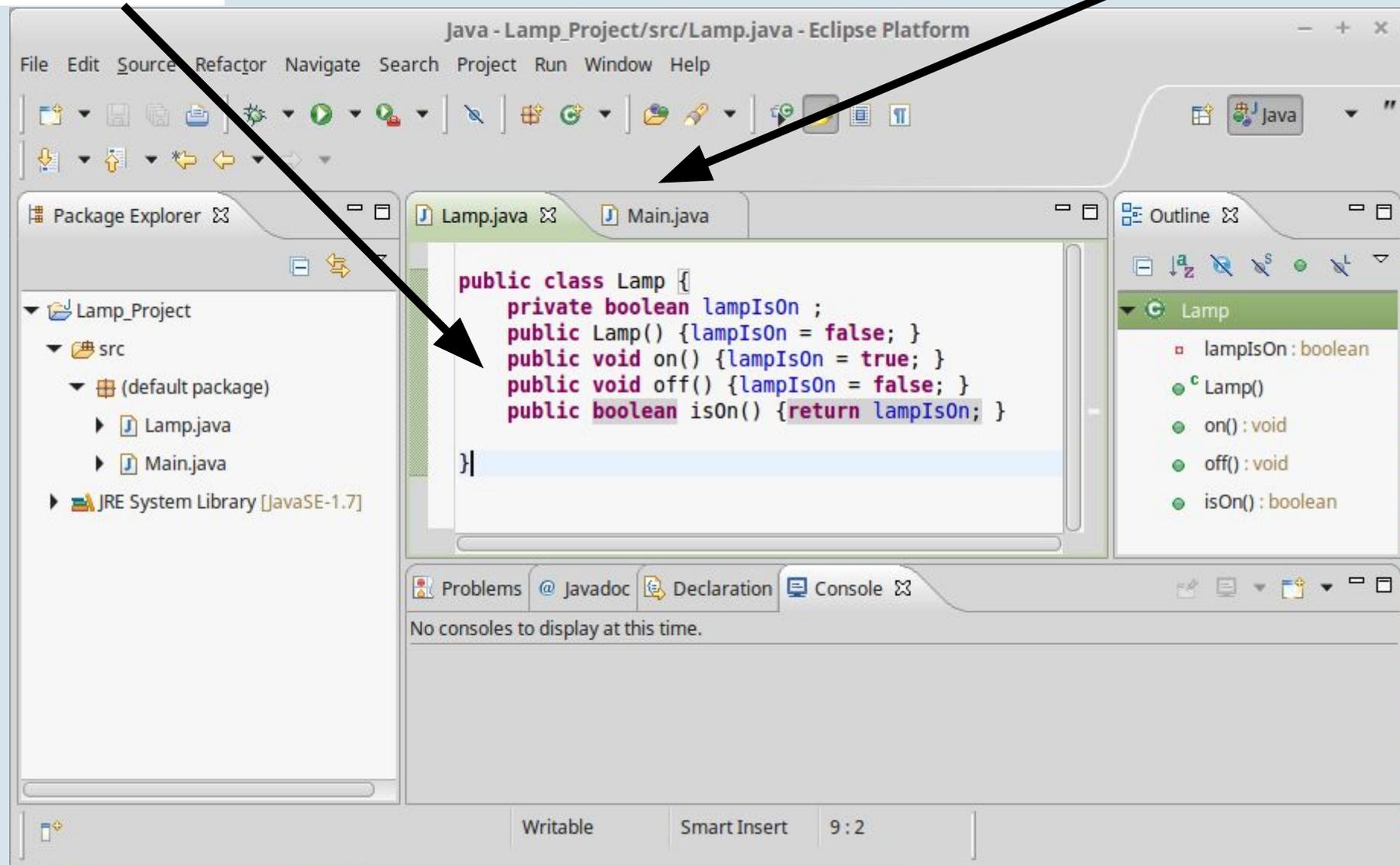
A classe `Main`, muito brevemente (mais pormenores mais tarde):

- Define o modo como a aplicação vai ser executada.
- Não se criam objectos da classe `Main` - o compilador executa sempre primeiro o seu método `main`.
- No método `main` são criados e usados os objectos de quaisquer outras classes (já definidas), para realizar as tarefas da aplicação.

O IDE Eclipse

Texto do Programa
Lamp.java

Classes do Programa



O IDE Eclipse

Texto do Programa
Main.java

Classes do Programa

The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows the project structure: Lamp_Project > src > (default package) > Lamp.java and Main.java. The main editor window shows the code for Main.java:

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    Lamp lamp1 = new Lamp();  
    System.out.println(lamp1.isOn());  
    lamp1.on();  
    System.out.println(lamp1.isOn());  
    lamp1.on();  
    System.out.println(lamp1.isOn());  
    lamp1.off();  
    System.out.println(lamp1.isOn());  
}
```

The Outline view on the right shows the class Main with the main method selected. The Console view at the bottom shows the output of the program:

```
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-i386/bin/java (Sep 6, 2016 9:34:35 PM)  
false  
true  
true  
false
```

Arrows from the text boxes point to the corresponding elements in the IDE: the code editor, the Package Explorer, the Outline view, and the Console view.

Escrita de valores na
consola (
“System.out.println”).

Interacção com objectos.
Teste à classe Lamp (feito na classe Main)

Classe Lamp

(geradora de objectos “lâmpada”)

A especificação de uma classe em Java tem sempre esta forma:

Nome da classe

```
public class ????
```

Aqui: Definição das variáveis e do construtor... veremos mais adiante

Aqui: Definição das operações (chamadas tecnicamente “métodos”) dos objectos da classe `Lamp`.

Declaramos a classe como pública (`public`), para podermos aceder a ela livremente.

A ordem de declaração dos métodos deve facilitar a compreensão da classe por parte do programador

Classe Lamp

(geradora de objectos “lâmpada”)

Nome da classe



```
public class Lamp {
```

O nome da classe é um **identificador** qualquer, escolhido pelo programador (por si!)

Para identificador da classe use um nome (substantivo), iniciado por uma maiúscula

```
}
```

Identificadores (nomes) em Java

- Nos programas é necessário dar nomes a vários ingredientes (operações, classes, variáveis, etc).
- Os **identificadores** são escolhidos livremente pelo programador, e devem ser bons nomes para as coisas que pretendem identificar.
- Os identificadores na linguagem Java
 - Podem incluir letras, dígitos e o carácter “_”
 - Não podem começar por um dígito
 - Não podem conter espaços
 - Não podem ser palavras já reservadas pela linguagem Java (por exemplo, `class`, `int`, etc.).

Identificadores (nomes) em Java

- Identificadores válidos:
 - `mySpace`
 - `BladeRunner`
 - `this_is_an_identifier`
 - `agent007`
- Sequências de caracteres que **não são identificadores**:
 - `0thesis` (começa por um dígito)
 - `My My` (contém um espaço em branco)
 - `class` (palavra já reservada pela linguagem)
 - `What?` (contém um caracter especial “?”)

Classe Lamp (definição dos métodos)

Cada declaração de método indica o programa associado a uma certa operação da interface do objecto

```
public class Lamp {
```

```
    public void on () { ... }
```

```
    public void off () { ... }
```

```
    public boolean isOn () { ... }
```

```
}
```

Métodos

Declaramos os métodos como públicos (**public**), para podermos usá-los livremente.

Classe Lamp (definição dos métodos)

Cada declaração de método indica as instruções associadas a uma certa operação da interface do objecto

As instruções associadas ao método

`public class Lamp { on serão escritas aqui entre as { ... }`

```
public void on () { ... }
```

```
public void off () { ... }
```

```
public boolean isOn () { ... }
```

```
}
```

Declaração do método `on`

Classe Lamp (definição dos métodos)

Cada declaração de método indica as instruções associadas a uma certa operação da interface do objecto

As instruções associadas ao método

`public class Lamp { off serão escritas aqui entre as { ... }`

```
public void on() { ... }
```

```
public void off() { ... }
```

```
public boolean isOn() { ... }
```

```
}
```

Declaração do método `off`

Classe Lamp (definição dos métodos)

Cada declaração de método indica as instruções associadas a uma certa operação da interface do objecto

As instruções associadas ao método `isOn` serão escritas aqui entre as `{...}`

```
public class Lamp {  
  
    public void on ()    { ... }  
    public void off ()  { ... }  
    public boolean isOn () { ... }  
  
}
```

Declaração do método `isOn`

Classe Lamp

O estado da lâmpada, resultante da última operação modificadora efectuada, terá que ser guardado em **memória**

Vamos guardar o estado da lâmpada numa **variável**, à qual vamos dar o nome `lampIsOn`

```
public class Lamp {  
  
    public void on ()    { ... }  
    public void off ()  { ... }  
    public boolean isOn () { ... }  
  
}
```

Variáveis e Memória

- Uma variável em Java designa uma zona na memória do computador que pode guardar um valor de certo tipo.
- As variáveis são referidas pelo seu nome (identificador).
- Quando um objecto é criado, devemos assumir que todas as suas variáveis contêm um valor desconhecido.
- Para definir (ou substituir) o valor guardado numa variável, podemos usar a **instrução de afectação** (também conhecida como **atribuição**).

```
public void on () { lampIsOn = true; }  
public void off () { lampIsOn = false; }
```

atribuição do valor **true** à variável `lampIsOn`

Variáveis e Memória

- Uma variável em Java designa uma zona na memória do computador que pode guardar um valor de certo tipo.
- As variáveis são referidas pelo seu nome (identificador).
- Quando um objecto é criado, devemos assumir que todas as suas variáveis contêm um valor desconhecido.
- Para definir (ou substituir) o valor guardado numa variável, podemos usar a **instrução de afectação** (também conhecida como **atribuição**).

```
public void on () { lampIsOn = true; }  
public void off () { lampIsOn = false; }
```

atribuição do
valor **false** à
variável
lampIsOn

Memória de um Objecto

```
public class Lamp {  
    private boolean lampIsOn ;  
}
```

“memória” de cada objecto

A memória própria de cada objecto é especificada por um conjunto de **declarações de variáveis de instância**

Em Java, as declarações de variáveis de instância escrevem-se sempre na forma seguinte:

```
Acesso DescritorDeTipo Identificador ; //recomendada  
Acesso DescritorDeTipo Identificador = ValorInicial ;//não recomendada
```

}

Declaração de Variáveis

Modificador de acesso `private`. As variáveis de instância **não devem ser acessíveis fora da classe.**

```
public class Lamp {  
    private boolean lampIsOn ;  
}
```

DescritorDeTipo (boolean)

Identificador (lampIsOn)

A memória própria de cada objecto é especificada por um conjunto de **declarações de variáveis de instância**

Em Java, as declarações de variáveis de instância escrevem-se sempre na forma seguinte:

```
Acesso DescritorDeTipo Identificador ;  
private boolean lampIsOn ;
```

Identificador na Declaração de Variável

O *Identificador* dá o nome à variável.
Esse nome é escolhido livremente pelo programador!

Identificador (lampIsOn)

```
public class Lamp {  
    private boolean lampIsOn ;  
}
```

A memória própria de cada objecto é especificada por um conjunto de **declarações de variáveis de instância**

Em Java, as declarações de variáveis de instância escrevem-se sempre na forma seguinte:

```
Acesso DescritorDeTipo Identificador ;  
private boolean lampIsOn ;
```

Identificador na Declaração de Variável

O *Identificador* dá o nome à variável.
Esse nome é escolhido livremente pelo programador!

Identificador (lampIsOn)

```
public class Lamp {  
    private boolean lampIsOn ;  
}
```

Para nome da variável use um nome (substantivo),
iniciado por uma minúscula, e.g. lampIsOn

Em Java, as declarações de variáveis de instância
escrevem-se sempre na forma seguinte:

Acesso DescritorDeTipo Identificador ;
private boolean lampIsOn ;

Descritor de Tipo na Declaração de Variável

DescritorDeTipo (**boolean**)

O *DescritorDeTipo* indica que valores podem ser guardados na variável

```
public class Lamp {
```



```
private boolean lampIsOn ;
```

A memória própria de cada objecto é especificada por um conjunto de **declarações de variáveis de instância**

Em Java, as declarações de variáveis de instância escrevem-se sempre na forma seguinte:

Acesso DescritorDeTipo Identificador ;
private boolean lampIsOn ;

Tipos de dados em Java

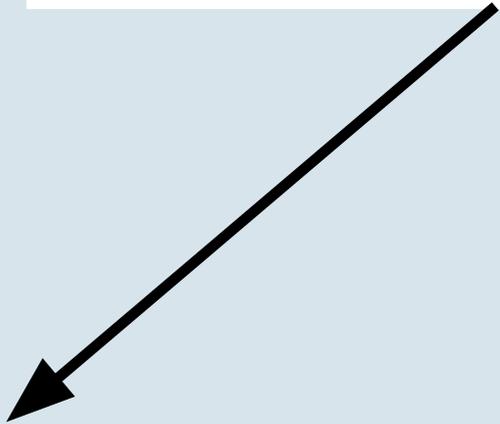
- A linguagem Java suporta muitos **tipos de dados**
- Um tipo é um conjunto de valores da mesma natureza .
- Para já, vamos conhecer os tipos **boolean** e **int** :
- O tipo **boolean**
 - É o tipo dos valores lógicos (booleanos).
 - Só existem dois valores booleanos: **false** e **true**.
- O tipo **int**
 - É o tipo dos naturais positivos e negativos (inteiros).
 - Existem muitos valores inteiros : **0**, **1**, **-10**, **1000**, etc.
 - Em Java, os inteiros são limitados ao que é possível representar em 4 bytes $[-2.147.483.648, 2.147.483.647]$, ou seja, $[-2^{31}, 2^{31}-1]$.

Definição do método `isOn`

O estado da lâmpada, resultante da última operação modificadora efectuada, é guardado na **variável** `lampIsOn`:
`true` -> aceso; `false` -> apagado

```
public class Lamp {  
    private boolean lampIsOn ;  
    ...  
    public void on() { ... }  
    public void off() { ... }  
    public boolean isOn() { return lampIsOn; }  
}
```

Instruções
associadas ao
método `isOn`



A instrução “**return** `lampIsOn`” devolve o valor da
variável `lampIsOn` como resultado do método `isOn`

A instrução “return”

return *Expressão*

- Para devolver o resultado de uma operação de um objecto é necessário usar a **instrução return**.
- A forma geral da instrução **return** é

```
return Expressão
```

Expressão é uma composição de constantes e/ou variáveis, envolvendo eventualmente outras operações, que calcula o valor a devolver como resultado do método.

- Depois de executar a instrução

```
return Expressão
```

O conjunto de instruções no corpo do método **termina**, devolvendo o valor retornado ao utilizador.

- O valor da *Expressão* deve ser do mesmo tipo que o da operação.

Definição do método `isOn`

DescritorDeTipo (`boolean`)
Indica que o método devolve um valor booleano

Lista de argumentos vazia (`()`)
indica que o método não recebe dados de entrada

```
public boolean isOn () { return lampIsOn; }
```

Identificador: `isOn`
Indica o nome do método

Corpo: `return lampIsOn;`
Define o programa que implementa o método `isOn`
`return lampIsOn;`

Definição do método `on`

DescritorDeTipo (`void`)
Indica que o método não devolve nenhum resultado

Lista de argumentos vazia (`()`)
indica que o método não recebe dados de entrada

```
public void on () { lampIsOn = true; }
```

Identificador : `on`
Indica o nome do método

Corpo : `lampIsOn = true;`
Define o programa que implementa o método `on`
`lampIsOn = true;`

Definição do método `off`

DescritorDeTipo (`void`)
Indica que o método não devolve nenhum resultado

Lista de argumentos vazia (`()`)
indica que o método não recebe dados de entrada

```
public void off () { lampIsOn = false; }
```

Identificador : `off`
Indica o nome do método

Corpo : `lampIsOn = false;`
Define o programa que implementa o método `off`
`lampIsOn = false;`

A instrução de “afectação”

variableId = expressão

- Uma variável em Java é uma zona na memória do computador, que pode guardar um valor de certo tipo.
- Para definir (ou substituir) o valor guardado numa variável é necessário usar a **instrução de afectação (ou seja, atribuição)**.
- A forma geral da instrução de afectação é:

```
variableId = expressão
```

- `variableId` é o identificador da variável a afectar.
- `expressão` é uma expressão que calcula qual o valor a afectar à variável.
- Depois da afectação ser executada, o valor previamente guardado na variável **perde-se para sempre**.

Definição dos métodos

Em Java, as declarações de métodos escrevem-se sempre na forma seguinte:
Acesso DescritorDeTipo Identificador (lista de parâmetros) {Corpo }

```
public class Lamp {  
  
    public void on() { lampIsOn = true; }  
    public void off() { lampIsOn = false; }  
    public boolean isOn() { return lampIsOn; }  
}
```

Nestas operações, a lista de parâmetros é vazia!

Definição dos métodos

Em Java, as declarações de métodos escrevem-se sempre na forma seguinte:

Acesso *DescritorDeTipo* *Identificador* (*parâmetros*) { *Corpo* }

```
public class Lamp {  
    Corpo (return lampIsOn; )  
  
    public void on() { lampIsOn = true; }  
    public void off() { lampIsOn = false; }  
    public boolean isOn() { return lampIsOn; }  
}
```

DescritorDeTipo (**boolean**)

Identificador (isOn)

Definição dos métodos

Em Java, as declarações de métodos escrevem-se sempre na forma seguinte:

Acesso *DescriptorDeTipo* *Identificador* (*parâmetros*) { *Corpo* }

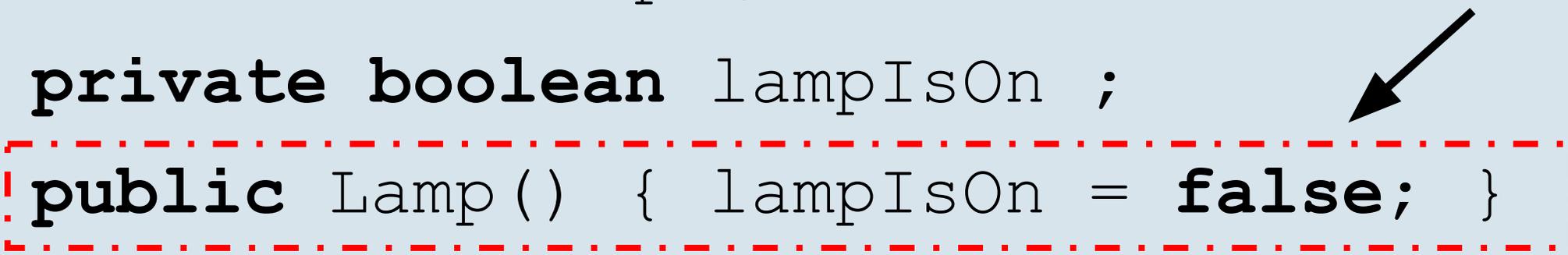
```
public class Lamp {  
    DescriptorDeTipo (void)  
    Identificador (on)  
    Corpo (lampIsOn = true; )  
    public void on() { lampIsOn = true; }  
    public void off() { lampIsOn = false; }  
    public boolean isOn() { return lampIsOn; }  
}
```

Construtor da classe Lamp

Todas as classes têm que definir ainda um **construtor**

construtor de cada objecto da classe Lamp

```
public class Lamp {  
    private boolean lampIsOn ;  
    public Lamp () { lampIsOn = false; }  
    public void on () { lampIsOn = true; }  
    public void off () { lampIsOn = false; }  
    public boolean isOn () { return lampIsOn; }  
}
```



Em geral, a função do **construtor** é definir os valores iniciais das variáveis de instância de cada objecto.

Construtor da classe Lamp

construtor de cada objecto da classe Lamp

```
public class Lamp {
```

```
    public Lamp() { lampIsOn = false; }
```

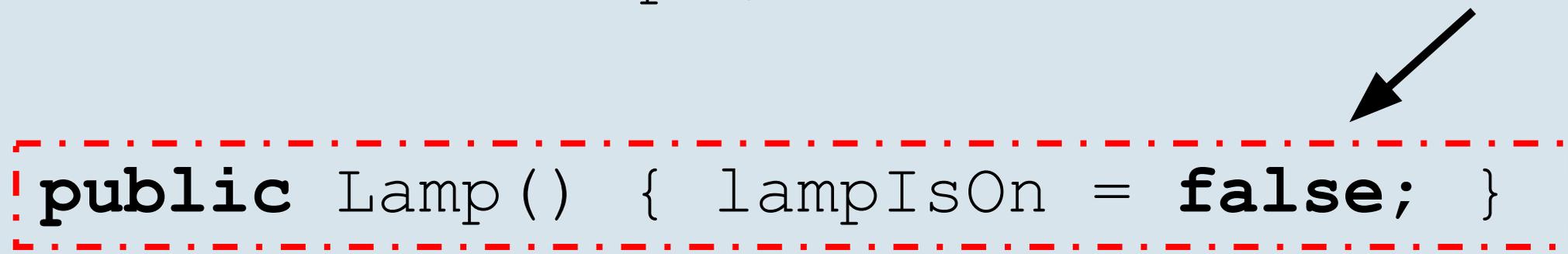
O construtor deve definir qual é o valor inicial de **todas** as variáveis de instância do objecto. Neste caso, define-se que o valor inicial da variável `lampIsOn` é o valor booleano **false**.

Construtor da classe Lamp

construtor de cada objecto da classe Lamp

```
public class Lamp {
```

```
    public Lamp() { lampIsOn = false; }
```



O construtor escreve-se:

```
Acesso NomeDaClasse() {  
    afecção de valores iniciais  
}
```

Construtor da classe Lamp

```
public class Lamp {
```

```
    public Lamp() { lampIsOn = false; }
```

O construtor escreve-se (neste exemplo):

```
    public Lamp() { lampIsOn = false; }
```

```
}
```

Define o valor inicial de variável `lampIsOn` de **cada objecto** como sendo o valor booleano **false**

A classe `Lamp` (Resumo)



A Classe Lamp (Resumo)

“memória” de cada objecto

```
public class Lamp {  
    private boolean lampIsOn ;  
    public Lamp() { lampIsOn = false; }  
    public void on() { lampIsOn = true; }  
    public void off() { lampIsOn = false; }  
    public boolean isOn() { return lampIsOn; }  
}
```

A Classe Lamp (Resumo)

Em Java, a “memória” de cada objecto consiste num certo conjunto de variáveis

“memória” de cada objecto

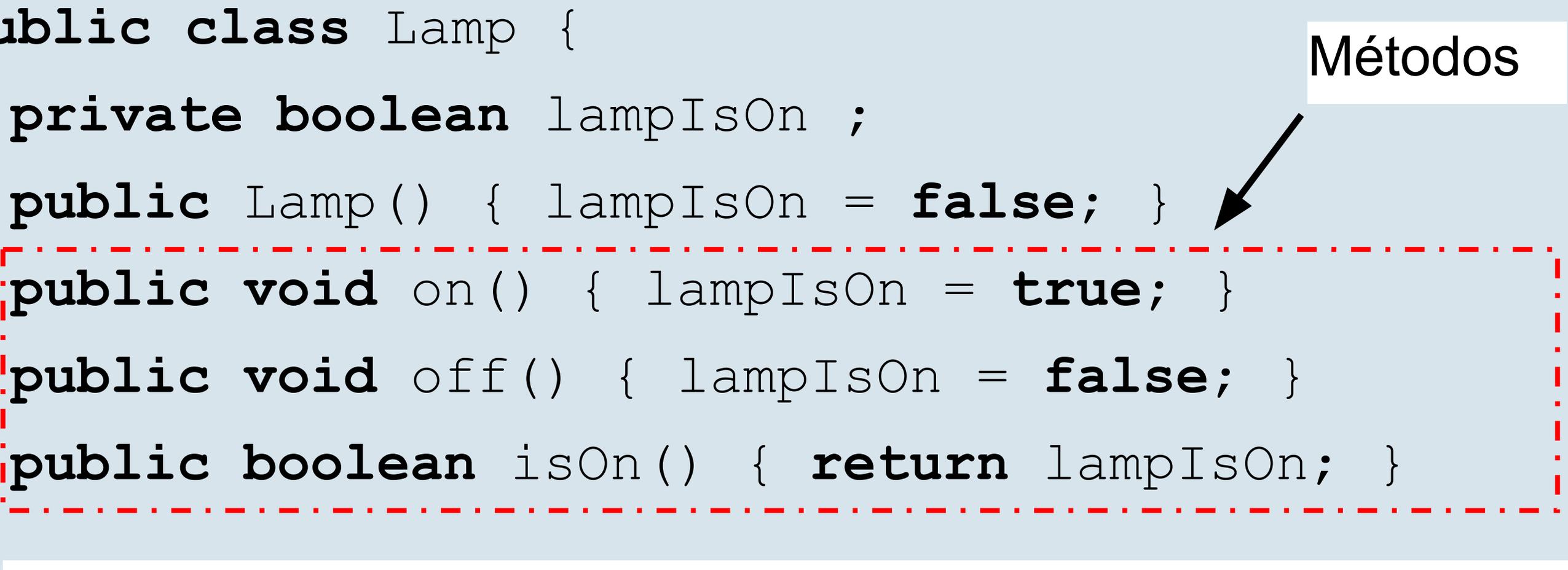
```
public class Lamp {  
    private boolean lampIsOn ;  
    public Lamp () { lampIsOn = false; }  
    public void on () { lampIsOn = true; }  
    public void off () { lampIsOn = false; }  
    public boolean isOn () { return lampIsOn; }  
}
```

Neste exemplo, a memória é a variável `lampIsOn` que guarda o estado da lâmpada (**true** : acesa; **false** : apagada)

A Classe Lamp (Resumo)

A definição dos métodos associa o **corpo do método** ao nome de cada operação da interface dos objectos

```
public class Lamp {  
    private boolean lampIsOn ;  
    public Lamp () { lampIsOn = false; }  
    public void on () { lampIsOn = true; }  
    public void off () { lampIsOn = false; }  
    public boolean isOn () { return lampIsOn; }  
}
```



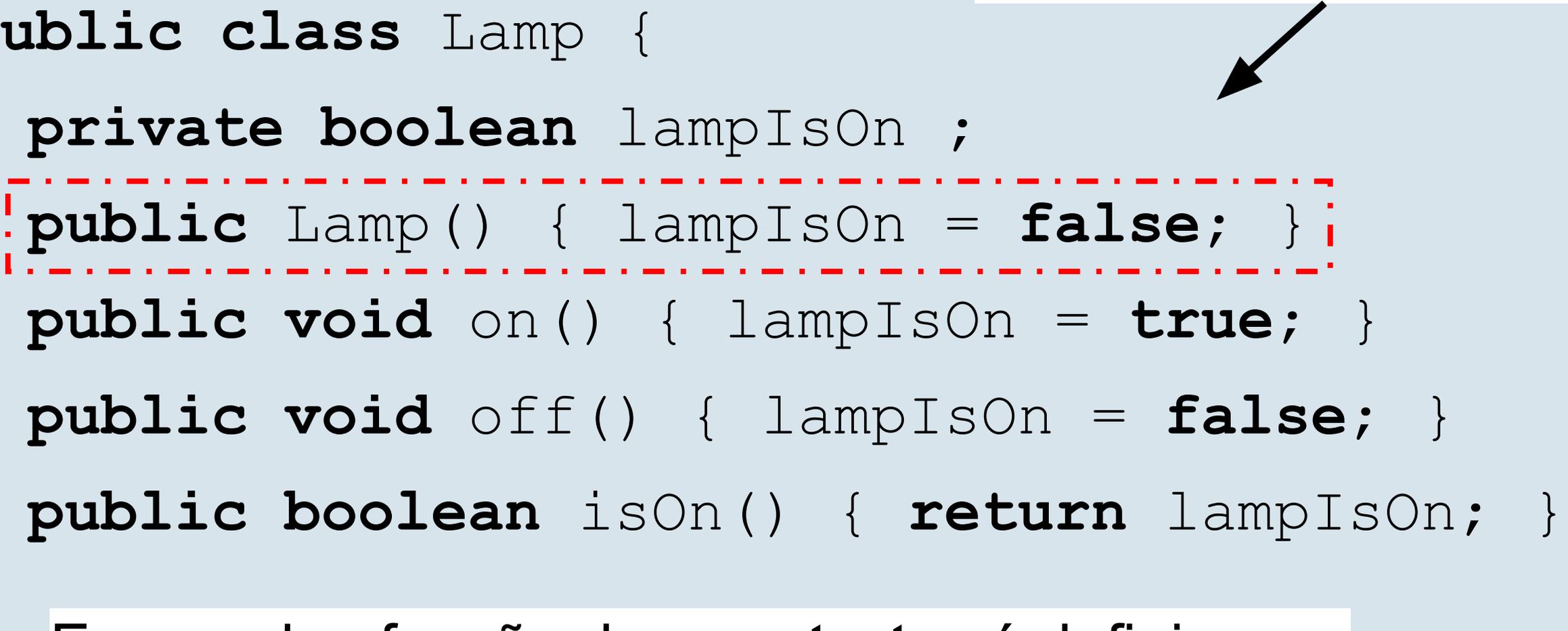
Os métodos `on` e `off` alteram o estado da lâmpada, enquanto o método `isOn` apenas indica se o estado da lâmpada é aceso

A Classe Lamp (Resumo)

Todas as classes têm que definir ainda um **construtor**

construtor de cada objecto

```
public class Lamp {  
    private boolean lampIsOn ;  
    public Lamp () { lampIsOn = false; }  
    public void on () { lampIsOn = true; }  
    public void off () { lampIsOn = false; }  
    public boolean isOn () { return lampIsOn; }  
}
```



Em geral, a função do **construtor** é definir os valores iniciais das variáveis de cada objecto.

A Classe Lamp

Verifique cuidadosamente se compreendeu a finalidade de todos os elementos desta definição.

```
public class Lamp {  
    private boolean lampIsOn ;  
    public Lamp () { lampIsOn = false; }  
    public void on () { lampIsOn = true; }  
    public void off () { lampIsOn = false; }  
    public boolean isOn () { return lampIsOn; }  
}
```

Se tem alguma dúvida, procure esclarecê-la já e agora!

Forma das classes Java

Classes Java (Resumo da Forma Geral)

```
public class classNameId {  
    private tipo varId;  
    private tipo varId;  
    ...  
    public classNameId() { corpo }  
    public tipo methodId() { corpo }  
    public tipo methodId() { corpo }  
    ...  
}
```

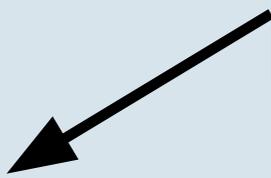
nome da classe

declaração de variáveis

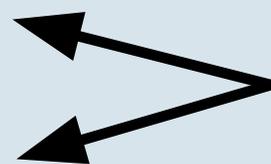
Classes Java (Resumo da Forma Geral)

```
public class classNameId {  
    private tipo varId;  
    private tipo varId;  
    ...  
    public classNameId() { corpo }  
    public tipo methodId() { corpo }  
    public tipo methodId() { corpo }  
    ...  
}
```

construtor



métodos



A classe Counter

O segundo objecto (de software)

- **Objectivo**
 - Simular um contador de valores inteiros.
- **Descrição**
 - Um contador manipula um valor inteiro.
 - **Funcionalidades**
 - O valor inteiro do contador pode ser incrementado e decrementado através das operações de “`inc`” e “`dec`”, respectivamente.
 - Quando o contador é criado o seu valor é zero.
 - Deve ser sempre possível consultar o valor guardado no contador com a operação “`status`”, e reiniciar o contador com o valor zero com a operação “`reset`”.
 - **Interacção com o utilizador**
 - Após criar um contador, pode invocar as operações do contador.

O meu segundo objecto (de software)

- **Qual o objecto a definir?**
 - Um objecto que representa um contador simples de valores inteiros.

- **Interface:**

`void inc()`

soma 1 (um) ao contador

`void dec()`

subtrai 1 (um) ao contador

`void reset()`

coloca o contador a 0 (zero)

`int status()`

consulta o valor corrente do contador



- As operações “`inc`”, “`dec`” e “`reset`” são modificadores.
- A operação “`status`” é de consulta (devolve um valor inteiro – `int`).

Comportamento esperado do contador

```
Counter c1 = new Counter();  
c1.status();  
// 0 (int)  
c1.inc();  
c1.status();  
// 1 (int)  
c1.inc();  
c1.status();  
// 2 (int)  
c1.dec();  
c1.reset();  
c1.status();  
// 0 (int)
```

Comportamento esperado do contador

```
Counter c1 = new Counter();  
c1.status();  
// 0 (int)  
c1.dec();  
c1.status();  
// -1 (int)  
c1.inc();  
c1.status();  
// 0 (int)  
c1.reset();  
c1.status();  
// 0 (int)
```

repare que o contador pode assumir valores negativos



Programação da classe Counter

- Programe em Java uma classe `Counter` que cumpra a especificação indicada
 - Para tal, precisa apenas de usar:
 - os ingredientes usados na classe `Lamp`
 - as operações `+` e `-` sobre os valores inteiros
- use “`x = x + 1`” para **somar** 1 à variável `x`
- use “`x = x - 1`” para **subtrair** 1 à variável `x`
- Experimente a sua classe usando o Eclipse.
Implemente uma classe `Main`.

Resumo

- Tivemos o nosso primeiro contacto com o Java.



- Foram introduzidos vários aspectos importantes:
 - Objectos e sua interface
 - Definição de classes em Java
 - Variáveis e memória
 - Métodos construtores, selectores e modificadores
 - Criação e utilização de objectos
- Certifique-se de que **compreendeu bem** cada aspecto, não só no contexto do problema, mas também em geral.