Classes e objectos

- A programação orientada a objectos tem a ver com a construção de programas a partir de objectos
- Desenhar um programa consiste em definir os objectos necessários, as suas funcionalidades e o modo como comunicam entre si, por forma a atingir os objectivos pretendidos
- Um objecto é uma combinação de dados (variáveis) e acções (métodos) intimamente relacionados
- Um objecto num programa funciona de modo semelhante a um objecto no mundo real
- Tem três partes: identidade, atributos e comportamento

Classes e objectos

- É possível (e comum) ter objectos semelhantes, com o mesmo comportamento, mas com atributos e identidade diferentes
- Em Java, como em qualquer linguagem OO, não é possível definir objectos directamente
- É necessário definir primeiro a classe a que o objecto pertence
- Uma classe é usada para definir um objecto (ou um conjunto de objectos semelhantes)
  - Funciona como um molde que pode ser utilizado para criar qualquer número de objectos semelhantes

Classes e objectos

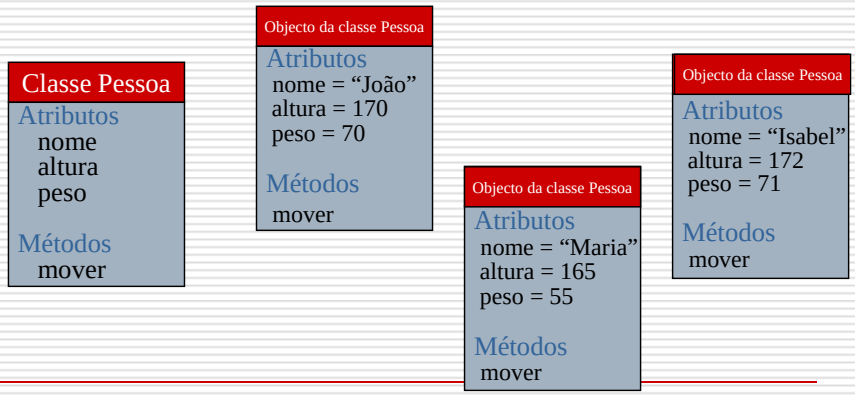
- A definição de uma classe implica a especificação dos seus atributos (variáveis) e do seu comportamento (métodos)
- Um objecto é, assim, definido como instância de uma classe e tem todas as variáveis e métodos definidos para essa classe
- A diferença entre uma classe e um objecto dessa classe é semelhante à diferença entre uma espécie (ex: Cão) e um elemento dessa espécie (ex: Scott)
- Por convenção, em Java os nomes de classes começam com maiúscula (ex: EstaClasse), e os nomes de objectos começam com minúsculas (ex: esteObjecto)

Classes e objectos

- Objectos
  - Substantivos
  - Coisas reais
- Atributos
  - Propriedades que o objecto tem
- Métodos
  - Acções que o objecto pode fazer
- Mensagens
  - Comunicação entre objectos
  - Um objecto pode pedir a outro para executar um dado método

Classes e objectos

- É possível (e comum) criar vários objectos a partir de uma mesma classe



Classes e objectos

- A criação de objectos é conseguida à custa do operador new, seguido do nome da classe a partir da qual se quer criar o objecto e de parêntesis ()
  - Se necessitar de parâmetros estes aparecem entre os ()
  - Exemplo: new Pessoa ("Maria",165, 55);
- Os objectos são criados na memória central
- Quando um objecto é criado é reservado um bloco de memória suficiente para armazenar todos as variáveis do objecto (nome, altura e peso no nosso exemplo)
- Este bloco de memória ficará ocupado até que o objecto seja destruído

Classes e objectos

- A localização de um objecto em memória não é controlada pelo programador
- Para interactuar com um objecto é necessário ter alguma maneira de o referenciar. Isto consegue-se à custa de uma referência
- Uma referência é um tipo especial de valor que identifica um objecto
- As referências podem ser armazenadas em variáveis, por exemplo:

```
Pessoa at;
at = new Pessoa (“Maria”, 165, 55);
```

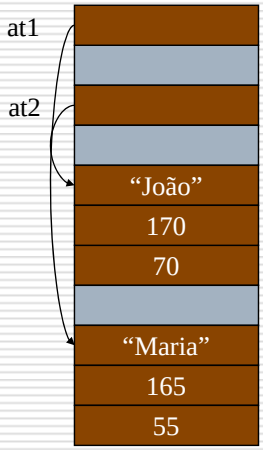

Classes e objectos

- Após a sua declaração, sabe-se que a variável at referencia objectos da classe Pessoa, mas o seu valor é indefinido
- A variável só passa a ter um valor definido após a utilização de new, por isso é comum usar-se apenas:

```
Pessoa at = new Pessoa (“Maria”, 165, 55);
```

- A variável at passa a armazenar a referência do objecto criado
- Podemos interactuar com o objecto através desta variável

Classes e objectos

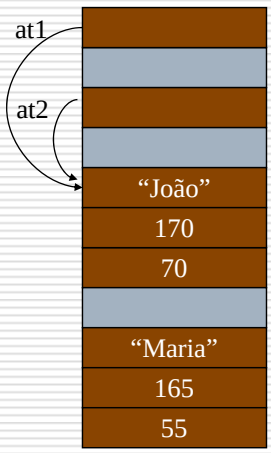


- É importante notar a diferença entre a referência a um objecto (indicação do local onde o objecto se encontra) e o próprio objecto
- Por exemplo,

```
Pessoa at1=new Pessoa("Maria",165,55);
Pessoa at2 = new Pessoa("João",170,70);
```

pode provocar a situação da figura

Classes e objectos



- Em consequência, a alteração da referência não provoca alteração no objecto. Por exemplo:

```
at1 = at2;
```

provoca o resultado da figura

- Os objectos permanecem inalteráveis e a única consequência neste caso é o facto de o segundo objecto ficar sem referência, o que provoca a sua destruição

Classes e objectos

- Quando um objecto já não tem qualquer referência válida para si, não pode ser acedido pelo programa
- É inútil e, por isso, chamado lixo ou garbage
- O Java efectua periodicamente recolha de lixo (garbage collection), devolvendo a memória ocupada por estes objectos ao sistema, de modo a que possa voltar a ser utilizada
- Noutras linguagens é o programador que tem que se preocupar em fazer a garbage collection

Classes e objectos

- Como já foi referido, a definição de uma classe implica a especificação dos seus atributos (variáveis) e do seu comportamento (métodos)
- Cada objecto criado a partir de uma dada classe tem também esses atributos e comportamentos
- Para conseguir que um objecto mostre um dado comportamento (execute um dado método) é necessário enviar-lhe uma mensagem
- Uma mensagem é um pedido que se faz a um objecto para que apresente um dado comportamento (o objecto terá que ser instância de uma classe com esse comportamento definido)

Classes e objectos

- Para enviar uma mensagem a um objecto é necessária uma instrução que consiste em:
  - Uma referência ao objecto receptor (ex: System.out)
  - Um ponto
  - A mensagem que se pretende mandar (ex: println)
- Exemplo: System.out.println ("Boa tarde…");

Classes e objectos

- O Java Development Kit (JDK) fornece centenas de classes pré-definidas para representar objectos de utilização comum
- O programador pode definir novas classes à medida das suas necessidades (como veremos mais tarde)
- A biblioteca de classes fornecida com o Java é conhecida como Java API
- As classes do Java API estão divididas em packages que agrupam classes de algum modo relacionadas
- Algumas packages comuns: java.applet, java.awt, java.lang, java.math, java.util, …..

Classes e objectos

- Para criar um objecto de uma classe incluída no Java API podemos utilizar o seu nome completo. Por exemplo:

```
java.util.Random random = new java.util.Random ();
```

- Em alternativa podemos começar por importar a classe utilizando a instrução import:

```
import java.util.Random;
...
Random random = new Random();
```


Classes e objectos

- Também é possível importar todas as classes de uma dada package. Por exemplo:

```
import java.util.*;
```

- A package java.lang é automaticamente importada para todos os programas em Java, pelo que não é necessário efectuar a sua importação explícita

Classes e objectos

- Uma das classes existente em java.lang é a String que, como o nome indica, serve para manipular cadeias de caracteres
- Para criar uma String podemos fazer:

```
String nome = new String ("Programação e Algoritmos III");
```

- Dado que a utilização de Strings é muito comum, o Java permite uma abreviatura:

```
String nome = "Programação e Algoritmos III";
```

- As Strings em Java são imutáveis, ou seja uma vez que lhe seja atribuído um valor, este não pode ser alterado

Classes e objectos

- Um caracter numa String é referido pela sua posição, ou index
- O index do primeiro caracter é zero
- A classe String inclui um vasto conjunto de métodos que permitem diversas manipulações de objectos deste tipo
- Exemplos:
  - length - devolve o número de caracteres da String
  - substring - devolve uma sub String da original
  - toLowerCase - devolve uma String igual à original com todos os caracteres convertidos para minúsculas
  - charAt - devolve o caracter num determinado ponto da String
  - equals - compara duas Strings

Classes e objectos

- Exemplo:

```
String frase = “Hoje é quinta feira”;
int tamanho = frase.length ();                          // tamanho = 19
String palavra = frase.substring (0, 4);                // palavra = “Hoje”
palavra = palavra.toLowerCase ();                       // palavra = “hoje”
char c = palavra.charAt (2);                            // c = ‘j’
boolean b = palavra.equals (“Hoje”);                    // b = false
int indice = frase.indexOf (“qui”);                     // indice = 7
palavra = frase.substring (indice, indice+7);           // palavra = “quinta “
palavra = palavra.trim ();                              // palavra = “quinta”
```


Classes e objectos

- É importante referir que, caso produzam alguma alteração, os métodos que à primeira vista alteram a String (como toLowerCase) na realidade devolvem uma nova String deixando a original intacta
- Isto decorre de as Strings serem objectos imutáveis, pelo que não é possível a sua alteração
- Em vez disso os métodos devolvem um novo objecto da mesma classe

Classes e objectos

- Exemplo: Programa que escreve as iniciais de três palavras

```
class Program2 {
    public static void main(String arg[]) {
    String   first = "Departamento";
    String   middle = "Engenharia";
    String   last = "Informática";
    String   initials, firstInit, middleInit, lastInit;

    firstInit = first.substring(0,1);
    middleInit = middle.substring(0,1);
    lastInit = last.substring(0,1);
    initials = firstInit.concat(middleInit);
    initials = initials.concat(lastInit);
    System.out.println(initials);
    }
}
```


Classes e objectos

- As instruções:

```
initials = firstInit.concat(middleInit);
initials = initials.concat(lastInit);
```

  poderiam ser substituídas por:

```
initials = firstInit.concat(middleInit).concat(lastInit);
```

- Funcionamento:
  - A mensagem concat é enviada a firstInit (com middleInit como argumento)
  - O objecto firstInit devolve uma referência para um novo objecto da classe String (“DE”)
  - A mensagem concat é enviada ao novo objecto(com lastInit como argumento)
  - A referência para um novo objecto (“DEI”) é devolvida e armazenada em initials
- A este tipo de estrutura dá-se o nome de cascata

Classes e objectos

- Em alternativa poderíamos usar:

```
initials = firstInit.concat(middleInit.concat(lastInit));
```

- Funcionamento:
  - A mensagem concat é enviada a middleInit (com lastInit como argumento)
  - O objecto middleInit devolve uma referência para um novo objecto da classe String (“EI”)
  - A mensagem concat é enviada ao objecto firstInit (com a referência anterior como argumento)
  - A referência para um novo objecto (“DEI”) é devolvida e armazenada em initials
- A este tipo de estrutura dá-se o nome de composição

Classes e objectos

- Numa cascata os resultados das mensagens são usados como receptores de mensagens adicionais, enquanto que numa composição os resultados das mensagens são usados como argumentos de outras mensagens
- Levado a um extremo a utilização de cascatas e / ou composições pode levar a código difícil de ler e propenso a erros
- Se utilizadas com cuidado e dentro de limites apertados, estas técnicas podem levar à escrita de programas mais claros e simples

Classes e objectos

- Como já vimos, a classe String tem um método substring que recebe dois inteiros. Por exemplo:

```
String frase = "Hoje é quinta feira";
String palavra = frase.substring (0, 4);          //palavra = "Hoje"
```

- No entanto, na mesma classe existe um outro método substring que recebe apenas um inteiro. Por exemplo:

```
String frase = "Hoje é quinta feira";
String palavra = frase.substring (14);            //palavra = "feira"
```

- Os dois métodos são diferentes e distinguem-se pelos parâmetros que requerem, ou seja são distintos porque têm uma assinatura diferente (a assinatura de um método é constituída pelo seu nome e pelos parâmetros requeridos)

Classes e objectos

- Os dois métodos substring da classe String são distintos e modelam comportamentos distintos (ainda que relacionados)
- Métodos com o mesmo nome, mas assinaturas diferentes na mesma classe dizem-se "overloaded"
- A prática de desenhar classes utilizando esta técnica chama-se "overloading"

Classes e objectos

- Uma outra classe útil é a StringTokenizer
- Facilita a divisão de uma String em partes (chamadas tokens)
- Por defeito os separadores considerados são os caracteres espaço, tab e newline
- Esta classe está definida na package java.util

Classes e objectos

- Exemplo de utilização da StringTokenizer:

```
String frase = "Programação e Algoritmos III";
StringTokenizer divisor = new StringTokenizer (frase);
while (divisor.hasMoreElements()) {
    String palavra = divisor.nextToken ();
    System.out.println (palavra);
}
```

- Resultado:

```
Programação
e
Algoritmos
III
```