PROGRAMAÇÃO ORIENTADA PELOS OBJECTOS

Aula 11

Extensibilidade e enumerações

Métodos e classes finais

Métodos "normais" *vs.* "finais"

- As classes, **concretas** ou **abstractas**, podem conter métodos normais, como os que temos usado, ou "finais"
- Um método diz-se **final** quando não pode ser redefinido numa sub-classe existente, ou que venha a ser criada no futuro

Implementação de um método `final`


```
public class Chess {
  public static final int WHITE = 0; // Peças brancas
  public static final int BLACK = 1; // Peças pretas
  //...
  /**
   * Retorna o primeiro jogador a mover peças, num jogo de xadrez.
   * De acordo com as regras, começam SEMPRE as brancas.
   * @return Começam as brancas.
   */
  public final int getFirstPlayer() {
    return Chess.WHITE;
  }
  //...
}
```

- Não queremos deixar que um dia mais tarde alguém viole as regras do xadrez, redefinindo este método! Assim, **temos a certeza de que começam sempre as brancas, nesta classe ou em qualquer sub-classe dela que venha a ser definida no futuro.**

A palavra reservada `final`

- Recorde o modificador **`final`**, usado no contexto da definição de constantes
  - Tal como com os métodos, **`final`** indica que o elemento por ele afectado não pode ser alterado

```
public class Chess {
  public static final int WHITE = 0; // Peças brancas
  public static final int BLACK = 1; // Peças pretas
  //...
public final int getFirstPlayer() {
    return Chess.WHITE;
  }
  //...
}
```

- Isto aplica-se a métodos, membros de dados, parâmetros, e mesmo a classes

A palavra reservada `final`

- Em geral, devemos declarar os métodos invocados pelos construtores como finais
  - Isso impede que, por polimorfia, esses métodos tenham um comportamento inesperado
- Frequentemente, podemos querer definir uma classe como final para garantir que ela não pode ser redefinida
  - Útil, por exemplo, para garantir que os objectos dessa classe são imutáveis (como acontece com as `Strings`)

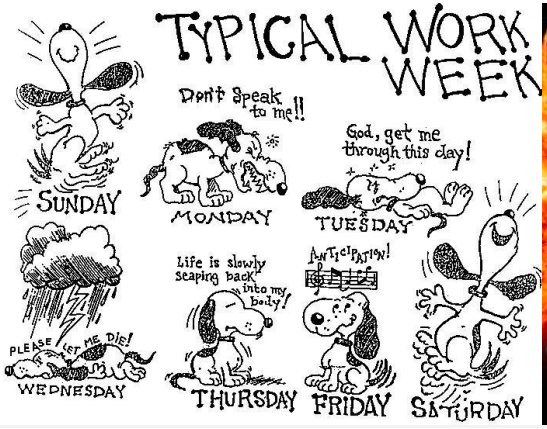
Objectos imutáveis

- Um objecto imutável é um objecto que se mantém idêntico ao longo de todo o seu ciclo de vida
  - São relativamente mais simples de entender e usar do que os objectos mutáveis
  - São inerentemente resistentes a multiplos fios de execução (thread-safe), o que significa que não necessitam de cuidados extra de sincronização
  - São muito úteis como "peças" na construção de outros objectos mais complexos

Objectos imutáveis

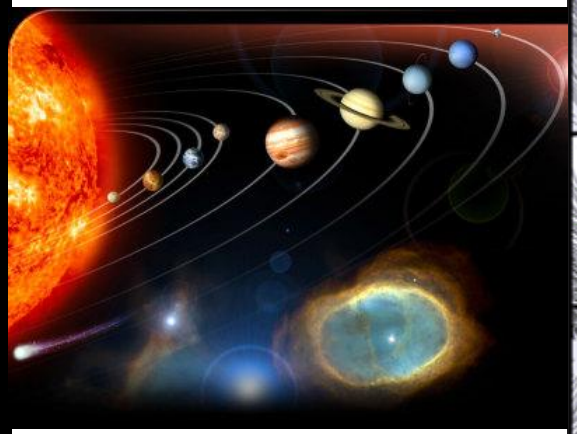
- Como criar um objecto imutável?
  - Declarar todos os seus membros como final
  - Inicializar todos os campos no construtor
  - Não disponibilizar métodos modificadores
    - Mas podem-se disponibilizar métodos de consulta
  - Declarar a classe como final, para os seus membros não poderem ser redefinidos
  - Garantir acesso exclusivo a componentes mutáveis que possa ter, por exemplo retornando cópias

Tipos de dados Enumerados



Para que serve um tipo de dados enumerado?

- Construa uma aplicação que indica o peso de um corpo nos vários planetas do sistema solar



Fed up with how her diet is going, Charlene takes a more serious aim at her target weight.

Tipos enumerados

- Um tipo enumerado é um tipo cujos membros de dados são um conjunto fixo de constantes
- Exemplos típicos
  - Pontos cardeais
    - NORTH, SOUTH, EAST, WEST
  - Dias da semana
    - SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
  - Naipes de um baralho de cartas
    - CUBS, HEARTS, DIAMONDS, SPADES
- Nota: representamos os campos do tipo enumerado com maiúsculas, dado que se tratam de constantes – mantemos assim a convenção habitual para as constantes

Definição de um tipo enumerado

- Define-se um tipo enumerado com a palavra reservada enum. Exemplos:
  - Pontos cardeais:

```
public enum CompassDirections {
   NORTH, SOUTH, EAST, WEST
}
```

  - Dias da semana:

```
public enum WeekDay {
   SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}
```

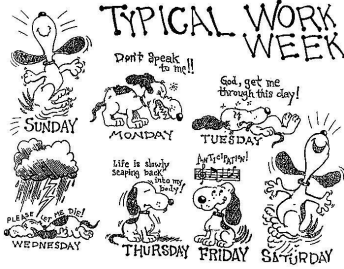
  - Naipes:

```
public enum CardSuit {
   CUBS, HEARTS, DIAMONDS, SPADES
}
```

Quando usar tipos enumerados?

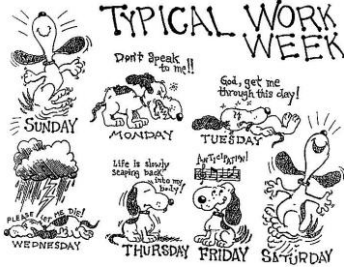
- Sempre que necessitamos de representar um conjunto fixo de constantes
  - Tipos enumerados naturais
    - tais como os pontos cardeais, dias da semana, meses do ano, ou planetas do sistema solar
  - Conjuntos de dados para os quais TODOS os valores possíveis são conhecidos em tempo de compilação
    - Escolhas de um menu, flags na linha de comando, etc.

Programando a classe de teste




```
public class EnumTest {
  public static void tellItLikeItIs(WeekDay day) {
    System.out.print(day + " ");
    switch (day) {
      case MONDAY:
      case THURSDAY:
        System.out.println("Maravilha, aula teórica de POO. ;-)");
        break;
      case TUESDAY:
      case WEDNESDAY:
      case FRIDAY:
        System.out.println("Hoje é dia de Eclipse. Vampiros?!? :-[");
        break;
      case SATURDAY:
      case SUNDAY: System.out.println("Mais tempo para programar! :-)");
        break;
      default: System.out.println("Mas… há outros?");
        break;
    } // switch
  } // tellItLikeItIs
```

Programa principal

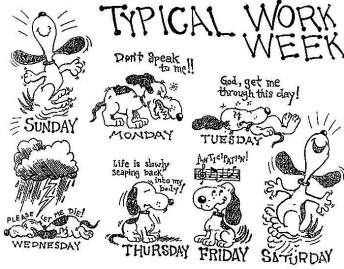



```
  public static void main(String[] args) {
     tellItLikeItIs(WeekDay.MONDAY);
     tellItLikeItIs(WeekDay.TUESDAY);
     tellItLikeItIs(WeekDay.WEDNESDAY);
     tellItLikeItIs(WeekDay.THURSDAY);
     tellItLikeItIs(WeekDay.FRIDAY);
     tellItLikeItIs(WeekDay.SATURDAY);
     tellItLikeItIs(WeekDay.SUNDAY);
  }
} // Fim da classe EnumTest
```

Isto assim é muito repetitivo.
Podemos fazer melhor!

```
MONDAY Maravilha, aula teórica de POO.
TUESDAY Hoje é dia de Eclipse. Vampiros?!? :-[
WEDNESDAY Hoje é dia de Eclipse. Vampiros?!? :-[
THURSDAY Maravilha, aula teórica de POO.
FRIDAY Hoje é dia de Eclipse. Vampiros?!? :-[
SATURDAY Fim de semana! Mais tempo para programar! :-)
SUNDAY Fim de semana! Mais tempo para programar! :-)
```

Programa principal




```
  public static void main(String[] args) {
    for(WeekDay d: WeekDay.values())
      tellItLikeItIs(d);
  }
} // Fim da classe EnumTest
```

A operação values devolve um vector de WeekDay. Esta operação é herdada da classe

java.lang.enum

O ciclo for each visita todos os elementos do tipo enumerado.

```
MONDAY Maravilha, aula teórica de POO.
TUESDAY Hoje é dia de Eclipse. Vampiros?!? :-[
WEDNESDAY Hoje é dia de Eclipse. Vampiros?!? :-[
THURSDAY Maravilha, aula teórica de POO.
FRIDAY Hoje é dia de Eclipse. Vampiros?!? :-[
SATURDAY Fim de semana! Mais tempo para programar! :-)
SUNDAY Fim de semana! Mais tempo para programar! :-)
```

O que é, afinal, um tipo enumerado em Java?

- Um tipo enumerado define uma classe que herda de java.lang.enum
- A declaração do enumerado pode incluir métodos e outros campos
- Por ser sub-classe de java.lang.enum, tem automaticamente algumas operações
  - values() devolve um vector com todos os valores do tipo enumerado, pela ordem com a qual foram declarados
  - Este método é normalmente usado em conjugação com o for-each

Exemplo: Planetas


```
public enum Planet {
  // Planet (double mass, double radius)
  MERCURY (3.303e+23, 2.4397e6),
  VENUS   (4.869e+24, 6.0518e6),
  EARTH   (5.976e+24, 6.37814e6),
  MARS    (6.421e+23, 3.3972e6),
  JUPITER (1.9e+27,   7.1492e7),
  SATURN  (5.688e+26, 6.0268e7),
  URANUS  (8.686e+25, 2.5559e7),
  NEPTUNE (1.024e+26, 2.4746e7);

  private final double mass;   // in kilograms
  private final double radius; // in meters

  Planet(double mass, double radius) {
    this.mass = mass;
    this.radius = radius;
  }
```

Cada constante do enumerado Planet é declarada com valores para os parâmetros da massa e raio dos planetas.
Estes valores são passados ao construtor quando a constante é criada.
As constantes têm de ser declaradas antes de quaisquer outros membros de dados, ou operações.

O construtor tem de ser privado, ou protegido. Cria automaticamente as constantes declaradas no início do corpo da declaração do tipo enumerado.

Exemplo: Planetas


```
  private double mass() {
    return mass;
  }

  private double radius() {
    return radius;
  }

  // universal gravitational constant  (m3 kg-1 s-2)
  public static final double G = 6.67300E-11;

  public double surfaceGravity() {
    return G * mass() / (radius() * radius());
  }

  public double surfaceWeight(double otherMass) {
    return otherMass * surfaceGravity();
  }
}
```

O tipo enumerado contém outras operações e mesmo uma constante G. Note que o valor de G, apesar de constante, **não é** um dos elementos do tipo enumerado.

Exemplo: Planetas


```
public class PlanetsMain {

/**
 * Este programa permite calcular o peso de uma pessoa em cada um
 * dos planetas do sistema solar.
 * @param args
 */
    public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      double earthWeight = in.nextDouble();
      double mass = earthWeight/Planet.EARTH.surfaceGravity();
      for (Planet p : Planet.values())
        System.out.printf("Your weight on %s is %f%n",
                          p, p.surfaceWeight(mass));
    }
}
```

Exemplo de interacção

- Exemplo de cálculo do peso de uma pessoa com 70kg (na terra), nos várrios planetas do sistema solar

```
70
Your weight on MERCURY is 26,443033
Your weight on VENUS is 63,349937
Your weight on EARTH is 70,000000
Your weight on MARS is 26,511603
Your weight on JUPITER is 177,139027
Your weight on SATURN is 74,621088
Your weight on URANUS is 63,358904
Your weight on NEPTUNE is 79,682965
```

Packages

O que é um package?

- Um package é um espaço de nomes que organiza um conjunto de classes e interfaces relacionadas
- Conceptualmente semelhante às pastas que temos nos nossos computadores
- Os packages são uma forma de organizar as classes
  - Uma aplicação pode ser composta por milhares de classes
- A plataforma do Java tem uma vasta biblioteca com milhares classes
  - http://download.oracle.com/javase/6/docs/api/index.html