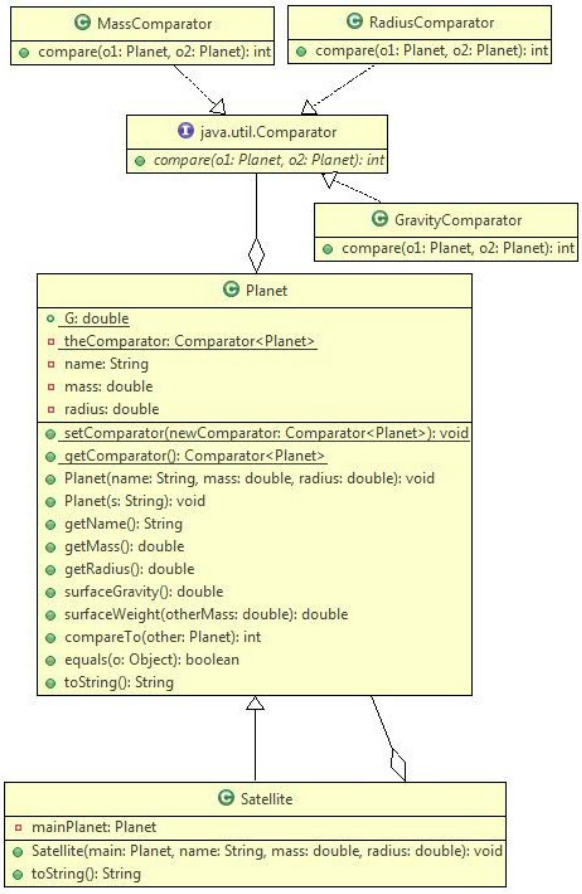
Programação Orientada pelos Objectos

Exame (época normal)
2010/06/22

Atenção: A fraude, mesmo quando detectada após a prova, é punida com a reprovação na cadeira.
Sugestão: Resolva o exame utilizando **lápis e borracha**. Evite rasuras!

Grupo 1 – Planetas

Considere o seguinte diagrama de classes que representa a classe Planet e uma interface Comparator<Planet> que permite comparar planetas dois a dois. São também representadas três implementações dessa interface e a subclasse Satellite que permite representar planetas que gravitam em torno de outro planeta (o seu mainPlanet).



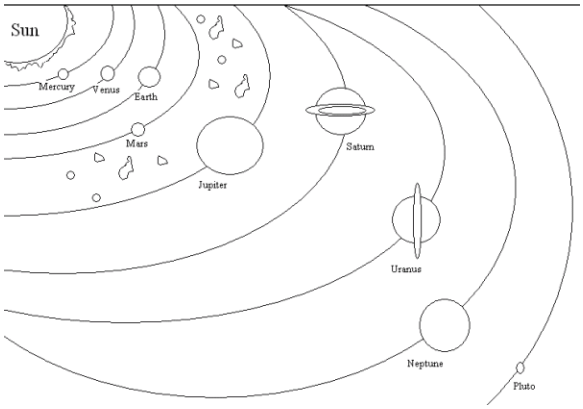
G é a constante de gravitação universal com o valor 6.67300E-17 ($Km^3\ kg^{-1}\ s^{-2}$). Sobre cada planeta sabe-se o seu nome, a sua massa (em Kg) e o seu raio (em Km).

A) Implemente **todos** os métodos representados da classe Planet, considerando que:

- O construtor Planet(s: String) recebe uma string contendo os campos nome, massa e raio, separados por um *tab* (carácter "\t").
- A função surfaceGravity() devolve a aceleração da gravidade à superfície do planeta em causa, que é igual à massa do planeta vezes a constante de gravitação universal, a dividir pelo quadrado do raio do planeta.
- A função surfaceWeight() devolve o peso de um corpo (cuja massa é passada por argumento) à superfície do planeta, que é igual à massa vezes a aceleração da gravidade à superfície do mesmo planeta.
- A função compareTo() usa a variável estática theComparator.
- Um planeta é igual a outro se tiver o mesmo nome.
- A função toString() devolve um string contendo o nome do planeta, a sua massa, o seu raio e a aceleração da gravidade, de acordo com o seguinte formato:
  MERCURY 3.303000E+23 Kg 2440 Km 3.703 ms-2

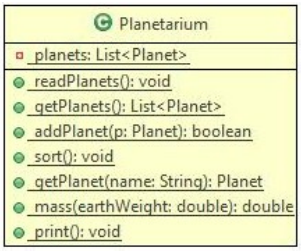
B) Programe as três classes que implementam a interface Comparator<Planet>. MassComparator servirá de base à ordenação **crescente** pela massa do planeta, RadiusComparator à ordenação **decrescente** pelo raio do planeta e GravityComparator à ordenação **crescente** pela gravidade à superfície do planeta.

C) Implemente a classe Satellite, em que os seus dois métodos devem usar, estendendo-os, os métodos correspondentes da classe ascendente. No caso da função toString() deve acrescentar-se entre parêntesis o nome do planeta principal deste satélite.



Grupo 2 – Planetário

Considere a seguinte classe Planetarium que armazena os planetas nele contidos num ArrayList.



A) Implemente a classe Planetarium, considerando que:

- A função readPlanets() lê o ficheiro "solarPlanets.txt", que inclui todos os planetas principais do sistema solar. Deve prever o tratamento de excepções na leitura. As linhas deste ficheiro têm o formato previsto no construtor Planet(s: String).
- A função getPlanets() devolve a variável planets.
- A função addPlanet() permite adicionar um planeta ao planetário e devolve true se este ainda não se encontrava lá.
- A função sort() permite ordenar todos os planetas do planetário de acordo com o critério corrente (estabelecido na classe Planet).
- A função getPlanet() permite procurar um planeta por nome, devolvendo null se não o encontrar.
- A função mass(double earthWeight) permite obter a massa de um dado corpo, que é o quociente entre o seu peso na Terra (parâmetro) e a aceleração da gravidade na Terra. Se o planeta Terra ("EARTH") não for encontrado no planetário, esta função devolve -1.
- A função print() permite escrever todos os planetas do planetário, um em cada linha, incluindo os satélites.

B) Crie uma classe Main e nela escreva a função main() que lê os planetas principais do sistema solar para um planetário (lidos do ficheiro anteriormente referido), adicione seguidamente (por programação) o satélite "MOON" ao planeta "EARTH". Escreva a função sortPlanets() que mostra todos os planetas (incluindo o satélite) ordenados, sucessivamente, por cada um dos três critérios: massa, raio e gravidade. Escreva a função showWeights(double myWeight) que, dado o seu peso no planeta Terra, mostre o seu peso à superfície de todos os planetas representados no planetário.

Grupo 3 – Números de Bell

Na análise combinatória definem-se os números Bell, em honra do matemático Eric Temple Bell (1883-1960). O n-ésimo número Bell corresponde ao nº de partições distintas de um conjunto com n elementos. Por exemplo, se tivéssemos 3 classes A, B e C, poderíamos dividi-las em pacotes de 5 formas diferentes:
{(A B C), (A B)(C), (A C)(B), (B C)(A), (A)(B)(C)}

Os primeiros números Bell (B_0 a B_{10}) são:
1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975

Um possível algoritmo para os números Bell calcula-os através de uma função recursiva com dois argumentos: *B(i,j)*. O n-ésimo número de Bell, B_n, é obtido através da chamada *B(n,n)*, ou seja:

```
public static long bell(int n)
{
        assert n >= 0;
        return B(n, n);
}
```

A) Programe a função *B(int i, int j)* tal que:
B(0,0) = B(1,1) = 1
B(n,1) = B(n-1,n-1) para n > 1
B(i,j) = B(i-1,j-1) + B(i,j-1) nos casos restantes

B) Programe uma função *testBell()* para o *JUnit* que teste os números Bell até 6.

C) Os números Bell podem ser facilmente representados através do chamado triângulo de Bell ou de Peirce. Escreva a função recursiva *void bellLine(int[] previous, int n)* para o desenhar, em que o 1º argumento é a linha anterior do triângulo e o 2º argumento é o número da linha correspondente a B_n, que é a última linha que se pretende representar (Nota: *previous* deve ter uma dimensão igual ao número de elementos que contém e *n* é constante em todas as chamadas recursivas). O algoritmo para desenhar cada linha do triângulo é:

   i) A primeira linha, correspondente a B_0, tem apenas o número 1;
   ii) Para as restantes linhas comece uma nova linha com o elemento mais à direita da linha anterior;
   iii) Cada um dos restantes números dessa linha é igual à soma do número à sua esquerda com o que está acima do número à sua esquerda (ou seja, situado na diagonal à esquerda, em cima).

O número mais à esquerda em cada linha (0, 1, 2, …) corresponde ao número Bell relativo a essa linha (B_0, B_1, B_2, …). Por exemplo, o triângulo de Bell para *n=5* seria:

```
1
1       2
2       3       5
5       7       10      15
15      20      27      37      52
52      67      87      114     151     203
```

D) Escreva uma função que represente o triângulo de Bell até B_{10}.