Linguagens e Ambientes de Programação (2018/2019)

Prática 10

Programação orientada pelos objetos em JavaScript usando classes. Exercícios 51 e 52.

Nota: Os exercícios da aula de hoje destinam-se a ser executados na consola. Nos laboratórios está instalado o JavaScript Rhino, que se invoca usando o comando "rhino".

Guião para usar a linguagem JavaScript na consola

1. Usar um editor de texto qualquer para escrever um ficheiro "Test.js". Gravar.
 2. Agora, para correr o programa, dar na consola o comando: "nodejs --use_strict Test.js".
-

Guião para usar a linguagem JavaScript dentro do Eclipse

1. Ativar Eclipse, escrevendo "eclipse" numa consola.
2. Window > Open Perspective... mudar perspectiva para JavaScript.
3. File > New > JavaScript Project. Escrever "Test" e fazer "Finish". O projeto foi criado.
4. File > New > JavaScriptSource File. Dar o nome "Test" e será automaticamente criado o ficheiro "Test.js".
5. Escrever o programa no ficheiro "Test.js" e fazer Ctrl-S (save) para gravar.
6. A lado da seta branca dentro da bola verde com a mala de ferramentas vermelha escolher "External Tools Configurations" e lá dentro escolher "Program" e fazer "New".
7. Em "Location" escrever "/usr/bin/nodejs", em "Working Directory" escolher a diretoria do projeto e em "Arguments" meter "--use_strict Test.js".
8. Fazer "Run".
9. Agora basta carregar na seta branca dentro da bola verde com a mala de ferramentas vermelha para correr o programa novamente.

Instalação: Já está instalado nos laboratórios. Para instalar no seu computador pessoal, use o comando do Eclipse "Help>Install New Software...". Na caixa "Work with:" insira "<http://download.eclipse.org/releases/neon>" e depois escolha apenas "Programing Languages/JavaScript Development Tools".

- 51 - Este é um problema de organização duma hierarquia de classes. É para ser programado em JavaScript, mas imitando as classes do Java, como se explicou na teórica 20.

Resolva o problema usando como orientação os dois exemplos finais da aula teórica 20.

Considere o problema da representação de sucessões numéricas em JavaScript. Vamos restringir-nos a três tipos de sucessões: aritméticas, geométricas e constantes.

- **Sucessão aritmética:** Caracteriza-se por um primeiro elemento a_0 e por um incremento fixo *inc*. Os elementos da sucessão são gerados usando a equação $a_i = a_{i-1} + inc$.
- **Sucessão geométrica:** Caracteriza-se por um primeiro elemento a_0 e por uma base fixa *mult*. Os elementos da sucessão são gerados usando a equação $a_i = a_{i-1} * mult$.
- **Sucessão constante:** Caracteriza-se pelo primeiro elemento a_0 , apenas. Os elementos da sucessão constante são todos iguais ao primeiro, i.e. $a_i = a_0$.

O objetivo é que cada objeto *succ* que represente uma sucessão consiga gerar os valores dessa sucessão através da invocação repetida de *succ.next()*. Na nossa representação, as sucessões devem portanto funcionar como iteradores que se lembram da posição corrente em que vão. Atenção: não guarde os valores gerados internamente num array - isso seria desperdiçar memória inutilmente.

Cada tipo de sucessão deverá ser representado por uma classe distinta. Os objetos criados a partir dessas classes (usando a operação *new*), representarão sucessões concretas. As três classes devem ter a seguinte funcionalidade comum:

- *first()* -- Faz reset da sucessão (isto é, coloca-a no início) e retorna o primeiro elemento a_0 .
- *curr()* -- Retorna o elemento corrente sem fazer avançar a sucessão.
- *next()* -- Faz avançar a sucessão e retorna o novo elemento corrente.
- *at(i)* -- Retorna o *i*-ésimo elemento da sucessão, isto é a_i . Entretanto o elemento corrente passa a ser o *i*-ésimo elemento da sucessão.
- *print(n)* -- Escreve os *n* primeiros elementos da sucessão. Entretanto, o elemento corrente passa a ser o *n*-ésimo elemento da sucessão. (Para escrever um inteiro, use a função do NodeJS "console.log".)

Resolva este problema tendo em mente os seguintes objetivos:

- Minimização do código.
- Extensibilidade do sistema.

Não se preocupe com a eficiência.

Antes de programar, desenhe num papel um esquema que ilustre a relação entre as classes que vai definir, e ainda quais são as variáveis e os métodos de instância de cada uma delas.

[Sugestões: 1. Defina uma classe abstrata e fatorize nela o máximo de código. 2. Note que os métodos "at" e "print" podem ser habilidosamente definidos à custa dos métodos "first" e "next".]

-
- 52 - Para provar que o sistema criado no problema anterior é realmente extensível, pedimos-lhe que o amplie com duas novas classes, representando dois novos tipos de sucessão:
 - **Sucessão aritmética otimizada:** Sucessão aritmética em que a operação "at" é muito eficiente. [Por vezes define-se uma classe, só para se tornarem mais eficientes determinadas operações. É o caso.]
 - **Sucessão composta soma:** Caracteriza-se por duas sucessões *a* e *b*. A sucessão soma define-se pela equação $c_i = a_i + b_i$.
-
- 53 - Adicione ainda mais três tipos de sucessão ao sistema:
 - **Sucessão composta alternante:** Caracteriza-se por duas sucessões *a* e *b*. Os elementos da sucessão alternante são gerados pela seguinte ordem: $a_1, b_1, a_2, b_2, \dots, a_i, b_i, \dots$
 - **Sucessão filtro:** Caracteriza-se por uma sucessão *a* e por um número inteiro *filtro*. Os elementos da sucessão filtro são, pela mesma ordem, os elementos da sucessão que são múltiplos de *filtro*.

- **Sucessão de fibonnaci:** Caracteriza-se pelos dois primeiros elementos a_0 e a_1 . Os elementos da sucessão são gerados usando a equação $a_i = a_{i-1} + a_{i-2}$.
-
-