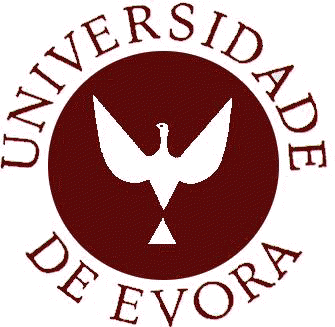
UNIVERSIDADE DE ÉVORA

TRABALHO PRÁTICO

PROGRAMAÇÃO 2

QUEM QUER SER MILIONÁRIO

23-06-2014



Discentes:

João Calhau – 31621

André Figueira – 31626

**Índice**

1. Introdução …………………………………………………... Pág. 3
2. O Código………………………………………………………. Pág. 4
   1. Questao\_QQSM

2.2QQSM

1. Conclusão ………………………………………………..… Pág. 13

**1. Introdução**

Segundo o âmbito da disciplina de Programação 2 foi-nos pedido que realizasse-mos um trabalho prático.

Este trabalho prático consiste em recriar o jogo Quem Quer Ser Milionário. Para a resolução deste trabalho iremos utilizar para a sua conclusão os apoios dos slides das aulas teóricas, os apontamentos das aulas práticas e o uso do Eclipse para a programação.

Com a resolução deste trabalho pretende-se então aumentar o conhecimento e entendimento da linguagem Java.

**2. O código**

O código esta dividido em duas classes, QSSM e Questao\_QQSM.

* 1. **Questao\_QQSM:**

Esta classe, serve o único propósito de receber 7 argumentos, como indica a imagem:

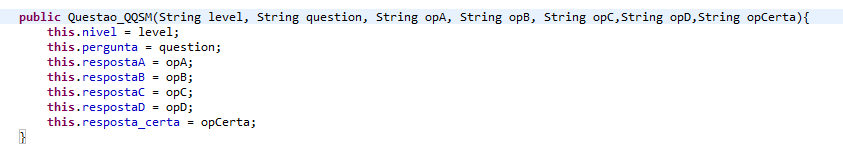


Figura 1: Constructor

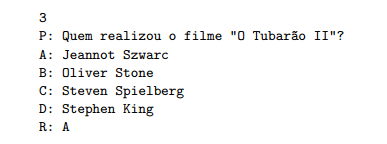
O propósito desta função é receber os argumentos necessários (de acordo com o constructor) para puder criar o formato desejado usando o método toString();

Figura 2: formato da questão

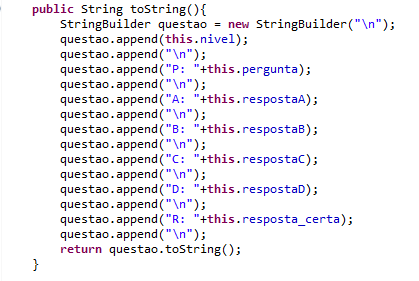


Figura 3: toString()

Após ter o formato desejado através do método **toString(),** na classe QQSM é executado uma chamada do método **add\_para\_ficheiro()**, que faz append ao conteúdo do ficheiro, sem fazer overwrite.

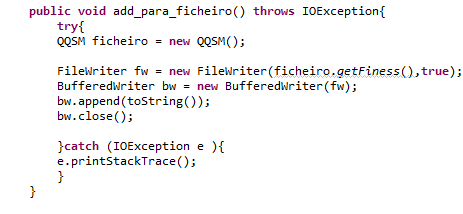


Figura 4: add\_para\_ficheiro()

**2.2 QQSM**

É esta a classe que contem o método **main**, isto significa que o programa irá ser executado através desta classe, começando pelo **main** obviamente.

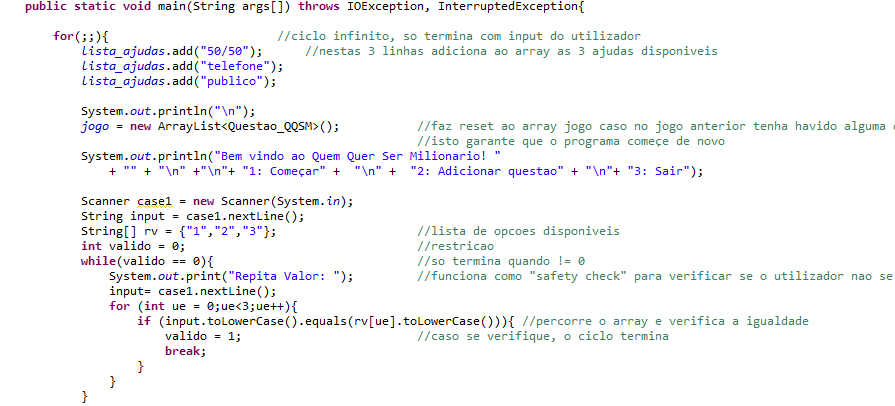
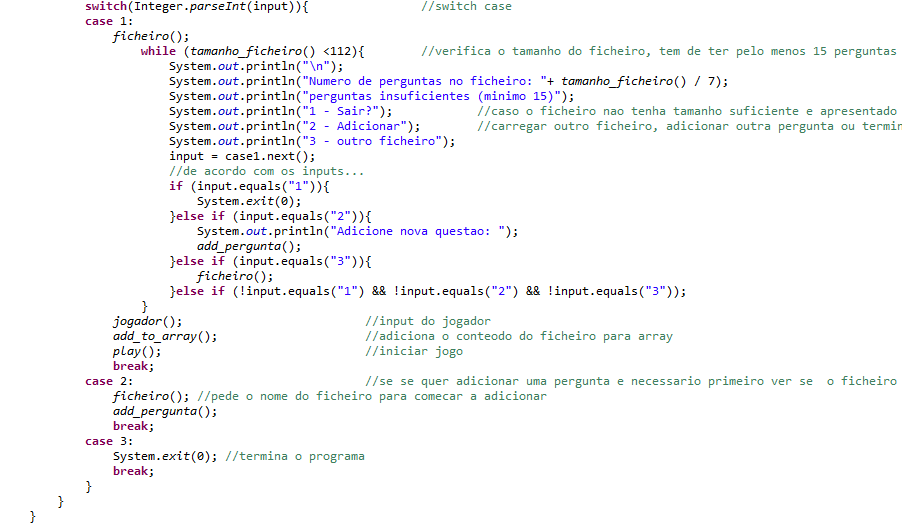


Figura 5: main

A classe main encontra-se sempre em ciclo infinito, isto permite com que seja executado um novo jogo, sempre que o anterior termina. Para o programa terminar quando o utilizador no **switch case** tem de introduzir um **3** para o programa executar **System.exit(0)**, caso contrario como dito anteriormente o programa está sempre em ciclo infinito.

No inicio do main nota-se sempre que se executa entra no ciclo, **jogo = new ArrayList<Questao\_QQSM>()** isto tem o propósito de inicializar o arraylist **jogo** que irá conter todas as perguntas do ficheiro. Não tem limite porque o ficheiro pode contar inúmeras perguntas, mas o jogo em si (não o array) funciona só com 15 perguntas, em seguinte nota-se para além da restrição (que só permite introduzir o que se encontra dentro do array, e se o input não for igual, entra dentro de um ciclo até o input ser igual a um dos elementos do array), no **case 1** a chamada de ficheiro(), este método simplesmente pede ao utilizador o nome do ficheiro + directoria onde se encontra, caso contrário criará um novo.

O ciclo **while** apresentado a seguir faz uma chamada a **tamanho\_ficheiro()** este método percorre o ficheiro dado pelo utilizador e soma todas as linhas cujo tamanho sejam diferentes de 0, o propósito serve para verificar o numero de perguntas existentes.

Dentro do ciclo **while** é dado opções ao utilizador, inserir novo ficheiro caso tenha introduzido incorrectamente o ficheiro, adicionar perguntas para puder jogar o jogo ou então sair do programa.

Assumindo que o ficheiro introduzido tem o requerimento mínimo, o programa passa ao método **jogador()** que pede o nome do concorrente ao utilizador

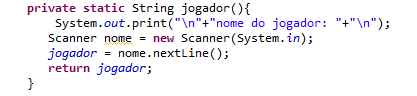


Figura 6: jogador()

A próxima chamada e referente a add\_para\_array() que adiciona todo o conteúdo do ficheiro para o arraylist **jogo**

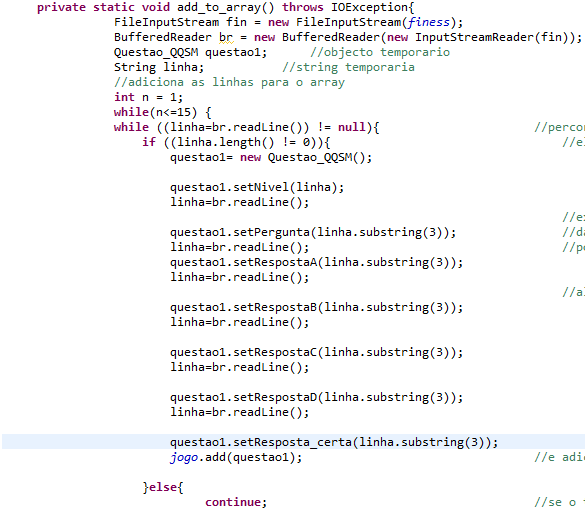
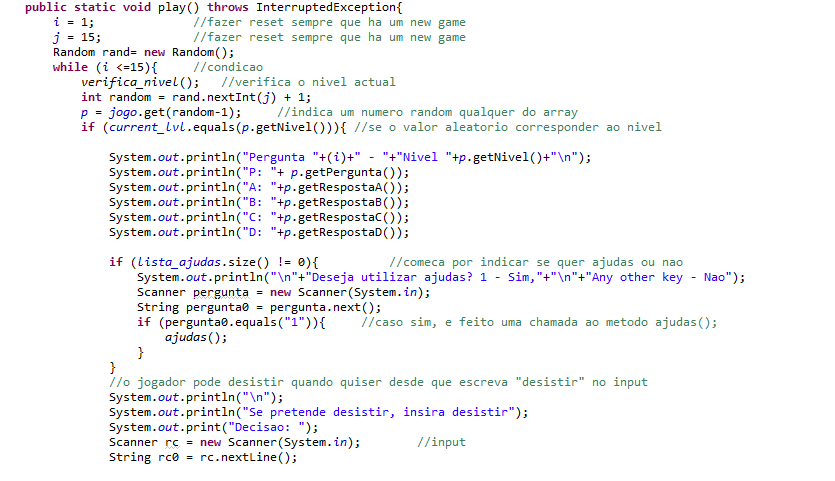
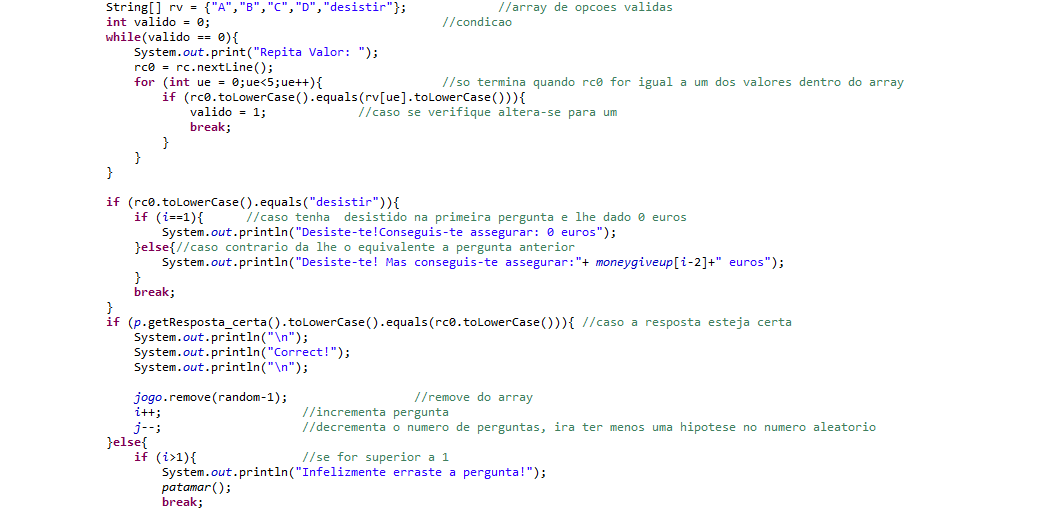
 

Figura 7: add\_to\_array()

Este método lê o ficheiro dado pelo método **ficheiro()** e percorre-o 7 em 7 linhas, caso o tamanho seja igual a 0 (ou seja, uma linha em branco passa para a próxima iteração, usando **continue**).

Enquanto percorre o ciclo, cada linha é associada a um **setter** que vai alterando o constructor e adicionando para o array **jogo**.

Após ter verificado o ficheiro, adicionar tudo para o array **jogo**, finalmente executa o método **play()**.



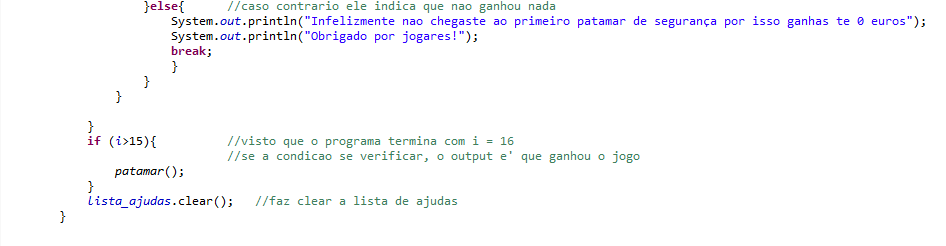


Figura 8: play()

Este método começa por inicializar a os índexes **i** e **j,** estes dois tem o propósito de incrementar e decrementar perguntas (numero de perguntas disponíveis), respectivamente. Para a execução do jogo é necessário um número aleatório, em que **j** irá fornecer o número de hipóteses aleatórias, possíveis que irá buscar perguntas aleatórias ao **jogo** através de **int random = rand.nextint (j) +1,** visto que o jogo poderá contar mais que 15 perguntas, significa que existe mais variedade. Em seguida é feita uma comparação que verifica se o nível da pergunta **jogo** é igual ao nível de acordo com **i** (mudando assim o nível em cada 5 perguntas, começando no nível 1) se for igual, o programa faz um output da perguntas mais respostas e o utilizador após analisar a pergunta tem de decidir se pretende usar ajudas ou não, caso não pretende basta introduzir input de o que quer que seja excepto 1 e o menu fecha, e se por acaso acidentalmente tenha selecionado que queria ajuda, pode anular a decisão premindo 4. Se decidir usar ajudas, o programa irá saltar para o método **ajudas()**. Se selecionar ajuda, terá 3 hipóteses, que só se poderão utilizar uma única vez, pois após a utilização são retirados do array **lista\_ajudas** (cujas ajudas são adicionadas no main), no fim do jogo e executado um clear() a **lista\_ajudas** para eliminar qualquer conteúdo. Para garantir que não são utilizadas é utilizado uma restrição que verifica se o tamanho do array é diferente de 0. Estas hipóteses consistem em 50/50, telefone e público. 50/50 devolve uma resposta aleatório mais a resposta certa (percorre um ciclo para ter certeza que não devolve duas respostas iguais). O telefone seleciona um numero aleatório do array **amigos** que consoante o nível de dificuldade lhe devolvem uma resposta, quanto maior for a dificuldade maior, mais aleatória será. Público, devolve uma percentagem aleatória do público em relação as respostas, por exemplo: “A: 25% ; B: 25% ; C: 25% ; D: 25%” e que entra em ciclo se e só se a soma das probabilidades não for igual a 1.

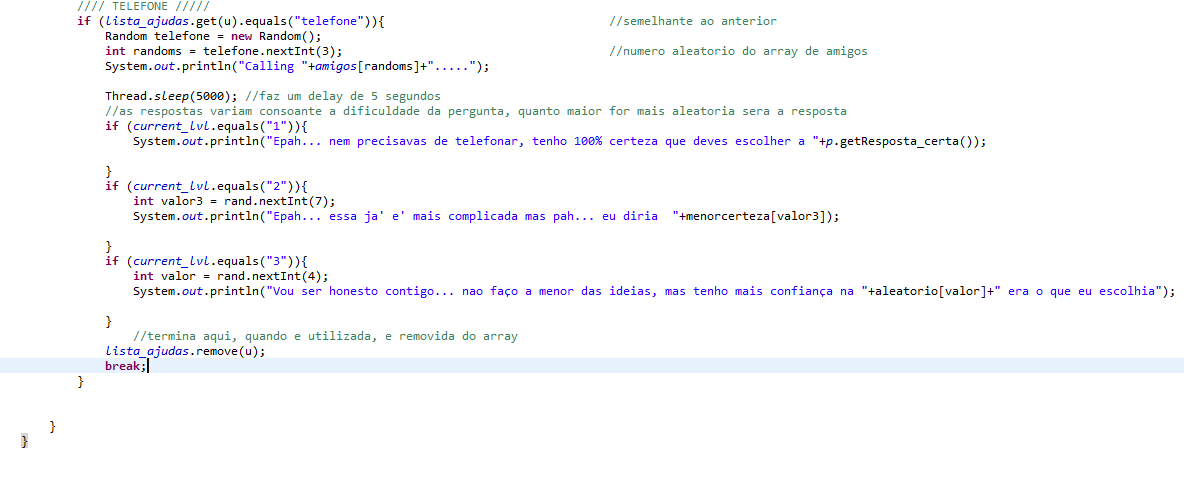
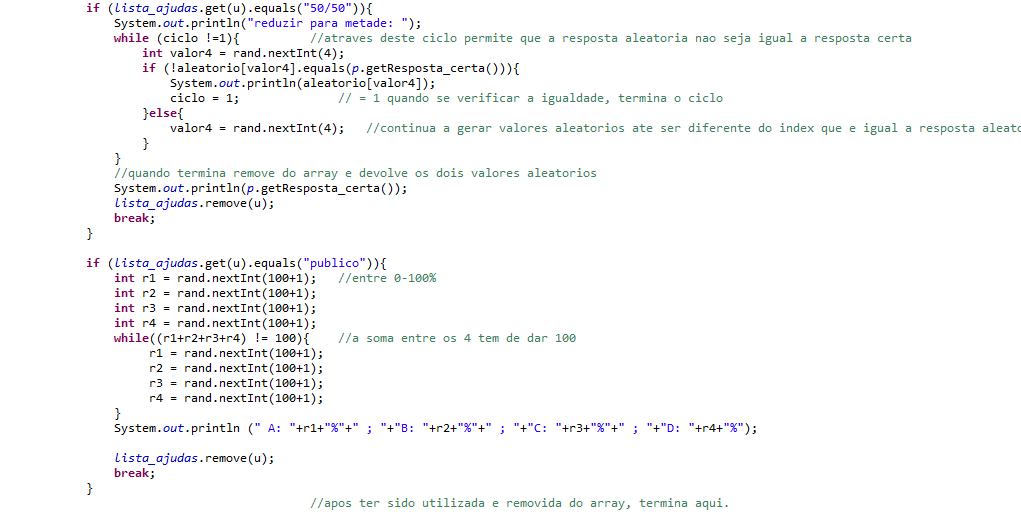
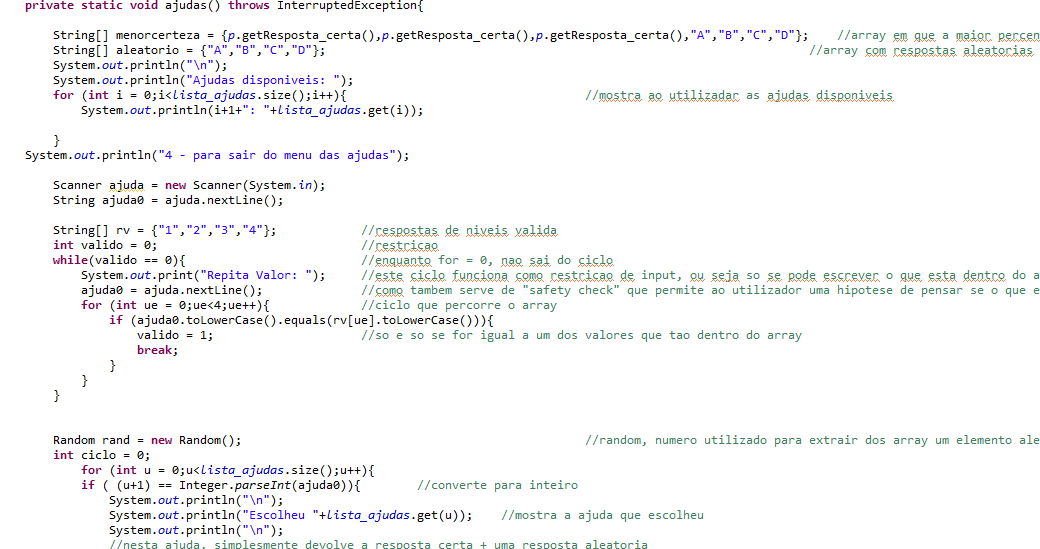


Figura 9: ajudas()

Após ter utilizado ou não as ajudas, é lhe perguntado a decisão, tem duas hipóteses, mas só a segunda é avaliada. Isto permite que o utilizador pense duas vezes antes de confirmar a resposta, por outro lado pode ser utilizado para verificar se o input está de forma correcta (só é permitido o que tiver dentro do array).

Se o concorrente acertar na resposta, passa para a próxima pergunta ate um máximo de 15. Quando acerta, os índex **i** e **j** incrementam e decrementam, respectivamente. Com **i** passa para a próxima pergunta, com **j** diminui o número de perguntas possíveis, em relação ao número aleatório.

Em vez de escrever a resposta certa pode desistir escrevendo “desistir”, e novamente lhe é perguntado se realmente quer desistir para confirmar a resposta. Se desistir leva o montante correspondente a pergunta interior, se desistir na primeira pergunta, obviamente leva 0 euros.

Se o concorrente errar uma pergunta é feito uma verificação para verificar o índex **i** se o índex for superior a 1 , faz –se uma chamada a patamar()

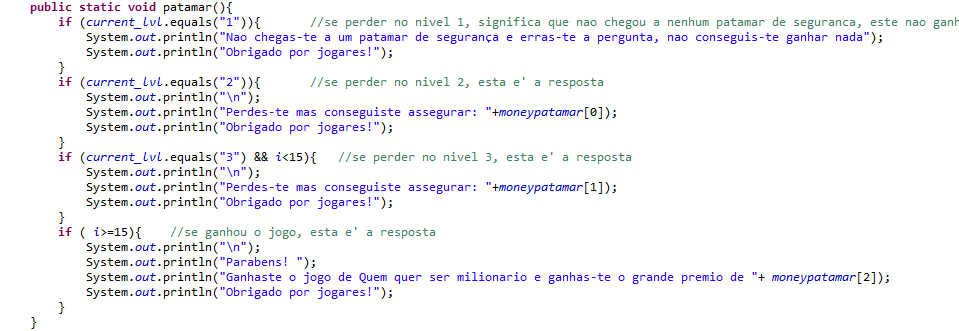


Figura 10: patamar()

Que verifica o nível actual onde se encontrava quando errou, se conseguiu chegar a um patamar de segurança o código irá verifica o nível cujo nos é fornecido por **verifica\_nivel():**

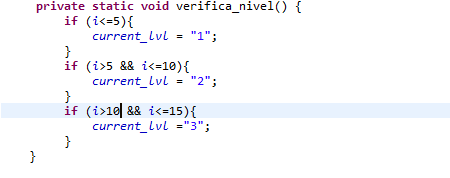


Figura 11: verifica\_nivel()

Dependendo do nível, irá devolver a resposta correspondente.

Se o concorrente conseguir chegar ao último nível e acertar executa a chamada patamar (mas verifica-se o índex **i>** 16, pois quando acertou a pergunta, o valor de i=15 foi incrementado).

Quando termina, quer por seja desistência ou por errar ou por ter ganho o jogo, o jogo recomeça e só termina quando o utilizador escolher a opção Sair.

**3. Conclusão**

Apos a conclusão deste trabalho podemos concluir que os objectivos propostos e adicionais (por nosso incentivo) foram concluídos e não só mas como também nosso conhecimento e entendimento em Java e em programação em si foram melhorados.