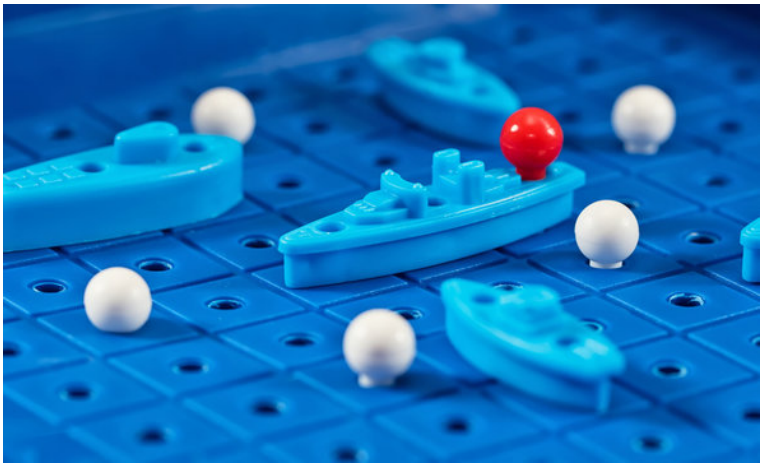
NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Departamento de Informática

Introdução à Programação
Batalha Naval II

Enunciado do Segundo Trabalho



Ano letivo 2023/24

Versão 1 (2023/11/16)

Preâmbulo

O trabalho é realizado em grupo. Cada grupo tem de ser constituído por dois alunos que, ou estão ambos inscritos pela primeira vez em IP, ou estão ambos inscritos, pelo menos, pela segunda vez em IP.

O trabalho é entregue no Mooshak, que aceita submissões até às **20h00** do dia **1 de dezembro de 2023**.

Leia este enunciado com a máxima atenção, para perceber muito bem o **novo** problema e todos os detalhes sobre o concurso do Mooshak e os critérios de avaliação do trabalho.

1 Conceitos e Objetivo do Trabalho

Nesta versão do jogo, algumas regras são bastante diferentes das habituais. Há no mínimo dois jogadores, mas pode haver mais, não existindo limite superior para o número de jogadores. Cada jogador é identificado pelo seu nome, que é único.

A batalha desenrola-se em *grelhas* retangulares (isto é, com várias linhas e várias colunas), que se encontram divididas em *células*, como se ilustra na Figura 1. As células são identificadas por um par (l, c), onde l e c são o número da linha e o número da coluna, respetivamente. O número da linha é atribuído de cima para baixo e o número da coluna é atribuído da esquerda para a direita. Tanto a primeira linha como a primeira coluna são identificadas com o número 1. Por exemplo, a célula $(1, 1)$ é a do topo esquerdo da grelha.

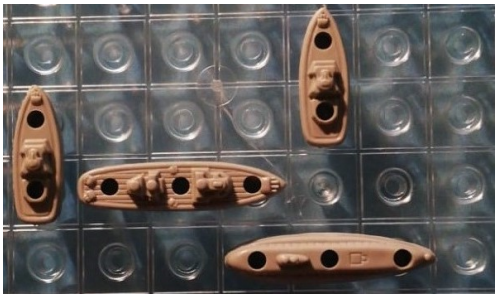


Figura 1: Grelha de 4 linhas e 7 colunas, com uma frota de 4 barcos.

Antes do jogo começar, cada jogador dispõe os seus barcos na sua grelha, sem sobreposições. As dimensões da grelha (número de linhas e número de colunas) e a constituição da frota (quais os tipos de barcos e quantos barcos há de cada tipo) não têm de ser iguais para todos os jogadores. Os barcos são todos lineares e têm de ser posicionados ou na vertical ou na horizontal. Note que os barcos podem estar encostados uns aos outros, sem água a separá-los, como na disposição esquematizada na Figura 1. O *número de canos* de um barco é o número de células que ocupa. No exemplo que estamos a usar, a frota tem quatro barcos: dois barcos de três canos e dois barcos de dois canos.

Joga um jogador de cada vez. Cada jogada consiste em escolher um adversário e dar um tiro numa posição da grelha desse adversário. Se o tiro atinge um barco, esse barco *afunda-se* (deixa de existir), independentemente do número de canos que tem. Se o tiro acerta numa posição onde já existiu um barco, o jogador sofre uma penalização. A *pontuação* de um jogador é determinada pelos barcos que afundou e pelos tiros que deu em células com "barcos afundados". A ordenação dos jogadores quanto ao desempenho, chamada *classificação*, baseia-se na pontuação dos jogadores.

Quando a frota de um jogador fica toda afundada, esse jogador é automaticamente *eliminado*, não podendo dar mais tiros nem voltar a ser atacado pelos adversários. O jogo termina assim que só houver um jogador *em jogo* (ou seja, que não foi eliminado).

```
....V..        ....X..
V...V..        A...X..
VHHH...        AXXX...
...BBB.        ...AAA.
```

Figura 2: Exemplos da representação da grelha ilustrada na Figura 1.

A disposição da frota na grelha vai ser representada por L sequências de caracteres, todas de comprimento C, onde L é o número de linhas e C é o número de colunas da grelha. Quando uma célula não tem barco, o carácter da sequência é '.' (ponto). As células onde existe um barco têm uma mesma letra nas posições correspondentes das sequências e, se houver barcos encostados

uns aos outros (na vertical ou na horizontal), as letras que os representam são diferentes. A Figura 2 tem duas representações possíveis da grelha da Figura 1.

Se, num determinado momento do jogo, houver barcos afundados, a representação do *estado da frota* tem o carácter '*' nas posições onde esses barcos se encontravam. A Figura 3 contém o estado da frota descrita pela representação da esquerda na Figura 2 se o único barco afundado fosse o que se encontrava na primeira coluna da grelha.

```
....V..
*...V..
*HHH...
...BBB.
```

Figura 3: Exemplo da representação do estado da frota.

O objetivo deste trabalho é programar em Java a versão do jogo Batalha Naval descrita neste enunciado. Com a informação sobre os nomes dos jogadores, a disposição das suas frotas e a sequência dos tiros dados, o programa deve ser capaz de indicar, em qualquer momento do jogo, qual é a pontuação e o estado da frota de cada jogador, qual é a classificação dos jogadores e quais são os jogadores em jogo. Também se pretende que, enquanto o jogo não tiver terminado, o programa possa indicar quem é o próximo jogador a dar um tiro.

2 Regras do Jogo

Competem entre si, dois ou mais jogadores. Antes do jogo começar, cada jogador dispõe os seus barcos na sua grelha (sendo obrigado a ter pelo menos um barco) e é definida a ordem pela qual os jogadores jogam. Joga um jogador de cada vez. O jogador que joga vai rodando, respeitando a ordem previamente definida e considerando-se que depois do último vem o primeiro.

Por exemplo, se houver quatro jogadores, identificados pelos nomes A, B, C e D, e for essa a ordem pela qual jogam: o primeiro a jogar é o jogador A; depois de A jogar, é a vez de B jogar; depois de B jogar, é a vez de C jogar; depois de C jogar, é a vez de D jogar; depois de D jogar, é novamente a vez de A jogar.

No início do jogo, todos os jogadores estão em jogo (nenhum jogador foi eliminado) e todos têm zero pontos. Mas, quando a frota de um jogador fica toda afundada, esse jogador é imediatamente eliminado: nunca mais joga (não dá mais tiros) e a sua grelha não pode ser o alvo dos tiros dos adversários.

A eliminação de um jogador não altera a ordem pela qual os outros jogadores jogam. Se a ordem inicial fosse A, B, C e D (como no exemplo acima) e C fosse eliminado, depois de B jogar, seria a vez de D jogar, exceto se D também tivesse sido eliminado. Caso C e D tivessem ambos sido eliminados, depois de B jogar, seria a vez de A jogar. O jogo termina assim que só houver um jogador em jogo.

Enquanto o jogo não tiver terminado, o jogador que tem o direito a jogar escolhe o adversário que quer atacar e dá um tiro numa posição da grelha desse adversário, ocorrendo uma das três seguintes situações:

- Se existe um barco nessa posição, esse barco vai ao fundo (deixando de existir). O jogador ganha tantos pontos quanto o número de canos do barco a multiplicar por 100.
- Se nunca existiu um barco nessa posição, a pontuação do jogador mantém-se.
- Se não existe um barco nessa posição porque o barco que existia foi afundado, o jogador perde tantos pontos quanto o número de canos do barco que existia a multiplicar por 30.

Na última jogada do jogo, o jogador *sobrevivente* (o que deu o tiro) recebe um bónus: depois de ter ganhado os pontos "normais" por ter afundado um barco, a sua pontuação é duplicada. O *vencedor* é o jogador que tiver a maior pontuação quando o jogo termina; se houver mais do que um jogador com a maior pontuação, ganha o sobrevivente, mesmo que a sua pontuação não seja a maior.

A partir do momento em que o jogo termina, nenhum jogador pode dar tiros. Consequentemente, a pontuação e o estado da frota dos jogadores já não podem mudar. Note que a pontuação pode ser positiva, zero ou negativa.

3 Especificação do Sistema

Pretende-se que a interface da aplicação seja simples, para poder ser utilizada em ambientes diversos e permitir automatizar o processo de teste. Por estes motivos, a entrada e a saída têm de respeitar o formato preciso especificado nesta secção. Pode assumir que o input obedece às restrições de valor e de formato indicadas, ou seja, que o utilizador não comete erros, para além dos previstos neste enunciado.

O programa lê linhas da entrada padrão (`System.in`) e do ficheiro de texto com o nome `fleets.txt`, guardado na diretoria corrente, escreve linhas na saída padrão (`System.out`) e distingue maiúsculas de minúsculas (por exemplo, as palavras "quit" e "Quit" são diferentes).

Para simplificar, no resto do documento, a palavra *frota* poderá ser usada para referir a *disposição da frota na grelha*.

Ficheiro com as Frotas

As várias frotas usadas no jogo estão guardadas no ficheiro `fleets.txt`. Esse ficheiro tem a seguinte estrutura (onde f denota o número positivo de frotas no ficheiro e o símbolo $\hookleftarrow$ representa uma mudança de linha):

$$
\begin{array}{l}
L_1\ C_1 \hookleftarrow \\
\mathit{linha}_1 \hookleftarrow \\
\mathit{linha}_2 \hookleftarrow \\
\ldots\ldots\ldots \\
\mathit{linha}_{L_1} \hookleftarrow \\
L_2\ C_2 \hookleftarrow \\
\mathit{linha}_1 \hookleftarrow \\
\mathit{linha}_2 \hookleftarrow \\
\ldots\ldots\ldots \\
\mathit{linha}_{L_2} \hookleftarrow \\
\ldots\ldots\ldots\ldots\ldots \\
L_f\ C_f \hookleftarrow \\
\mathit{linha}_1 \hookleftarrow \\
\mathit{linha}_2 \hookleftarrow \\
\ldots\ldots\ldots \\
\mathit{linha}_{L_f} \hookleftarrow
\end{array}
$$

onde, para cada frota i (com $i = 1, 2, \ldots, f$):

- L_i e C_i são números inteiros entre 1 e 100 (inclusive), que representam, respetivamente, o número de linhas e o número de colunas da grelha;

- linha_1, ..., linha_{L_i} são L_i sequências de caracteres, todas com comprimento C_i, que descrevem a frota. Cada elemento destas sequências é uma letra maiúscula (de 'A' a 'Z', sem acentos nem cedilhas) ou o carácter '.' (ponto). Cada frota tem pelo menos uma letra.

As frotas aparecem de seguida (sem linhas em branco a separá-las).

A ordem pela qual as frotas ocorrem no ficheiro `fleets.txt` vai ser usada para identificar (na entrada padrão) as frotas utilizadas no jogo: a primeira frota no ficheiro é identificada pelo número um, a segunda frota pelo número dois, e assim sucessivamente.

Entrada Padrão

A entrada padrão tem a seguinte estrutura:

$n\hookleftarrow$
nomeJogador$_1\hookleftarrow$
númeroFrota$_1\hookleftarrow$
nomeJogador$_2\hookleftarrow$
númeroFrota$_2\hookleftarrow$
.....
nomeJogador$_n\hookleftarrow$
númeroFrota$_n\hookleftarrow$
comando$\hookleftarrow$
comando$\hookleftarrow$
.....
comando$\hookleftarrow$
`quit`$\hookleftarrow$

onde:

- n é um número inteiro superior ou igual a 2;
- *nomeJogador*$_1, \ldots,$ *nomeJogador*$_n$ são n sequências de caracteres, todas diferentes, de comprimentos entre 1 e 40 (possivelmente constituídas por várias palavras, como "Fulano Jr.");
- *númeroFrota*$_1, \ldots,$ *númeroFrota*$_n$ são n inteiros positivos que não excedem o número de frotas no ficheiro `fleets.txt`, por ordem crescente (*númeroFrota*$_1 \leq$ *númeroFrota*$_2 \leq \cdots \leq$ *númeroFrota*$_n$);
- *comando* é um de seis comandos (chamados *comando-jogador*, *comando-pontuação*, *comando-frota*, *comando-classificação*, *comando-em-jogo* e *comando-tiro*) ou um comando inválido, explicados a seguir.

Depois da primeira linha, que especifica o número n de jogadores, há n pares de linhas, com o nome e o número de ordem da frota de cada jogador no ficheiro `fleets.txt`. A ordem pela qual os jogadores ocorrem define a ordem em que jogam: o primeiro a jogar será o jogador com o nome $nomeJogador_1$, o segundo será o que tem o nome $nomeJogador_2$, e assim sucessivamente.

Segue-se um número arbitrário de comandos. A última linha da entrada padrão tem um comando especial, o *comando-sair*, que só pode ocorrer na última linha porque faz terminar a execução do programa.

Comando-Jogador

O comando-jogador indica que se pretende saber quem é o próximo jogador a dar um tiro, no momento corrente do jogo. Este comando não altera o estado do jogo. As linhas com comandos-jogador têm:

`player`$\hookleftarrow$

O programa escreve uma linha na consola, distinguindo dois casos:

- Se o jogo já tiver terminado, a linha tem:

  `The game is over`$\hookleftarrow$

- Nos restantes casos, a linha tem a seguinte forma, onde *nomeJogador* representa o nome do próximo jogador a dar um tiro:

  `Next player:` *nomeJogador*$\hookleftarrow$

Comando-Pontuação

O comando-pontuação indica que se pretende saber a pontuação do jogador com o nome dado, no momento corrente do jogo. Este comando não altera o estado do jogo. As linhas com comandos-pontuação têm a seguinte forma (com um espaço a separar as duas componentes):

`score` *`nomeJogador`*↩

onde *`nomeJogador`* é uma sequência não vazia de caracteres.

O programa escreve uma linha na consola, distinguindo dois casos:

- Se *`nomeJogador`* não for o nome de um dos jogadores, a linha tem:

  `Nonexistent player`↩

- Nos restantes casos, a linha tem a seguinte forma, onde *`pontuação`* representa a pontuação do jogador referido no comando:

  `nomeJogador` `has` *`pontuação`* `points`↩

Comando-Frota

O comando-frota indica que se pretende visualizar o estado da frota do jogador com o nome dado, no momento corrente do jogo. Este comando não altera o estado do jogo. As linhas com comandos-frota têm a seguinte forma (com um espaço a separar as duas componentes):

`fleet` *`nomeJogador`*↩

onde *`nomeJogador`* é uma sequência não vazia de caracteres.

O programa escreve uma linha na consola, distinguindo dois casos:

- Se *`nomeJogador`* não for o nome de um dos jogadores, a linha tem:

  `Nonexistent player`↩

- Nos restantes casos, a linha tem a seguinte forma, onde *`estadoFrota`* representa o estado da frota do jogador referido no comando:

  `estadoFrota`↩

  O estado da frota corresponde à disposição da frota do jogador onde todas as letras que representam barcos afundados por tiros dos adversários foram substituídas pelo carácter '*' (asterisco).

Comando-Classificação

O comando-classificação indica que se pretende saber o nome e a pontuação de todos os jogadores (em jogo ou eliminados), no momento corrente do jogo. Este comando não altera o estado do jogo. As linhas com comandos-classificação têm:

`scores`↩

Se *nomeJogador* e *pontuação* representarem, respetivamente, o nome e a pontuação de um jogador, no momento corrente do jogo, o programa escreve tantas linhas quantos os jogadores, cada uma com a informação de um jogador diferente e com a forma seguinte:

nomeJogador has *pontuação* points↩

As linhas devem ser escritas por ordem decrescente de pontuação; em caso de empate na pontuação, por ordem lexicográfica de nome.

Comando-Em-Jogo

O comando-em-jogo indica que se pretende saber o nome de todos os jogadores em jogo (aqueles cujas frotas ainda não foram totalmente afundadas), no momento corrente do jogo. Este comando não altera o estado do jogo. As linhas com comandos-em-jogo têm:

players↩

O programa escreve tantas linhas quantos os jogadores em jogo, no momento corrente do jogo. Cada linha tem a seguinte forma, onde *nomeJogador* denota o nome de um jogador em jogo:

nomeJogador↩

As linhas devem ser escritas pela ordem em que os jogadores ocorreram na entrada padrão.

Comando-Tiro

O comando-tiro indica que, no momento corrente do jogo, o jogador que tem o direito a jogar deu um tiro na frota de um adversário, quem é esse adversário e qual é a célula atingida. As linhas com comandos-tiro têm a seguinte forma (com um espaço a separar componentes consecutivas):

shoot *nrLinha* *nrColuna* *nomeJogador*↩

onde:

- *nomeJogador* é uma sequência não vazia de caracteres, que identifica o adversário alvo;
- *nrLinha* e *nrColuna* são dois números inteiros que indicam que a célula atingida na grelha do adversário é (*nrLinha*, *nrColuna*).

O programa deve usar esta informação para atualizar o estado do jogo, mas não deve escrever qualquer resultado, exceto nos casos seguintes, nos quais o estado do jogo não muda e o programa escreve uma linha na consola:

- Se o jogo já tiver terminado, a linha tem:

  The game is over↩

- Se o jogo ainda não tiver terminado e *nomeJogador* for o nome do jogador que está a jogar, a linha tem:

  Self-inflicted shot↩

- Se o jogo ainda não tiver terminado e *nomeJogador* não for o nome de um dos jogadores, a linha tem:

```
Nonexistent player↩
```

- Se o jogo ainda não tiver terminado e *nomeJogador* for o nome de um jogador eliminado, a linha tem:

```
Eliminated player↩
```

- Se o jogo ainda não tiver terminado e *nomeJogador* for o nome de um adversário em jogo (não eliminado), mas *nrLinha* não for um número entre 1 e o número de linhas da grelha do adversário referido no comando ou *nrColuna* não for um número entre 1 e o número de colunas da grelha desse adversário, a linha tem:

```
Invalid shot↩
```

Comando-Sair

O comando-sair indica que se pretende terminar a execução do programa. A linha com o comando-sair tem:

```
quit↩
```

O programa termina, escrevendo uma linha na consola. Distinguem-se dois casos:

- Se o jogo ainda não tiver terminado, a linha tem:

```
The game was not over yet...↩
```

- Se o jogo já tiver terminado, a linha tem a seguinte forma, onde *nomeVencedor* denota o nome do jogador que ganhou o jogo:

```
nomeVencedor won the game!↩
```

Comandos Inválidos

Sempre que o utilizador escrever uma linha que não comece com as palavras "player", "players", "score", "scores", "fleet", "shoot" ou "quit", o estado do jogo não deve ser alterado e o programa deve escrever uma linha com:

```
Invalid command↩
```

4 Exemplo

Apresenta-se um exemplo que assume que o ficheiro `fleets.txt` tem o conteúdo apresentado na Figura 4.[1] Note que o exemplo é muito incompleto: há muitas situações que não são ilustradas e que podem ocorrer.

A coluna da esquerda ilustra a interação: o input está escrito a azul e o output a preto. Todas as linhas do input e do output terminam com o símbolo de mudança de linha, que se omitiu para aumentar a legibilidade. A coluna da direita tem informação para o leitor do enunciado, servindo apenas para relembrar as regras descritas anteriormente. Nessa coluna, "**c**" abrevia "cano(s)" e "pts" abrevia "pontos".

[1] No Mooshak, o conteúdo do ficheiro `fleets.txt` é diferente.

```
2 8
RRRGG.R.
......R.
2 8
DU...TTT
DU......
1 5
...AA
3 8
RRRGG.R.
......R.
.BB...R.
```

Figura 4: Conteúdo do ficheiro `fleets.txt`

`3`	Neste jogo há três jogadores.
`John`	Nome do primeiro jogador (o que começa a jogar).
`1`	A frota de John é a primeira frota em `fleets.txt`.
`Doe`	Nome do segundo jogador.
`1`	A frota de Doe é a primeira frota em `fleets.txt`.
`Laura`	Nome do terceiro jogador.
`3`	A frota de Laura é a terceira frota em `fleets.txt`.
`player`	
`Next player: John`	
`shoot 1 3 Doe`	John atinge barco de 3 **c** de Doe; ganha 300 pts.
`fleet Doe`	
`***GG.R.`	
`......R.`	
`score John`	
`John has 300 points`	
`player`	
`Next player: Doe`	
`shoot 2 6 John`	Doe acerta na água de John.
`player`	
`Next player: Laura`	
`shoot 2 7 Doe`	Laura atinge barco de 2 **c** de Doe; ganha 200 pts.
`fleet Doe`	
`***GG.*.`	
`......*.`	
`shoot 1 7 Doe`	John acerta em barco afundado (de 2 **c**) de Doe; perde 60 pts.
`shoot 1 8 John`	Doe acerta na água de John.
`shoot 2 3 John`	Laura acerta na água de John.
`shoot 1 5 Doe`	John atinge barco de 2 **c** de Doe; ganha 200 pts. Doe é eliminado.
`fleet Doe`	
`*****.*.`	
`......*.`	
`player`	
`Next player: Laura`	Doe nunca mais joga.
`players`	Já só há 2 jogadores em jogo.
`John`	
`Laura`	

```
score John
John has 440 points
shoot 1 4 John          Laura atinge barco de 2 c de John; ganha 200 pts.
shoot 0 1 Doe           Doe não pode ser atacado.
Eliminated player
shoot 0 1 John          John não se pode atacar a si próprio.
Self-inflicted shot
shoot 0 1 Laura         A célula (0, 1) não existe.
Invalid shot
shoot 1 1 Laura         John acerta na água de Laura.
shoot 1 7 John          Laura atinge barco de 2 c de John; ganha 200 pts.
scores
Laura has 600 points
John has 440 points
Doe has 0 points
fleet John Doe
Nonexistent player
score john
Nonexistent player
Player
Invalid command
player
Next player: John
shoot 1 2 Laura         John acerta na água de Laura.
shoot 1 2 John          Laura atinge barco de 3 c de John. John é eliminado. O jogo termina.
scores
Laura has 1800 points   Laura ganhou 900 pts de bónus.
John has 440 points
Doe has 0 points
players                 Já só há 1 jogador em jogo.
Laura
player
The game is over
quit
Laura won the game!
```

5 Entrega do Trabalho

O trabalho é entregue no Mooshak. Deverão submeter um arquivo `.zip` ao Problema A do concurso **IP2324-P2**. Não se esqueçam que:

- O arquivo deve conter apenas todos os ficheiros `.java` que tiverem criado para resolver o problema.
- O arquivo tem necessariamente de conter um ficheiro `Main.java`, onde está o método `main`.
- A classe `Main` tem de estar na raiz do arquivo (tem de pertencer ao pacote principal (`default`)).
- A versão de Java instalada no Mooshak é a 17.[2]

[2] Esta informação é irrelevante se só usarem as instruções de Java dadas nas aulas porque, nesse caso, o programa deverá ter o mesmo comportamento nas vossas máquinas e no Mooshak.

Para poderem submeter um programa, é necessário que o grupo se tenha inscrito no concurso. Os passos para proceder à inscrição são os seguintes:

1. Acedam ao Mooshak.
2. Selecionem *Register for on-line contest.*
3. Escolham o concurso IP2324-P2.
4. Preencham o formulário com o **nome do grupo**, o **endereço de email do grupo** e a seleção do *Group* IP.

O Mooshak enviar-vos-á uma mensagem para o endereço indicado, com o nome do concurso, o nome do grupo (também chamado o nome do utilizador) e a password.

O nome e o endereço de email do grupo (que escrevem no formulário da inscrição) têm de satisfazer as regras seguintes:

- O **nome do grupo** tem de ter a forma **xxxxx_yyyyy**, onde **xxxxx** e **yyyyy** representam os números de aluno dos membros do grupo.
- O **endereço de email** do grupo (para o qual o Mooshak envia a password) tem de ser o endereço institucional (@campus.fct.unl.pt) de um dos membros do grupo.

Por exemplo, o grupo constituído pelos alunos com os números 98765 e 98789 é o utilizador com o nome 98765_98789. Só serão considerados entregues (e avaliados) os programas dos utilizadores do concurso IP2324-P2 que respeitem estas regras. Se se enganarem a escrever o nome ou o endereço de email do grupo, enviem uma mensagem à Professora Carla Ferreira (carla.ferreira@fct.unl.pt) a solicitar a remoção da inscrição.

O concurso IP2324-P2 abre no dia 20 de novembro e encerra às **20h00** do dia **1 de dezembro de 2023** (sexta-feira). Podem ressubmeter o trabalho as vezes que entenderem, até à hora limite de submissão. Apenas será avaliado o programa que obtiver a **maior pontuação** no Mooshak; se houver vários programas com a maior pontuação, será avaliado, de entre esses, o **último** que tiver sido submetido. Se quiserem que o programa avaliado seja outro, têm de enviar uma mensagem à Professora Margarida Mamede (mm@fct.unl.pt) até uma hora após o concurso fechar, indicando o número da submissão que pretendem que seja avaliada.

6 Critérios de Avaliação do Trabalho

De acordo com o Regulamento de Avaliação de Conhecimentos da FCT NOVA:

- Existe fraude quando:
  (a) Se utiliza ou tenta utilizar, sob qualquer forma, num teste, exame, ou outra forma de avaliação, presencial ou a distância, informação ou equipamento não autorizado;
  (b) Se presta ou recebe colaboração não autorizada na realização dos exames, testes, ou qualquer outra prova de avaliação de conhecimentos individuais;
  (c) Se presta ou recebe colaboração, não permitida pelas regras aplicáveis a cada caso, na realização de trabalhos práticos, relatórios ou outros elementos de avaliação.
- Os estudantes diretamente envolvidos numa fraude são liminarmente reprovados na disciplina, sem prejuízo de eventual procedimento disciplinar ou cível.

Em IP, o documento Colaboração Permitida e Não Permitida, disponibilizado no Moodle, clarifica as alíneas transcritas acima. Os alunos que cometerem fraude num trabalho não obterão frequência.

A avaliação do trabalho tem duas componentes independentes, cujas notas se somam para obter a nota do trabalho:

- **Funcionalidade** (correção dos resultados produzidos)**: 10 valores**

  Um programa submetido ao concurso que só use as classes da biblioteca permitidas e que obtenha P pontos no Mooshak terá $P/10$ valores.[3] Se o programa usar classes da biblioteca não permitidas, a nota da funcionalidade será inferior $P/10$ valores.

  As classes da biblioteca permitidas são **apenas as que foram usadas nas aulas teórico-práticas** que decorreram até ao dia 14 de novembro de 2023.

  Serão fornecidos testes de treino, semelhantes aos utilizados pelo Mooshak. Se o programa produzir os resultados corretos com todos os testes de treino, é provável (mas não é garantido) que obtenha 100 pontos no Mooshak.

- **Qualidade do código: 10 valores**[4]

  Um código com qualidade tem, entre outras, as seguintes características:

  - **Várias classes que caracterizem bem as diferentes entidades do problema**;
  - Classes, métodos, variáveis e constantes com objetivos bem definidos e as restrições de acesso apropriadas;
  - Algoritmos simples e bem estruturados, implementados com as instruções mais adequadas;
  - Identificadores que expressem os conceitos que representam, escritos de acordo com as convenções ensinadas (por exemplo, o nome de uma classe deve ser um substantivo que começa com uma letra maiúscula);
  - Precondições nos construtores e nos métodos públicos (à exceção do método main);
  - Indentação correta,[5] linhas com 100 caracteres (no máximo)[6] e métodos com 25 linhas (no máximo);
  - Um comentário antes de cada método (à exceção do método main) que indique resumidamente o que o método faz.

Bom trabalho!

[3] O trabalho pode ser realizado incrementalmente. Por exemplo, pode começar por assumir que só há barcos na horizontal e que nenhum jogador dá tiros em barcos afundados. É claro que, enquanto o programa não produzir os resultados corretos em todos os testes do Mooshak, não obterá 100 pontos.

[4] Note que a qualidade do código tem um peso muito grande na nota do trabalho.

[5] No Eclipse, pode usar o comando Ctrl I (Windows) ou command I (Mac).

[6] Na contagem do número de caracteres de uma linha de código, considere que um tab equivale a 4 caracteres.