



# Análise e Desenho de Software

## 3ª fase do Trabalho 2024/2025

### Enunciado

Pretende-se com este trabalho que os alunos exercitem a compreensão dos artefactos UML criados durante o processo de desenvolvimento da aplicação SeatSeller e dos padrões de desenho abordados na disciplina, implementando em Java esses artefactos e técnicas. Para isso, são aqui apresentados:

- os diagramas de interação (DIs) das várias operações do sistema dos casos de uso a implementar;
- o diagrama de classes do desenho (DCD).

Os alunos deverão implementar as classes da camada do domínio. As classes que constituem uma possível camada de *interface* com o utilizador são dadas.

Os casos de uso a implementar são:

- UC4. Criar Grelha
- UC6. Reservar Lugar (“Criar reserva” na fase 2)
- UC7. Concluir Reserva (“Concluir pagamento” na fase 2)
- UC9. Associar Grelha
- UC10. Desassociar Grelha

A aplicação SeatSeller compõem-se de vários pacotes:

1. domain
2. console
3. gui
4. monstercard
5. portugueseExpress

O pacote 1 deverá incluir todas as classes do domínio, incluindo as *interfaces*/classes “fachada” e as *interfaces*/classes que fazem o papel de adaptadores a serviços variados, serviços estes potencialmente externos à aplicação. Todas as *interfaces*/classes que os alunos deverão implementar são parte deste pacote.

Os pacotes 2 e 3, fornecidos por nós, permitem a execução da aplicação usando dois tipos diferentes de UI – através da consola (executar a classe `Main` do pacote 2) e num ambiente de janelas (executar a classe `Main` do pacote 3)

Os pacotes 4 e 5, fornecidos por nós, representam sistemas externos de cartões de crédito, definindo APIs que as classes da aplicação SeatSeller deverão usar.

Segue-se descrição mais detalhada das partes mais relevantes à implementação do projeto.

### Pacote **domain**

O pacote **domain** inclui todas as classes, *interfaces* e enumerados descritos no pacote **domain** do diagrama de classes do desenho (DCD), apresentado em anexo a este documento. Os métodos destas classes deverão implementar os desenhos dos diagramas de interação (DIs), também em anexo.

Para maior legibilidade, no DCD não se mostram os métodos de cada classe. Segue-se, para algumas classes/*interfaces*, uma descrição mais pormenorizada para colmatar algumas dúvidas que possam ficar após inspeção do DCD e dos DIs.

---

Este pacote deverá conter as classes **Configuration** e **SeatSeller**, a construir pelos alunos, e quatro sub-pacotes descritos mais à frente.

- A classe **SeatSeller** deverá implementar a *interface* **ISeatSeller** descrita no sub-pacote **domain.api** (mais à frente neste texto). A sua implementação define o comportamento do Objeto Inicial (ver diagramas de interação). Já é dada parcialmente implementada, relativamente aos casos de uso que não dependam da autenticação (neste caso, o registo e autenticação de utilizadores).
- a classe **Configuration** deve ser *singleton* e no seu construtor deve carregar as propriedades definidas no ficheiro de propriedades de nome “`config.properties`” guardando-as num objeto do tipo **Properties** para que fiquem acessíveis aos objetos do domínio durante a execução da aplicação (através de método(s) adequados(s)). Deve implementar os seguintes métodos de instância:
  - **boolean cativarDuranteReservas()**, que devolve `true` se a propriedade “cativar” tiver o valor “true” e `false` caso contrário;
  - **double valorDuranteReservas()**, que devolve o valor da propriedade “retirar”, ou `1.0` se a propriedade não estiver definida;
  - **String valorPropriedade(String propName)**, que devolve o valor da propriedade de nome “propName”;

Embora os alunos não tenham que fazer os casos de uso Registrar Utilizador e Autenticar (já são fornecidos), têm que implementar alguns métodos de classes do pacote **domain** (**CatalogoUtilizadores** e **Utilizador**) dos quais os *handlers* dependem.

### **Pacote *domain.api* (fornecido)**

Contém as *interfaces* que o domínio torna visíveis a outros pacotes. Neste caso contém as *interfaces* correspondentes aos *handlers* dos casos de uso e a *interface* que define o Objeto Inicial (**ISeatSeller**).

Contém também um sub-pacote **exceptions**, que fornece as classes que implementam as exceções que os *handlers* podem lançar (de acordo com a documentação das *interfaces* que os definem). Como se pode verificar, todas estas classes estendem a classe **Exception** do java.

Contém ainda um pacote **wrappers** contendo classes que não são do domínio do problema mas que permitem passar informação não-simples para as classes cliente do domínio; neste caso contém uma única classe – **Combinacao** – cujas instâncias encapsulam tuplos <descriçãoDeGrelha, descriçãoDeTipoLugar, preço, numLugaresDisponiveis>, necessárias no caso de uso Reservar Lugar.

Tanto os pacotes `gui` como `console` por nós fornecidos, estão construídos só com base neste pacote **domain.api**, dando liberdade a que a implementação do domínio possa ser feita livremente (desde que implemente todas estas *interfaces*). Um exemplo da aplicação do princípio *information hiding*.

### **Pacote *domain.handlers* (a fazer pelos alunos)**

O pacote **domain.handlers** irá conter implementações dos *handlers* dos casos de uso, e também uma classe **Sessao**, de acordo com as *interfaces* do pacote **domain.api**.

As implementações dos métodos destas classes deverão estar de acordo com os Diagramas de Interação (DIs) por nós fornecidos. As exceções definidas no pacote **domain.api.exceptions** deverão ser usadas por estes métodos para sinalizar situações de exceção (e.g., grelha já existente no caso de uso Criar Grelha, etc).

A classe **domain.SeatSeller** (mais acima neste documento) cria e devolve instâncias de *handlers* que não dependam da autenticação. Um destes *handlers* é o que autentica o utilizador, devolvendo uma instância da classe **handlers.Sessao**.

Esta classe **Sessao** encapsula os dados do utilizador autenticado, bem como disponibiliza métodos para criar e obter *handlers* para os casos de uso que necessitam de autenticação. Os alunos devem também implementar esta classe **Sessao**.

### **Pacote *domain.core* (a fazer pelos alunos)**

O pacote **domain.core**, a implementar pelos alunos, deve incluir todas as classes, *interfaces* e enumerados descritos no diagrama de classes do desenho (DCD), apresentado em anexo a este documento. Os métodos deverão implementar os desenhos dos DIs, também em anexo.

- 
- a classe `domain.core.lugares.Lugar` deve redefinir o método `toString` (para efeitos de teste).
  - na classe `domain.core.lugares.Grelha`
    - o método que regista a confirmação de reserva de um dos seus lugares, deve notificar os seus observadores (`PropertyChangeListeners`) indicando o lugar e a grelha reservados;
  - na classe `domain.core.reservas.Reserva`
    - o método `double getValorEmFalta()`, deve calcular o total da reserva (invocando o método `getSubtotal` sobre todas as suas linhas de reserva) e calcular o valor já pago (invocando, sobre todos os pagamentos a ela associados, os métodos que permitem calcular esse valor); o diagrama de interação da operação de sistema `confirmarValorEmFalta` do UC7 não está suficientemente detalhado relativamente a este método `getValorEmFalta` da classe `Reserva`;
  - na classe `domain.core.utilizadores.Funcionario`,
    - no método `void propertyChange(PropertyChangeEvent evt)`, método definido no *interface* `java.beans.PropertyChangeListener`, deverão ser notificadas todas as instâncias de `api.INotificacaoReceiver`. Assim, será passado o evento à camada de apresentação (ação já implementada) para que o utilizador autenticado saiba que existe uma nova reserva associada à sua grelha.
    - Nos métodos `void associarGrelha(Grelha g, INotificacaoReceiver notR)` e `void desassociarGrelha(Grelha g, INotificacaoReceiver notR)` o funcionário deve inscrever-se (desinscrever-se) como observador da grelha `g` e deve também adicionar (remover) `notR` da sua lista de `INotificacaoReceivers`.
  - a *interface* `domain.core.lugares.alocacao.IEncontrarLugarStrategy` define estratégias de procura de lugar disponível numa grelha. Contém o método:
    - `Optional<Lugar> getLugar(Grelha g, TipoDeLugar tp, LocalDate d, LocalTime t)` que devolve um `Optional` contendo um lugar do tipo e da grelha indicados, que esteja disponível na data e hora indicados, ou `empty` se tal lugar não existir;
  - a classe `domain.core.lugares.alocacao.PrimeiroLugarStrategy` que implementa o *interface* acima, deve devolver o primeiro lugar disponível pela ordem em que está na grelha.
  - a classe `domain.core.lugares.alocacao.LugarMaisAfastadoStrategy` que implementa o *interface* acima, deve devolver o lugar cuja soma das distâncias a todos os lugares indisponíveis seja maior (em caso de empate, o primeiro por ordem).
  - a classe `domain.core.lugares.alocacao.LugarAleatorioStrategy` que implementa o *interface* acima, deve devolver um lugar disponível aleatoriamente.
  - a classe `domain.core.lugares.alocacao.EncontrarLugarStrategyFactory` é um *Singleton* que representa uma fábrica de estratégias para encontrar lugares disponíveis. Para isso deve pedir ao objeto `configuration` (que também é um *singleton*) o nome da estratégia indicada pela propriedade “`encontrarLugarStrategy`” definida no ficheiro de configurações da aplicação. É uma *strategy* deste tipo que deverá criar dinamicamente (usando reflexão) e devolver no seu método `getEncontrarLugarStrategy()`. Caso o carregamento da classe indicada na configuração não tenha sucesso, a *factory* deve criar e devolver uma instância de `LugarAleatorioStrategy`.

## **Pacote `domain.cartoesdecredito` (a fazer pelos alunos)**

Neste pacote serão definidas as interfaces e classes que permitem a interação com um ou mais sistemas de cartões de crédito. Por constituir um potencial ponto de variação, aplicámos o padrão *Adapter*, fazendo essa interação através de adaptadores.

- a *interface* `ISistemaDeCartoesDeCreditoAdapter` define tipos de adaptadores para interação com sistemas de cartões de crédito. Contém os métodos seguintes, que todos os adaptadores devem implementar:
  - `boolean validar (String num, int ccv, int mes, int ano)` que retorna `true` se o cartão de crédito com o número `num`, código `ccv2 ccv`, e data de expiração `mes/ano`, for válido;
  - `boolean cativar (String num, int ccv, int mes, int ano, double qt)` que cativa a quantia `qt` no cartão de crédito com o número `num`, código `ccv2 ccv`, e data de expiração `mes/ano`, retornando `true`; retorna `false` se não for possível cativar;
  - `boolean retirar (String num, int ccv, int mes, int ano, double qt)` que retira a quantia `qt` do cartão de crédito com o número `num`, código `ccv2 ccv`, e data de expiração `mes/ano`, retornando `true`; retorna `false` se não for possível retirar ;

- as classes `MonsterCardAdapter` e `PortugueseExpressAdapter` do pacote `domain.cartoesdecredito` implementam `ISistemaDeCartoesDeCreditoAdapter`, implementando os três métodos de forma a seguirem as APIs dos sistemas de cartões de crédito correspondentes (Ver Anexos III e IV).
- a classe `SistemaDeCartoesDeCreditoAdapterFactory` é um *Singleton* que representa uma fábrica de adaptadores para sistemas de cartões de crédito. Para isso deve pedir ao objeto `Configuration` (que também é um *singleton*) o nome do adaptador indicado pela propriedade “cartaoAdapter” definida no ficheiro de configurações da aplicação. É um *adapter* deste tipo que deverá criar dinamicamente (usando reflexão) e devolver no seu método `getSistemaDeCartoesDeCreditoAdapter()`
- a classe `SistemaDeCartoesDeCreditoDummy` deste pacote implementa `ISistemaDeCartoesDeCreditoAdapter`, implementando os três métodos de modo a retornarem `true`. Esta classe só serve para que a *factory* possa ter uma alternativa no caso do carregamento da classe indicada na configuração não ter sucesso.

## Ficheiro de texto `config.properties` (fornecido)

No projeto é também fornecido um ficheirinho de texto definindo os valores para as propriedades que vão ser necessárias durante a execução da aplicação.

Este ficheiro deverá estar no `currentdir` aquando do arranque da aplicação (mesmo caminho do projeto eclipse). Os valores a serem usados para a geração de output exemplo são indicados abaixo. Deve experimentar mudar esses valores e verificar a consequência na execução da aplicação.

```
cativar = false
retirar = 10
cartaoAdapter = domain.cartoesdecredito.PortugueseExpressAdapter
encontrarLugarStrategy = domain.core.lugares.alocacao.PrimeiroLugarStrategy
```

## Utilização de padrões de desenho

No desenho do modelo de classes apresentado, foram usados vários padrões de desenho. Seguem-se alguns padrões e exemplos de utilização neste projeto:

- **Factory.** Usamos fábricas para criar as estratégias de escolha de lugar disponível e os adaptadores para os vários sistemas externos de cartões de crédito;
- **Singleton.** As fábricas, por exemplo; a classe `domain.Configuration` também;
- **Iterator.** A linguagem Java tem suporte para este padrão, através dos interfaces `java.util.Iterator` `java.util.Iterable` e do ciclo `foreach`.
- **Façade.** As classes do domínio devem ser protegidas por uma fachada; os *handlers* dos casos de uso são casos particulares de objetos fachada; A classe `domain.handlers.Sessao` é também um objeto fachada que esconde o funcionamento das classes de tipos de Utilizador e os casos de uso associados.
- **Observer.** Notificação entre `domain.utilizadores.Funcionario` (observador) e `domain.lugares.Grelha` (observável);
- **Strategy.** A interface `IEncontrarLugarStrategy` do pacote `domain.lugares.alocacao` “embrulha” as várias formas ou estratégias de escolha de um lugar disponível;
- **Adapter.** A interface `domain.cartoesdecredito.ISistemaDeCartoesDeCreditoAdapter` embrulha as várias implementações de adaptadores para comunicação com Sistemas de cartões de crédito;
- **Protected variations.** “Proteger” os elementos da aplicação das variações noutros elementos (objetos, sistemas, subsistemas), “embrulhando” o foco de instabilidade com uma *interface* (ou classe abstrata) e usando polimorfismo para usar várias implementações dessa *interface*. Vários dos padrões referidos acima (e.g., Strategy, Adapter, Factory) são variações mais específicas deste padrão.

## Material fornecido

São fornecidos dois ficheiros:

- O pdf `IDs&DCD` contendo os Diagramas de Interação (DIs) e o Diagrama de Classes do Desenho (DCD).
- O zip `SeatSeller-alunos.zip` contendo os pacotes de classes java e o ficheiro `config.properties`.

**ATENÇÃO:** ver na página seguinte como usar o `SeatSeller-alunos.zip` que vos é disponibilizado.

---

## O que há a fazer

Produzir as classes e *interfaces* Java mencionadas anteriormente de acordo com os DIs e o DCD fornecidos e, eventualmente, outras classes ou métodos que considere relevantes.

É preciso tomar em atenção que a solução de desenho descrita não é suficientemente detalhada para que não tenha de tomar ainda algumas decisões ao nível da implementação (por exemplo, a acessibilidade das classes e métodos, a factorização de código através de definição e utilização de classes abstratas, a redefinição apropriada dos métodos herdados de *Object*, etc). Devem ser tomadas as decisões que forem consideradas mais adequadas relativamente a tudo o que não estiver descrito.

Todas as classes **devem estar convenientemente documentadas**. Além dos comentários em *javadoc* (úteis para os clientes da classe) deverá sempre explicitar informação útil para quem tiver de alterar a classe.

## Como entregar

Fazer **upload** no **Moodle** do código Java produzido, através de um ficheiro *zip* contendo TODAS as classes do projeto. **Este ficheiro zip deverá ter o nome ADSxxx** (onde **xxx** é o número do vosso grupo). Somente um dos elementos do grupo deve fazer o *upload*.

**IMPORTANTE:** Acrescentem também na classe `console.Main` comentários com a indicação do número do vosso grupo e dos números e nomes dos elementos que o compõem.

## Quando entregar

O Moodle permitirá o *upload* dos projetos até às 23h55 do dia 26 de Maio.

## O que não devem esquecer

A apresentação e documentação da implementação que produzirem e entregarem é **TAMBÉM** importante na medida em que pode facilitar ou dificultar a sua leitura, interpretação e manuseamento. Este aspeto não deve portanto ser descurado!

Qualquer forma de plágio implica a não aprovação na disciplina. Plágio inclui, entre outras formas, copiar ou deixar copiar trabalhos de desenho ou programação. Cada grupo de trabalho é responsável por assegurar que os seus trabalhos são atribuíveis apenas a si.

## Como usar as classes fornecidas no *SeatSeller-alunos.zip*

- Assegurar-se que tem uma versão do java 17 ou posterior.
- Fazer download do Java FX (caso ainda não o tenha instalado):
  - No link <https://gluonhq.com/products/javafx/> escolher o adequado ao vosso sistema operativo;
  - Proceder à instalação, escolhendo o local no vosso disco (precisarão de o localizar mais tarde);
- Descomprimir o ficheiro *SeatSeller-alunos.zip* fornecido, que contém várias pastas com ficheiros java.

### No Eclipse:

- Criar um projeto Java
  - Dar-lhe o nome ADSxxx (onde xxx é o número do grupo);
  - Não criar module-info na criação do projeto;
  - Certifiquem-se que o JRE é 17 ou superior; (\*)
- Copiar as pastas contidas no zip fornecido para a pasta src do vosso novo projeto
  - Pode fazê-lo por arrastamento;

- São 5 pastas (*console*, *domain*, *gui*, *monstercard*, *portugueseexpress*), que dão origem a 15 pacotes e sub-pacotes no projeto java, três deles vazios;
- Não se esqueça de arrastar também o ficheiro *config.properties* para a raiz do projeto;
- O pacote *domain* e o pacote *gui* apresentam erros de compilação (o primeiro porque lhe faltam as classes e métodos que os alunos têm que implementar; o segundo porque precisa do Java FX instalado e ligado ao projeto – ver abaixo).

(\*) Se instalarem a nova versão do java no vosso computador, depois, no Eclipse, escolham essa nova versão em Eclipse → Preferences → Java → Installed JREs


Embora o pacote *gui* apresente erros de compilação, se quiser somente executar o *main* da classe *console.Main*, que é um cliente que só usa a consola, pode fazê-lo pois esta classe não usa Java FX.

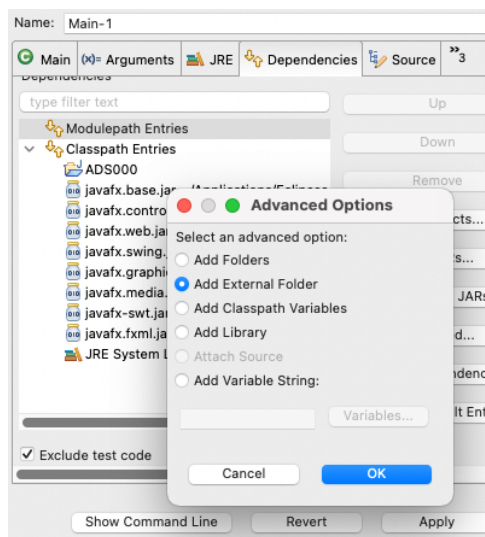
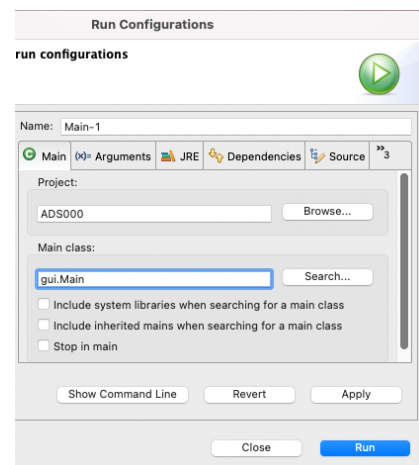
## Para poder usar o cliente gui

Se quiser ter a possibilidade de executar a aplicação *SeatSeller* através da interface gráfica (executando o *main* da classe *gui.Main*) deverá ainda dar os passos seguintes:

- Selecionar o projeto e escolher Properties
  - Selecionar opção “Java Build Path” à esquerda na janela
  - Selecionar a aba “Libraries” em cima
  - Selecionar “ClassPath” na janela, e o botão “Add external jars” à direita
  - Procurar na janela de ficheiros a pasta *lib* do *javafx-sdk-20.0.1*
  - De dentro dessa pasta selecionar todos os ficheiros *.jar* acessíveis e clicar em Open
  - De seguida, Apply and Close. Verá os erros de compilação a desaparecer.

Quando quiser executar a aplicação, no Eclipse:

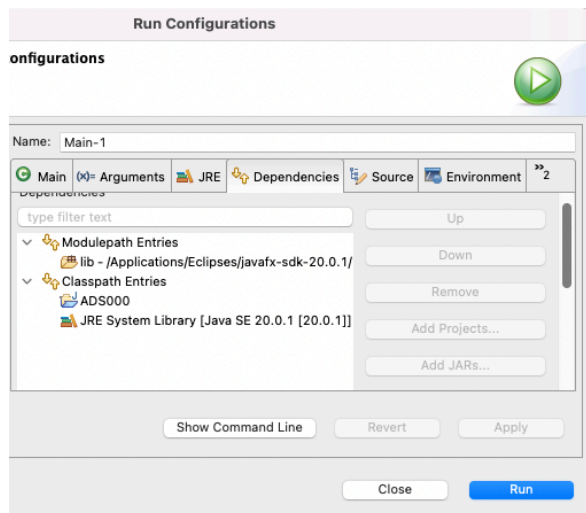
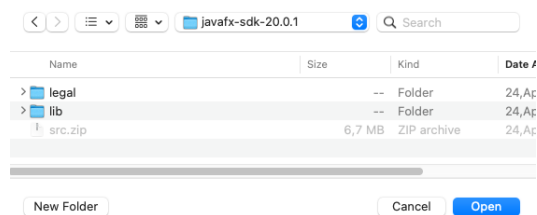
- Para executar o *main* da classe *gui.Main*, deve:
  - Selecionar a classe *gui.Main*;
  - Menu Run → Run configurations;
  - Carregar em  para criar uma nova configuração;
  - Dê um nome à nova configuração (Main-1 no exemplo ao lado);
  - Na aba “Main”, certifique-se que o projeto e a Main class estão corretas (pode ser que tenha que usar Browse e Search para isso);
  - Carregar em Apply;



○ De seguida, na aba “Dependencies”, carregar no “Modulepath Entries” e escolher o botão “Advanced” à direita;

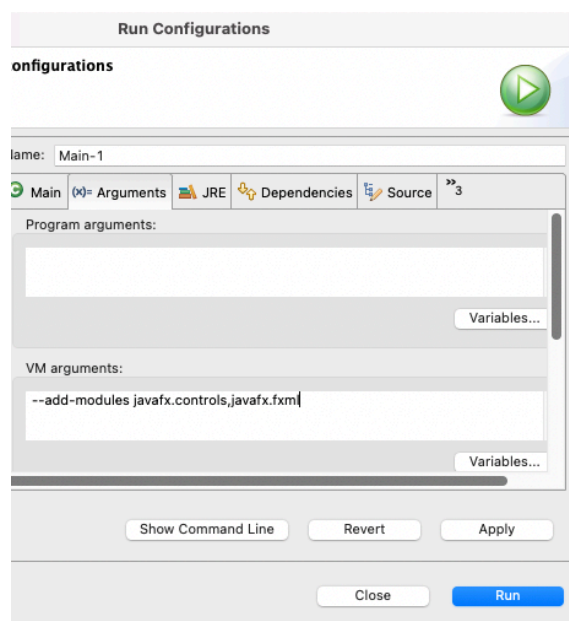
○ Na janela “Advanced Options” escolher “Add External Folder” e OK;

- De seguida, na janela do browser, escolher a pasta lib do pacote javafx-sdk-20.0.1 (pode ter que subir um nível para selecionar a própria pasta lib);
- Clicar em Open;
- Carregar em Apply;



- De seguida, selecionar todos os .jar do javafx que estão nas "Classpath Entries" e clicar no botão Remove à direita;
- Carregar em Apply;

- De seguida, na aba "(x)=Arguments", escrever, na caixa "VM arguments",  
--add-modules javafx.controls, javafx.fxml
- Carregar em Apply;
- Finalmente, carregar em Run.



## Quando entregar

Até às 23h55 do dia 26 de Maio.

## O que não devem esquecer

A apresentação e documentação da implementação que produzirem e entregarem é TAMBÉM importante na medida em que pode facilitar ou dificultar a sua leitura, interpretação e manuseamento. Este aspeto não deve ser descurado!

Qualquer forma de plágio implica a não aprovação na disciplina. Plágio inclui, entre outras formas, copiar ou deixar copiar trabalhos de desenho ou programação. Cada grupo de trabalho é responsável por assegurar que os seus trabalhos são atribuíveis apenas a si.



---

## Anexo I: Output do Programa de Teste

A classe `console.Main` tem um método `main` que provoca o registo e autenticação de utilizadores, a criação de tipos de lugar, a criação de grelhas, a reserva de lugares e a conclusão de uma reserva.

Qualquer situação fora do normal (correspondente aos cenários alternativos dos casos de uso), detetada pelo lançamento de uma exceção, é descrita através de um objeto do tipo `java.util.Logger`.

```
May 01, 2025 3:45:29 PM console.Startup criarContas
INFO: Utilizadores criados
May 01, 2025 3:45:29 PM console.Startup criarTiposDeLugar
INFO: Tipos de lugares criados
May 01, 2025 3:45:29 PM console.Startup criarGrelhas
INFO: Grelhas criadas
May 01, 2025 3:45:29 PM console.Startup lambda$1
INFO: Funcionario Serafim registado
May 01, 2025 3:45:29 PM console.Startup lambda$1
INFO: Serafim associado a grelha Sala VIP
May 01, 2025 3:45:29 PM console.Startup lambda$1
INFO: Serafim associado a grelha Sala 1
May 01, 2025 3:45:29 PM console.Startup criarReservas
INFO: Atribuido lugar Lugar Lugar VIP:0,0 em Sala VIP
May 01, 2025 3:45:29 PM portugueseexpress.PortugueseExpress validate
INFO: Validating card...
May 01, 2025 3:45:29 PM console.Startup criarReservas
INFO: Preco: 12.0
May 01, 2025 3:45:29 PM portugueseexpress.PortugueseExpress charge
INFO: Charging card...
May 01, 2025 3:45:29 PM console.Startup lambda$2
INFO: Notificação ao Serafim: Lugar Lugar VIP:0,0 em Sala VIP
May 01, 2025 3:45:29 PM console.Startup lambda$2
INFO: Notificação ao Serafim: Lugar Lugar VIP:0,0 em Sala VIP
May 01, 2025 3:45:29 PM console.Startup criarReservas
INFO: Confirmada Reserva: R0
May 01, 2025 3:45:29 PM console.Startup concluirReserva
INFO: Valor em falta: 2.0
May 01, 2025 3:45:29 PM portugueseexpress.PortugueseExpress validate
INFO: Validating card...
May 01, 2025 3:45:29 PM portugueseexpress.PortugueseExpress charge
INFO: Charging card...
```



## Anexo II – Breve explicação da interface de utilizador

A interface de utilizador desenvolvida é muito simples e padrão, pelo que não estamos à espera que seja difícil de utilizar. Ao arrancar com a aplicação (executando a classe `gui.Main`) aparece o seguinte ecrã

Este ecrã implementa o caso de uso *Login de utilizador*. Existem alguns utilizadores predefinidos:

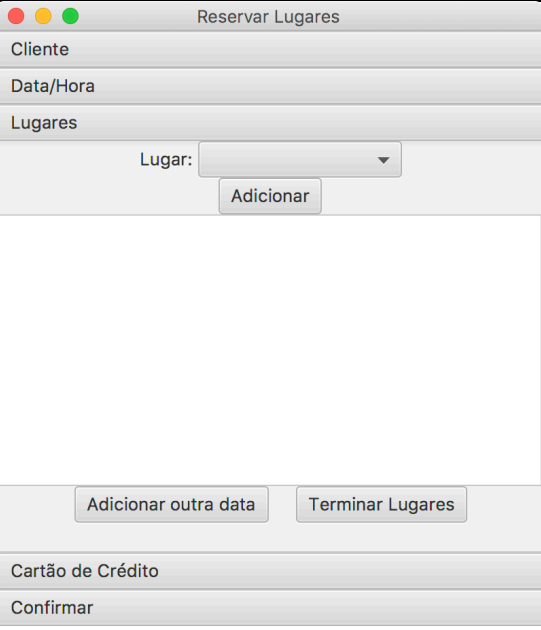
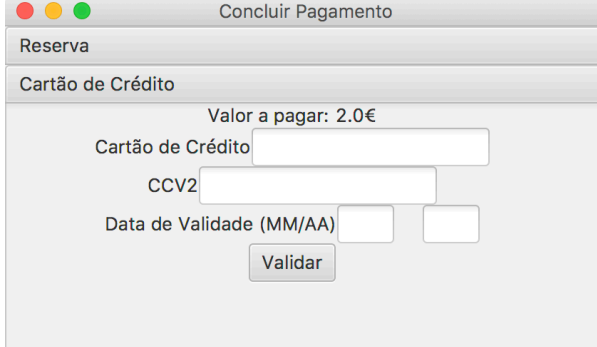
Utilizador	Password	Tipo de Utilizador
admin	admin	Administrador
serafim	serafim	Funcionário (Turno das 00h às 23h59)
zacarias	zacarias	Funcionário (Turno das 13h às 20h)
ana	ana	Cliente Final
maria	maria	Cliente Final

No ecrã principal, poderá fazer os 3 casos de usos principais desenvolvidos na Fase 2. A opção de criar grelha ficará desativada caso o utilizador não seja administrador.

Na opção de criar grelha, será pedido o nome da grelha e o índice correspondente (número decimal).

Nos passos seguintes serão pedidas as dimensões da sala em lugares (altura e largura). Posteriormente será pedido um Tipo Padrão (por defeito não existe nenhum).

Finalmente será possível definir para cada lugar (pela sua coordenada) o tipo de lugar a usar, usando o botão “Definir Lugar”. Assim que estiver concluído este processo, poderá usar o botão “Terminar” para voltar ao menu principal.

	<p>No ecrã de “Reservar Lugares” é possível começar por indicar o cliente a quem a reserva pertence. No caso do utilizador autenticado ser Cliente Final, este passo não ocorre, sendo o próprio utilizador o dono da reserva.</p> <p>No segundo passo é escolhido o dia e hora da reserva (no futuro).</p> <p>No terceiro passo (mostrado ao lado), é possível escolher diferentes combinações de tipo de lugar/grelha. Na listagem no meio aparece cada lugar reservado (terão de ser diferentes!).</p> <p>É possível voltar ao passo 2 usando o botão “Adicionar outra data”, ou avançar para pagamento com “Terminar Lugares”.</p> <p>No penúltimo passo é possível dar os dados do cartão de crédito (sendo que o ccv2 tem de ter 3 dígitos, e a data de expiração tem de ser no futuro).</p> <p>No último passo é mostrado o preço total a pagar, que se pode fazer no botão Confirmar.</p>
	<p>Neste ecrã é possível escolher a reserva a terminar o pagamento (Por <i>default</i> existe uma reserva R0 para testes).</p> <p>No segundo passo é mostrado o valor em falta e é possível indicar os dados do cartão de crédito a usar.</p>

## Anexo III – API MonsterCard

- 1) Exemplo do uso da API para verificar a validade de um cartão:

```
Card c = new Card("1234123412341234", "123", "05", "18");
MonsterCardAPI m = new MonsterCardAPI();
boolean ok = m.isValid(c);
```

- 2) Exemplo do uso da API para cativar 10 euros no cartão:

```
m.block(c, 10)
```

- 3) Exemplo do uso da API para cativar 10 euros no cartão:

```
m.charge(c, 10)
```

## Anexo IV – API PortugueseExpress

- 1) Exemplo do uso da API para verificar a validade de um cartão:

```
PortugueseExpress api = new PortugueseExpress();
api.setNumber("1234123412341234")
api.setCcv(123)
api.setMonth(5)
api.setYear(2018)
boolean ok = api.validate()
```

- 
- 1) Exemplo do uso da API para cativar 10 euros no cartão:

```
api.block(10);
```

- 2) Exemplo do uso da API para cativar 10 euros no cartão:

```
api.charge(10);
```