Programação Orientada por Objetos

Desenho de aplicações

Prof. José Cordeiro,

Prof. Cédric Grueau,

Prif. Laercio Júnior

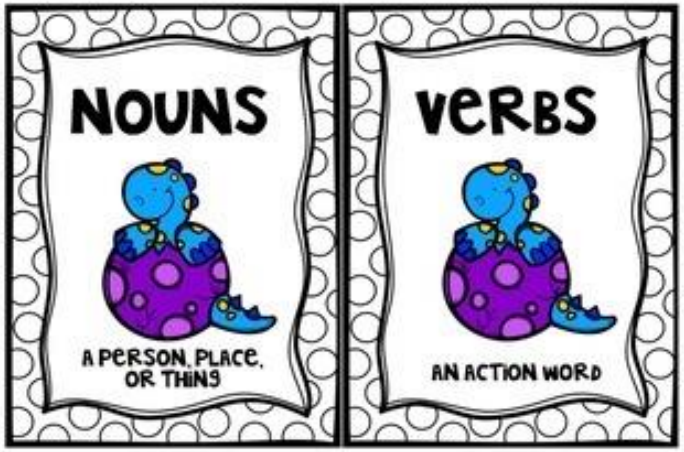
Departamento de Sistemas e Informática

Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal

2019/2020

Módulo Desenho de Aplicações

- Sessão 1: Método dos verbos/substantivos
- Sessão 2: Cartas CRC e Cenários
- Sessão 3: Outros Aspetos do Desenvolvimento
- Sessão 4: Padrões de Desenho



Módulo 4 – Desenho de aplicações

SESSÃO 1 – MÉTODO VERBOS/SUBSTANTIVOS

Exemplo – Sistema de reservas

- Sistema de reserva de lugares para cinemas.
  - O sistema de reservas do cinema deve guardar as reservas de lugares para várias salas.
  - Cada sala tem os lugares dispostos por filas.
  - Os clientes podem reservar os lugares e ficam com a letra da fila e o número do lugar.
  - Eles podem solicitar a reserva de vários lugares adjacentes.
  - Cada reserva é para uma determinada sessão (ou seja a projeção de um dado filme a uma certa hora).
  - As sessões têm uma data e hora e estão escalonadas para uma determinada sala.
  - O sistema guarda o número do telefone do cliente.

Exemplo – Sistema de reservas

- Diagrama de classes da aplicação **Sistema de reservas**:

Exemplo – Sistema de reservas

- Quais são as classes da aplicação?
  - Até agora de uma forma ou de outra tínhamos as classes da solução. Num **projeto de software** real temos de encontrar essas classes, o que não é uma tarefa simples.
  - Os passos iniciais do desenvolvimento de software são a **análise e o design**.
    - Analisa-se o problema.
    - Desenha-se a solução (o design da solução).
  - Não existe um método bem definido e único para encontrarmos as classes da solução. Algumas abordagens simples são:
    - O método **verbos/substantivos**.
    - A utilização de **cartas CRC**.

Método verbo/substantivo

- Quais são as classes da aplicação?
  - Os **substantivos** descrevem "coisas".
    - São uma fonte de referência de **classes e objetos**.
  - Os **verbos** referem ações.
    - Podem ser a fonte para encontrarmos as **interações** entre objetos.
    - Como as ações representam comportamentos, logo referem **métodos**.

Exemplo – Sistema de reservas

Verbos e substantivos

- O sistema de reservas do cinema deve guardar as reservas de lugares para várias salas.
- Cada sala tem os lugares dispostos por filas.
- Os clientes podem reservar os lugares e ficam com a letra da fila e o número do lugar.
- Eles podem solicitar a reserva de vários lugares adjacentes.
- Cada reserva é para uma determinada sessão (ou seja a projeção de um dado filme a uma certa hora).
- As sessões têm uma data e hora e estão escalonadas para uma determinada sala.
- O sistema guarda o número do telefone do cliente.

Exemplo – Sistema de reservas

- **Verbos** e **substantivos**
  - Vamos procurar os nomes (os substantivos) que nos irão dar uma pista dos objetos e classes envolvidos.
  - Depois procuramos os verbos juntamente com os substantivos a que se referem.

Exemplo – Sistema de reservas

Substantivos

- O **sistema de reservas** do **cinema** deve guardar as **reservas** de **lugares** para várias **salas**.
- Cada **sala** tem os **lugares** dispostos por **filas**.
- Os **clientes** podem reservar os **lugares** e ficam com a **letra** da **fila** e o **número** do **lugar**.
- Eles podem solicitar a **reserva** de vários **lugares** adjacentes.
- Cada **reserva** é para uma determinada **sessão** (ou seja a **projeção** de um dado **filme** a uma certa **hora**).
- As **sessões** têm uma **data** e **hora** e estão escalonadas para uma determinada **sala**.
- O **sistema** guarda o **número** do **telefone** do cliente.

Exemplo – Sistema de reservas

Verbos e substantivos

- O **sistema de reservas** do **cinema** deve **guardar** as **reservas** de **lugares** para várias **salas**.
- Cada **sala tem** os **lugares** dispostos por **filas**.
- Os **clientes** podem **reservar** os **lugares** e **ficam** com a **letra** da **fila** e o **número** do **lugar**.
- Eles podem **solicitar** a **reserva** de vários **lugares** adjacentes.
- Cada **reserva é** para uma determinada **sessão** (ou seja a **projeção** de um dado **filme** a uma certa **hora**).
- As **sessões têm** uma **data** e **hora** e **estão** escalonadas para uma determinada **sala**.
- O **sistema guarda** o **número** do **telefone** do cliente.

Exemplo – Sistema de reservas

- **Verbos** e **substantivos**

Sistema de reservas
Guarda (reserva de lugares)
Guarda (número telefone)

Sala
Tem (lugares)

Filme

Cliente
Reserva (lugares)
Ficam com (letra fila, número lugar)
Solicitam (reserva de lugar)

Hora

Data

Reserva de lugar

Sessão
Escalonada para (sala)

Lugar

Número telefone

Fila

Módulo 4 – Desenho de aplicações

SESSÃO 2 – CARTAS CRC E CENÁRIOS

Cartas CRC

- Cartas **CRC**
  - CRC significa:
    - **C** – **Classes**
    - **R** – **Responsabilidades**
    - **C** - **Colaboradores**
  - Utilizam-se cartas reais divididas em áreas para cada um dos componentes identificados.
  - Este método foi inicialmente descrito por Kent Beck e Ward Cunningham.

Cartas CRC

- A carta **CRC**

Classe (nome)	**Colaboradores**
Responsabilidades	

Cenários

- Depois de se obterem as possíveis classes do sistema e que são identificadas nas cartas CRC é necessário descobrir as interações entre elas e isso será feito através de **Cenários**.
- Um **cenário** descreve um caso concreto da realização de uma atividade no sistema.
  - Vamos identificar o que se faz (em cada classe) quando se utiliza o sistema.
  - É uma forma de encontrar as **interações entre objetos** (colaborações).
  - Pode ser feito como uma **atividade de grupo**.

Cenários – exemplo prático

SistemaReservas	Colaboradores
Pode encontrar as sessões pelo título do filme e pelo dia	**Sessão**
Guarda coleções de sessões	**Coleção**
Obtém e mostra os detalhes das sessões	
…	

Cenários

- Os cenários permitem verificar se a descrição do problema é clara e se está completa.
- É necessário perder algum tempo a encontrar os vários cenários de utilização.
- A análise efetuada do problema leva-nos à solução
  - Se encontrarmos os erros e omissões nesta fase estaremos a poupar tempo e esforço desperdiçados mais tarde.

Desenho de classes

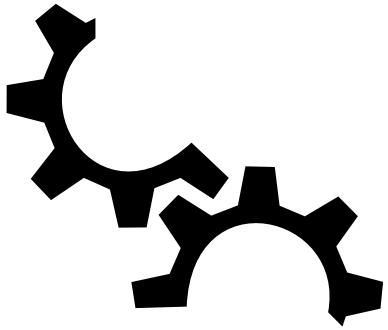
- A análise dos cenários ajuda a clarificar a estrutura da aplicação
  - Cada carta está relacionada com uma classe
  - As colaborações revelam a cooperação entre classes/interação entre objetos.
- As responsabilidades revelam métodos públicos.
  - E algumas vezes atributos; por exemplo: "guarda *coleções*..."

Desenho de classes

- As responsabilidades devem ser atribuídas/implementadas seguindo os princípios do desenho de classes estudado anteriormente:
  - **Coesão**
  - **Acoplamento**
  - **Desenho Orientado por responsabilidades**

Desenho da interface das classes

- **Depois de encontrar as classes**, responsabilidades e colaboradores é ainda necessário traduzir as descrições informais para chamada de **métodos e parâmetros** dos mesmos.
  - Repetir a análise de cenários tendo em conta agora as chamadas de métodos, passagem de parâmetros e valores de retorno.
  - Anotar as assinaturas dos métodos obtidas.
- Criar a **estrutura das classes com os métodos públicos** ainda sem o código interno.
- Um desenho cuidadoso é a chave do sucesso da implementação.



Módulo 4 – Desenho de aplicações

SESSÃO 3 – OUTROS ASPETOS DO DESENVOLVIMENTO

- **Documentação**
  - Escrever os comentários das classes.
  - Escrever os comentários dos métodos.
  - Descrever claramente qual a finalidade das classes e métodos.
  - A documentação nesta altura assegura que:
    - O foco é **no que faz** e não em **como se faz**.
    - E que isso não é esquecido.

Cooperação

- **Trabalho de equipa**
  - O trabalho de equipa irá provavelmente ser a regra e não a exceção.
  - A documentação é essencial para o trabalho de equipa.
  - Um desenho orientado por objetos claro com componentes fracamente acoplados suporta igualmente a cooperação.

Protótipos

- **Prototipagem**
  - Devem-se construir protótipos da aplicação.
    - Os protótipos são versões da aplicação onde uma parte é simulada para que se possa experimentar com outras partes.
  - Os protótipos ajudam no inicio na análise do sistema.
    - Permitem uma identificação inicial do problema.
  - Os componentes ainda incompletos podem ser simulados.
    - Por exemplo retornando valores fixos.
    - É de evitar comportamentos aleatórios que são difíceis de prever.

Desenvolvimento de software

- O modelo **Waterfall**
  - **Análise**
  - **Design**
  - **Implementação**
  - **Testes** unitários e de integração
  - **Instalação**
- É o modelo de desenvolvimento mais conhecido e tradicional.
  - Não é o mais usado.
  - Parte do princípio que os programadores conhecem à partida todas as funcionalidades em detalhe e que o sistema não será alterado depois de entregue.

Desenvolvimento de software

- O modelo **Iterativo**
- Baseia-se em prototipagem, interações com o cliente e pequenos ciclos de desenvolvimento iterativos e incrementais
  - **Iterações com:**
    - **Análise**
    - **Design**
    - **Protótipo**
    - **Feedback do cliente**
- É, talvez, o modelo de desenvolvimento mais comum atualmente.
  - É um modelo mais realístico.
  - O software vai *crescendo* em vez de ser desenhado.

Módulo 4 – Desenho de aplicações

SESSÃO 4 – PADRÕES DE DESENHO

Utilização de padrões de desenho

- Tendo em conta que:
  - As relações entre classes são bastante importantes e podem ser complexas.
  - Algumas relações que se estabelecem entre classes ocorrem em várias aplicações.
- Existem **padrões de desenho** de classes que ajudam a identificar e clarificar as relações entre classes
  - Conhecidos como **Software Design Patterns**.
  - São descrições de soluções recorrentes para problemas recorrentes.
  - São problemas estudados com boas soluções que são exemplos de boas práticas no desenvolvimento de software.
  - São descritos com base numa estrutura (tal como as cartas CRC) e que foi inicialmente usada no campo da arquitetura.

Estrutura dos padrões de desenho

- **Software Design Patterns**:
  - **Nome** do padrão
  - **Problema abordado** pelo padrão
  - Como é proposta a **solução**
    - Estrutura
    - Participantes
    - Colaborações
  - **Consequências** da solução
    - Resultados e compromissos

Padrão Singleton

- Padrão **Singleton**
  - É um **padrão criacional**.
    - Lida com o problema de existir apenas uma única instância de uma dada classe.
  - Todos os clientes utilizam o mesmo objeto.
  - A solução neste caso é tornar o construtor privado para impedir que se criem objetos externamente.
  - É obtida a única instância existente através de um método estático `getInstance()`
  - Exemplo: A classe `Canvas` de um objeto **Shape**.

Exemplo – Padrão Singleton

Padrão **Decorator**

- É um **padrão estrutural**.
  - Lida com o problema de adicionar funcionalidade a um objeto existente sem utilizar a herança de classes.
- Os clientes deste padrão requerem um objeto de um determinado tipo (uma interface ou uma superclasse) mas com funcionalidades extra.
- A solução é criar um objeto que inclui (*wraps*) o objeto que se pretende utilizar e que será utilizado em vez do original.
  - O novo objeto inclui a funcionalidade que se pretende invocando-a diretamente no objeto incluído e adicionalmente tem os métodos que aumentam a sua funcionalidade.
- Exemplo: A classe **`BufferedReader`** é um **Decorator** da classe **`Reader`**.
  - Pode ser usada em vez da classe **`Reader`** porque inclui a mesma funcionalidade mas tem um comportamento adicional ao da classe original.

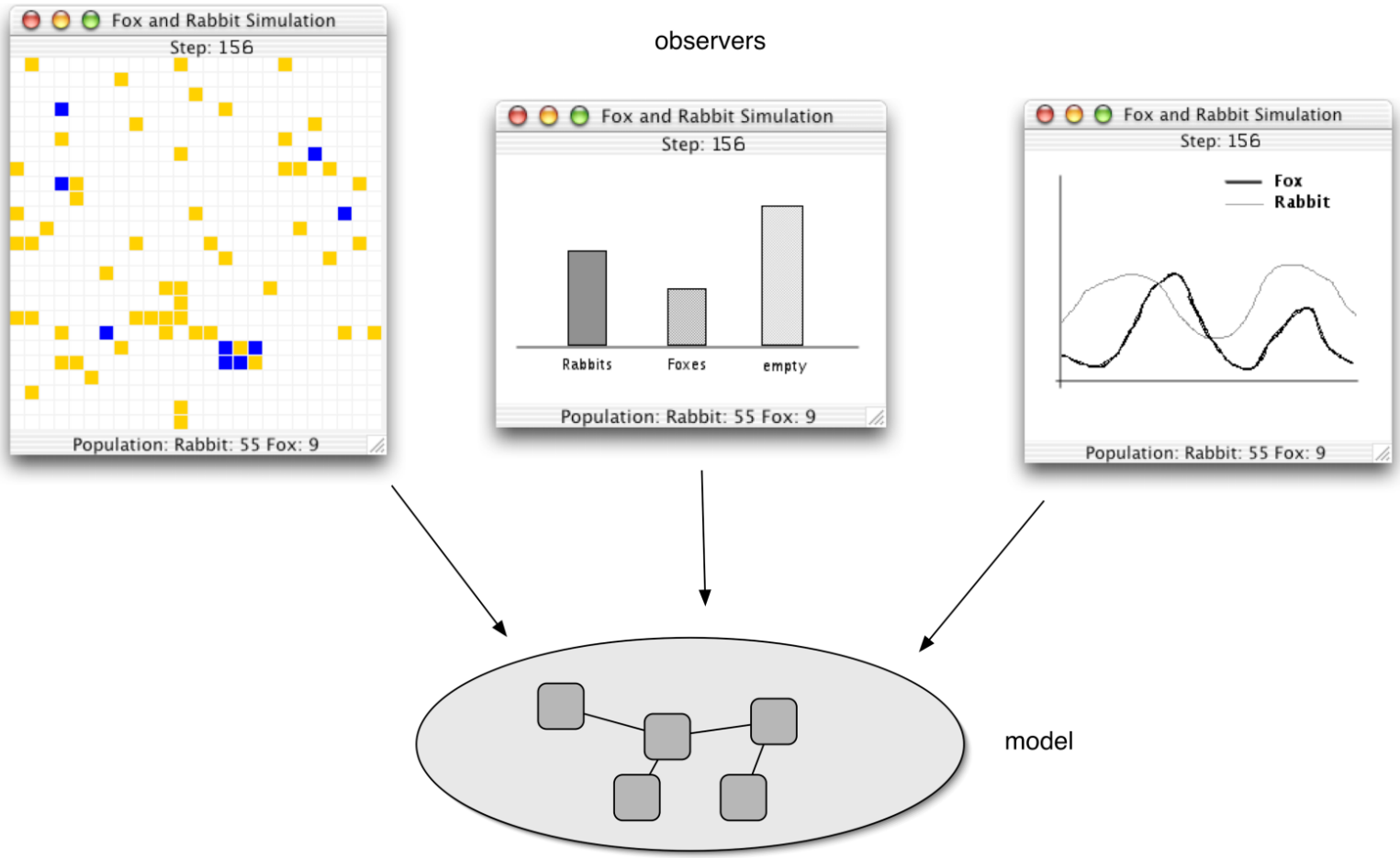
Padrão Factory Method

- Padrão **Factory Method**
  - É um **padrão criacional**.
    - Lida com o problema da criação de um objeto sem especificar concretamente qual a classe que vai ser utilizada.
  - Os clientes deste padrão requerem um objeto de um determinado tipo: de uma dada interface ou superclasse.
  - A solução é criar um método que tem liberdade para devolver um objeto de qualquer classe que implemente essa interface ou que seja derivado dessa superclasse.
    - O tipo exato do objeto depende do contexto.
  - Exemplo: Os métodos `iterator` das classes de coleção.

Padrão Observer

- Padrão **Observer**
  - É um **padrão comportamental**.
    - Lida com o problema de separar o modelo interno de uma vista desse modelo.
  - Define uma relação de um para muitos entre objetos.
  - O objeto observado notifica todos os objetos observadores de qualquer alteração no seu estado.
  - Uma classe ou um conjunto de classes incluem a lógica e os dados do programa e permitem a existência de observadores desses dados. Depois são criadas visualizações independentes da lógica que são atualizadas sempre que mudam os dados.
    - Podemos ter várias visualizações que ficam independentes dos dados.
  - Exemplo: A classe `SimulatorView` do projeto *Foxes-and-rabbits*.

- Padrão **Observer**



Bibliografia

- Objects First with Java (6th Edition), David Barnes & Michael Kölling, Pearson Education Limited, 2016
  - Capítulo 15