Análise Orientada a Objectos

- Antes de poder codificar a solução para um dado problema é necessário fazer uma correcta definição das classes necessárias, das responsabilidades de cada uma e dos esquemas de colaboração entre elas
  - Esta fase de análise é tanto mais importante quanto maior for o software a desenvolver
  - A codificação de cada classe deve começar apenas após uma análise correcta e pode decorrer independentemente relativamente a cada classe, uma vez que estas devem ser entidades autónomas umas das outras e do contexto em que são utilizadas

Análise Orientada a Objectos

- Esta fase ocorre tipicamente em colaboração com o cliente e/ou com alguém com conhecimento do domínio em causa
- O objectivo da fase de análise é criar um modelo abstracto do software utilizando linguagem do cliente (ou do domínio). Geralmente podem identificar-se três passos:
  - Identificar as classes que modelam o sistema formando um conjunto de abstracções natural e sensato
  - Determinar o objectivo, ou responsabilidade principal, de cada classe.
    - As responsabilidades de uma classe indicam o que as instâncias dessa classe devem saber fazer (comportamentos) e que informação devem manter (atributos)
  - Determinar quais as colaborações (ajudas) que cada classe necessita
    - Muitas vezes as classes delegam parte das suas responsabilidades em objectos de outras classes (*collaborators* ou *helpers*)

Análise Orientada a Objectos

- A análise é geralmente feita por equipas que integram especialistas em análise, mas também especialistas do dominio a ser modelado
  - O trabalho destas equipas passa por várias fases, sendo que num primeiro momento apenas as classes e responsabilidades mais importantes e evidentes são identificadas
  - Em fases posteriores o trabalho é refinado, levando frequentemente à identificação de novas classes e/ou responsabilidades

Análise Orientada a Objectos

- Identificação de classes
  - Uma forma simples de começar esta fase é escrever os substantivos que se encontram na descrição do problema
    - Podem ser considerados como classes potenciais que modelam parte do sistema
  - Escrever também outras potenciais classes necessárias para modelar o sistema
    - Alguém com experiência da área pode sugerir classes que venham a ser necessárias
    - Outras classes podem ser identificadas através de palavras que são ditas ao descrever, clarificar ou discutir o problema
  - Nesta fase a equipa não se deve preocupar demasiado em verificar a adequação das classes propostas
  - Em suma, as classes podem ser identificadas a partir da descrição do problema, do conhecimento sobre o domínio e da discussão entre os membros da equipas de análise

Análise Orientada a Objectos

- Geralmente é preferível que as classes modelem directamente o mundo real
  - Facilita o trabalho de análise
  - Facilita o trabalho posterior de manutenção ou actualização do software
- Uma vez identificada uma lista de classes, é necessário dividi-las em função da sua adequação ao problema
  - Classes que seguramente são necessárias
  - Classes que talvez venham a ser necessárias
  - Classes desnecessárias ou que podem complicar a solução

Análise Orientada a Objectos

- Eliminar classes redundantes
  - Identificadas mais do que uma vez
  - Considerar apenas os substantivos essenciais ao problema
  - Eliminar classes que embora façam sentido não fazem parte do núcleo central do sistema
- Escrever um pequeno texto sobre cada classe da lista resultante e o seu papel no sistema

Análise Orientada a Objectos

- Após a identificação das classes e da responsabilidade principal de cada uma é possível passar a uma fase de detalhe do modelo
  - Outras responsabilidades de cada classe
  - Colaborações necessárias para que uma classe desempenhe o seu papel
    - **Um sistema é um conjunto de objectos independentes que desempenham o seu papel para um fim comum. Não deve haver classes "todo poderosas" a tentar fazer tudo**
- É importante colocar questões como:
  - Quais as responsabilidades desta classe?
  - Que classe é responsável por determinada acção ou possui determinado conhecimento?

Análise Orientada a Objectos

- Nesta fase, a equipa pretende responder a questões como:
  - Que outras responsabilidades deve a classe assumir?
  - Que classe deve assumir uma dada responsabilidade?
  - Que classes ajudam outras a cumprir as suas responsabilidades?
- Para isto os membros da equipa devem sempre ter presente:
  - O que é que cada objecto de uma classe sabe
  - O que é que cada objecto de uma classe é capaz de fazer
  - Antropomorfismo

Análise Orientada a Objectos

- Quando uma classe tem uma dada responsabilidade todos os objectos instanciados a partir dela têm essa mesma responsabilidade
- As responsabilidades podem ser identificadas a partir de várias fontes
  - A especificação do problema
  - As classes já existentes
  - As ideias geradas pela discussão na equipa
- Quando um responsabilidade é identificada deve ser atribuída a uma das classes já existentes ou a uma nova classe se necessário

Análise Orientada a Objectos

- Nesta fase é comum que as equipas utilizem técnicas de *role playing* e que efectuem a representação das classes através de CRC cards (*class-responsability-collaboration*)
  - Como o nome indica, um CRC card (há quem lhes chame CRH Card, *helper* em vez de *collaborator*), é uma ficha em papel onde se escreve o nome da classe, as suas responsabilidades principais e as suas colaborações

Análise Orientada a Objectos

Class: cdCollection	
Responsibilities:	**Helpers:**
know all CDs	vector of CDs
retrieve a CD	
addNewCD(CD aCD, int trayNumber)	
removeCD(int trayNumber)	
CD getCD(int trayNumber)	

Análise Orientada a Objectos

- Embora neste exemplo se incluam valores de retorno e parâmetros, muitas vezes as equipas optam por incluir apenas a lista de responsabilidades que a classe deve assumir
- A maior parte das responsabilidades mais tarde serão traduzidas em métodos, embora alguma possam ser traduzidas em atributos (por exemplo “know all CDs”)

Análise Orientada a Objectos

- Nas actividades de role play cada elemento da equipa assume o papel de uma ou mais classes
- São colocados cenários (O que acontece se ...?) e cada membro da equipa tenta desempenhar o papel da(s) sua(s) classe(s) a partir daí
- Estas actividades tendem à identificação de novas responsabilidades que são adicionadas aos cartões
- Estas actividades prosseguem até que tenham sido analisados diversos cenários previsíveis e a equipa esteja convencida que o sistema responde bem a qualquer deles

Análise Orientada a Objectos

- Após a análise de diversos cenários os cartões devem ser dispostos sobre uma mesa para se obter uma representação visual do sistema
  - Cartões relacionados por relações hierárquicas devem ser sobrepostos parcialmente
  - Cartões que representam colaboradores devem ser colocados perto
  - A disposição dos cartões permite verificar os agrupamentos lógicos e também identificar classes que pareçam ter demasiado trabalho e outras que participem pouco no sistema
    - Estas situações poderão ter que ser corrigidas pela equipa

Análise Orientada a Objectos

- Mais informação:
  - A proposta original dos CRC Cards
  - http://c2.com/doc/oopsla89/paper.html
  - Um tutorial resumido
  - http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/crc/
  - Um excelente texto com a descrição detalhada de um exemplo
  - http://www.cs.arizona.edu/people/mercer/design/12.pdf

UML

- O desenvolvimento de programas orientados aos objectos assenta na construção de um modelo
- O modelo é uma abstração dos aspectos essenciais do problema
- UML (Unified Modeling Language) inclui um conjunto de ferramentas criadas para representar esses modelos
- Nesta disciplina vamos utilizar apenas alguns aspectos básicos, ficando para outras disciplinas uma abordagem mais detalhada

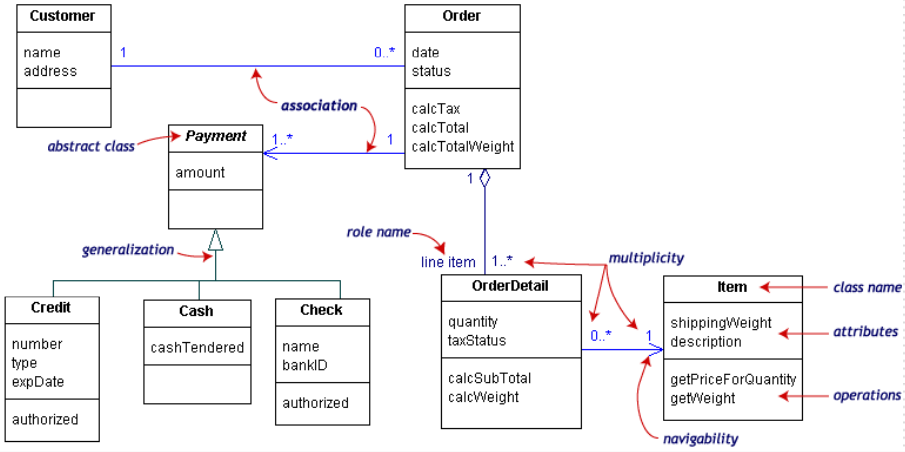
UML

- Em UML os modelos de software são representados por um conjunto de diagramas
- Estes são criados e utilizados durante a fase de análise e, posteriormente, como ferramenta de comunicação com os programadores
- Os modelos são constituídos por objectos (com os seus atributos e comportamentos) que interactuam entre si através do envio de mensagens

UML – Diagramas de classes

- Um diagrama de classes dá uma panorâmica geral do sistema, mostrando as suas classes e as relações entre elas
- Os diagramas de classe são estáticos, ou seja mostram o que interage, mas não o que acontece quando a interacção acontece

UML – Diagramas de classes



UML – Diagramas de classes

- Uma classe é representada por um rectângulo dividido em 3 partes, nome da classe, atributos, comportamentos
- No caso de classes abstractas o nome aparece em itálico (*Payment* no exemplo)
- As relações entre classes são representadas por linhas. No exemplo aparecem 3 tipos de relações:
  - **Associação** – há uma associação entre duas classes se uma instância de uma classe precisa de conhecer a outra para desempenhar o seu papel. Num diagrama, uma associação é uma ligação entre duas classes
  - **Agregação** – uma associação em que uma das classes pertence a uma colecção. Num diagrama, uma agregação termina em forma de diamante apontando para a classe que contém o conjunto. No exemplo, Order tem uma colecção de OrderDetails

UML – Diagramas de classes

- **Generalização** – uma associação em que há uma relação de herança, indicando que uma das classes é superclasse da outra. Uma generalização termina num triângulo que aponta para a superclasse. No exemplo, *Payment* é uma superclasse de Cash, Check e Credit

- Os extremos das associações podem ter uma indicação de função para clarificar a natureza da associação. Por exemplo, OrderDetail é um "line item" de cada Order

UML – Diagramas de classes

- As setas de navegabilidade em cada associação indicam em que direcção a associação pode ser utilizada. Por exemplo, pode perguntar-se a uma OrderDetail pelo seu Item, mas não o oposto. A seta permite ainda identificar a quem cabe a implementação da associação; neste caso OrderDetail tem um Item. Associações sem setas são bidireccionais

UML – Diagramas de classes

- A cardinalidade de um extremo de uma associação é o número de possíveis instâncias da classe nesse extremo associadas com uma única instância da classe no outro extremo
  - Cardinalidade é indicada por números ou por intervalos. No exemplo, só pode haver um Customer para cada Order, mas um Customer pode ter qualquer número de Order.

UML – Diagramas de classes

- Algumas cardinalidades comuns

Cardinalidade	Significado
0..1	0 ou 1 instância (n..m significa n a m instâncias)
0..* ou *	Qualquer número de instâncias (mesmo zero)
1	Exactamente uma instância
1..*	Pelo menos uma instância

UML – Diagramas de classes

- Todos os diagramas de classe incluem classes, associações e cardinalidade
- Navegabilidade e papeis são opcionais para aumentar a clareza do diagrama

UML

- O UML inclui outros tipos de diagramas:
  - Use case diagrams
  - Class diagrams
  - Object diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Statechart diagrams
  - Activity diagrams
  - Component diagrams
  - Deployment diagrams
- Uma introdução sucinta pode ser encontrada em http://bdn.borland.com/article/0,1410,31863,00.html