DESENHO DE SOFTWARE
Software Design

pmorais@upt.pt

- "There are two ways of constructing a software design;
  one way is to make it so simple that there are obviously no deficiencies,
  and the other way is to make it so complicated that there are no obvious deficiencies.
  The first method is far more difficult." - C. A. R. Hoare

- Weinberg's Law: If builders built buildings the way programmers wrote programs, the first woodpecker that came along would destroy civilization.

Software muda.../Software changes...

- Software não é como uma novela que é escrita uma vez e nunca mais muda
- *Software is not like a novel that is written once and then remains unchanged.*

- Software é corrigido, mantido, adaptado, evolui,…
- *Software is extended, corrected, maintained, ported, adapted…*

- Software é construido por diferentes pessoas, ao longo do tempo (anos…)
- *The work is done by different people over time (often decades).*

- O software:
  - Ou é continuamente mantido
  - Ou morre
- *There are only two options for software:*
  - *Either it is continuously maintained*
  - *or it dies.*

- Software que não pode ser mantido será abandonado
- *Software that cannot be maintained will be thrown away.*

Qualidade no Desenho de software
Quality in software design

- Começando pelo desenho de classes…
- Como escrever classes de forma a que sejam facilmente
  - Entendidas
  - Mantidas
  - Reutilizadas?
- *How to write classes in a way that they are easily understandable, maintainable and reusable*

Desenho de classes/ Designing classes

- Factores que influenciam o desenho?
- O que torna o desenho de uma classe bom ou mau?
- *What makes design good or bad?*
- Desenhar "bem" classes pode demorar mais do que desenhar "mal", mas a longo prazo o esforço compensa
- Tempo necessário para modificar um código mal desenhado > tempo necessário para modificar um código bem desenhado

Buzzwords

Desenho orientado por responsabilidades/responsibility-driven design

Encapsulamento/encapsulation

Herança/inheritance

Acoplamento/coupling

Sobreposição/overriding

interface

Coesão/cohesion

Sobrecarga/overloading

Classes abstractas/Abstract classes

Métodos mutantes /mutator methods

polimorfismo/polymorphism

Qualidade no desenho de classes/Code quality
Principais conceitos/Main points

- Acoplamento (coupling)
  - Interligação entre as classes (*links between separate units of a program*)
  - Encapsulamento *(encapsulation)*
- Coesão (cohesion)
  - Nº e diversidade de tarefas pelas quais é responsável uma unidade de uma aplicação (*refers to the number and diversity of tasks that a single unit is responsible for*)
- Desenho orientado por responsabilidade (Responsibility-driven design)
- Refactoring
- Localizar a mudança (Localizing change)

Acoplamento/Coupling

- Pretende-se um baixo grau de acoplamento
- We aim for *loose coupling*

- If two classes depend closely on many details of each other, we say they are *tightly coupled*
  - Classes independentes e que comunicam umas com as outras através de uma interface bem definida

- Grau de acoplamento determina a dificuldade em fazer alterações numa aplicação

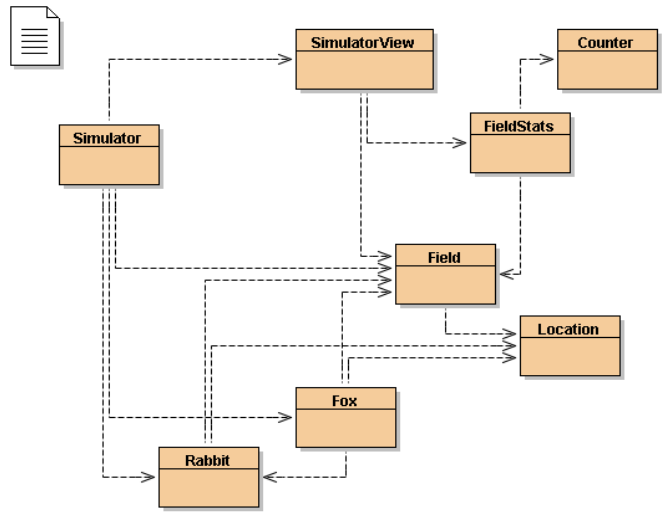
- Correcto encapsulamento nas classes reduz acoplamento
  - Apenas o que a classe faz deve ser visível do exterior, não como faz

Baixo Acoplamento/ Loose coupling

- Baixo acoplamento possibilta
  - Perceber uma classe sem ler as outras;
  - Alterar uma classe sem afectar outras;
    - » Melhora manutenção

- *Loose coupling makes it possible to:*
  - *understand one class without reading others;*
  - *change one class without affecting others.*
    - » *Thus: improves maintainability*

Acoplamento/Coupling
ligação entre classes – que diz este diagrama? – *what can we say about this diagram?*



Fragmento de código que define as variáveis de instância de uma classe Aluno, e parte de um método, de uma outra classe que actualiza o ano do aluno.

- **class Student {**

```
public String name;
public int year;
public int code;
public String course;

public Student (String n, String l, int a, int c){
        name = n;
        course = l;
        year = a;
        code = c;}
}
```

Alterações a fazer ao código apresentado para diminuir o acoplamento?

How to change the code in order to decrease coupling?

- **class StudentManagement**

```
…
Public changeYear(int newYear, Student refStudent){
refStudent.year = newYear;}
```

Coesão/*Cohesion*

- Pretende-se um alto grau de coesão
- *We aim for high cohesion*

- Uma unidade de código deve executar uma única tarefa
- *A method should be responsible for one and only one well defined task*
  - Um método deve implementar uma operação
  - Uma classe deve representar um tipo de entidade
  - *Classes should represent one single, well defined entity*

- If each unit is responsible for one single logical task, we say it has *high cohesion*
- Cohesion applies to classes and methods

Coesão/*Cohesion*

- Princípio da reutilização – se um método ou classe é responsável por uma única coisa bem definida, então terá maior probabilidade de poder ser usado num outro contexto
- *High cohesion makes it easier to reuse classes or methods.*

- Duplicação de código – mau desenho
  - Inconsistências
  - Normalmente um sintoma de má coesão

- *Code duplication*
  - *is an indicator of bad design,*
  - *makes maintenance harder,*
  - *can lead to introduction of errors during maintenance.*

Desenho orientado por responsabilidade
Responsability - driven design

- Responsability - driven design – ideia de que a classe deve ser responsável por manipular os seus próprios dados
- *Each class should be responsible for manipulating its own data.*
  - Outro aspecto que contribui para o grau de acoplamento
  - RDD leads to low coupling
- Quando é necessário adicionar alguma funcionalidade a uma aplicação, em que classe deve ser colocado o método que implementa essa funcionalidade?
- Question: where should we add a new method (which class)?
  - A classe que é responsável por armazenar alguns dados deve também ser responsável por manipulá-los
  - The class that owns the data should be responsible for processing it.

Refactoring

Processo de reestruturar um desenho existente (classes e métodos) para adaptá-los a alterações de requisitos, mantendo um bom desenho, facilitando (prevendo) alterações futuras

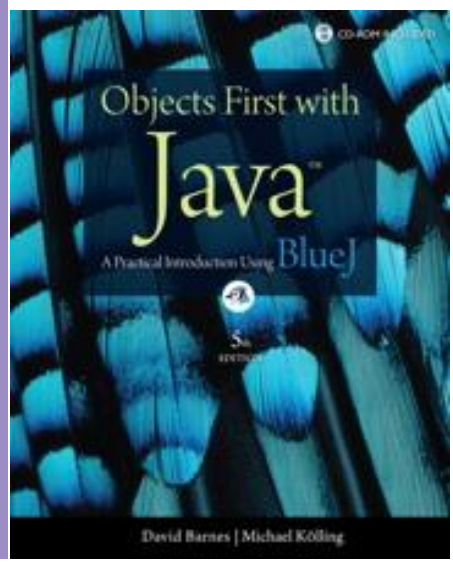
Activity of restructuring na existing design to maintain a good class design when the application is modified or extended

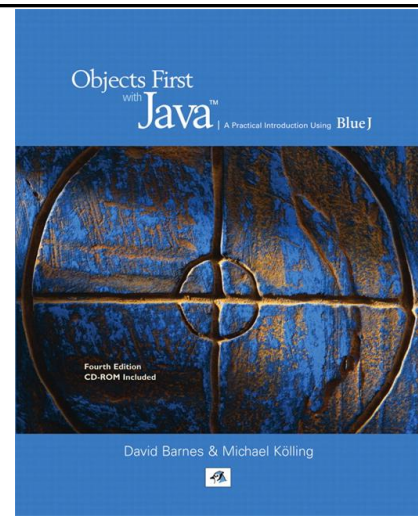
Localizar a mudança/ *localizing change*

- Criar um desenho de uma classe que facilite as alterações
- *When designing a class, we try to think what changes are likely to be made in the future.*
- *We aim to make those changes easy.*

- Alterações numa classe devem produzir o menor número de alterações nas outras classes
- *When a change is needed, as few classes as possible should be affected*
  - Pode conseguir-se com
    - » Baixo acoplamento
    - » Alta coesão
  - *One aim of reducing coupling and responsibility-driven design is to localize change*

Livro





Caso 1: Jogo world-of-zuul
Case 1: Game world-of-zuull

- Objectivos/Goals
  - Perceber conceitos de acoplamento e coesão
  - Understand coupling and cohesion

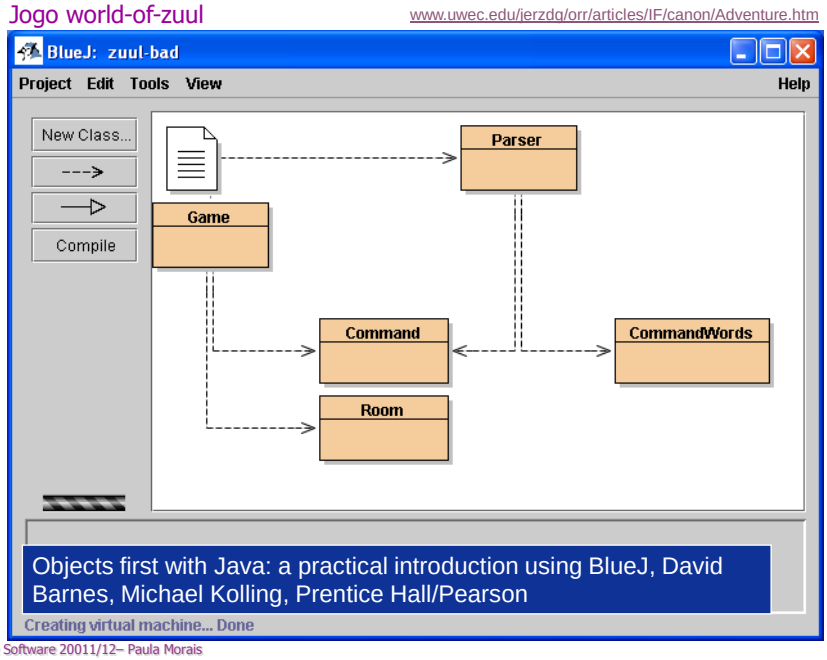
Fonte/Source: Cap 7 de Objects first with Java: a practical introduction using BlueJ, David Barnes, Michael Kolling, Prentice Hall/Pearson

Exemplo em BlueJ:

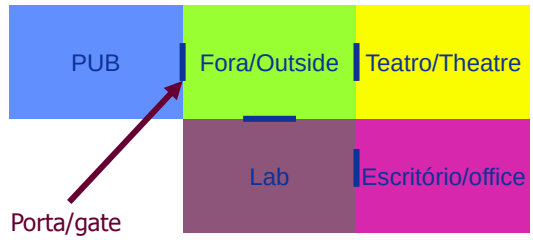
Jogo world-of-zuul

www.rickadams.org/adventure

www.uwec.edu/jerzdg/orr/articles/IF/canon/Adventure.htm



Mapa das salas existentes/ Rooms map



Desenho de classes - Duplicação de código

- Métodos printWelcome e goRoom contêm código duplicado
- *Methods printWelcome and goRoom have duplicate code*

- Ambos os métodos fazem 2 coisas:
  - printWelcome
    - » Imprime mensagem de boas vindas
    - » Imprime informação da localização corrente
  - goRoom
    - » Muda localização corrente
    - » Imprime informação da (nova) localização corrente

- Ambos os métodos imprimem informação da localização corrente, mas nenhum pode chamar o outro porque eles também fazem outras coisas – **MAU DESENHO*/BAD Design***
  - Desenhar um método mais coeso que apenas imprima a localização corrente e que possa ser reutilizado
    - » Desenhar método printLocationInfo()

Uso de Encapsulamento para reduzir acoplamento
Using encapsulation to reduce coupling

- Todas as variáveis *Exit* da classe Room são public
- *All Exit variables in class Room are public*

- Encapsulamento – esconder detalhes de implementação
- Encapsulating - Hiding implementation details
  - Apenas o que a classe faz deve ser visível do exterior, não como o faz
  - *Only information about what a class can do should be visible to the outside*

- Utilizar variáveis private e um método para as aceder
- *Make the fields private and use an accessor method to access them*

class Room

```
{
  private String description;
  private Room northExit;
  private Room southExit;
  private Room eastExit;
  private Room westExit;

Public Room getExit (String direction)
{   if (direction.equals("north"))
          return northExit;
    if (direction.equals("east"))
          return eastExit;
    if (direction.equals("south"))
          return southExit;
    if (direction.equals("west"))
          return westExit;
    return null;
}
} // end class Room
```

Class Game

```
…
Room nextRoom = null;
      if(direction.equals("north"))
         nextRoom = currentRoom.getExit("north");
      if(direction.equals("east"))
         nextRoom = currentRoom. getExit("ast");
      if(direction.equals("south"))
         nextRoom = currentRoom. getExit("south");
      if(direction.equals("west"))
         nextRoom = currentRoom. getExit("west");
```

Ou

```
Room nextRoom = null;
nextRoom = currentRoom. getExit(direction);
```

A representação interna da classe Room foi completamente desacoplada da interface

The internal representation in Room has been completely decoupled from the interface

```
private void printLocationInfo()
  {
  System.out.println("You are " + currentRoom.getDescription());
     System.out.print("Exits: ");
     if(currentRoom.getExit("north") != null)
        System.out.print("north ");
     if(currentRoom.getExit ("east")!= null)
        System.out.print("east ");
     if(currentRoom.getExit("south")!= null)
        System.out.print("south ");
     if(currentRoom.getExit("west")!= null)
        System.out.print("west ");
     System.out.println();
  }
```


Desenho de classes/Class Design

- Pretende-se adicionar uma nova direcção
  - Quatro direcções possíveis: N, S, E, W
  - Adicionar ***up*** e ***down***

- *We want to add a new direction of movement; currently, a player can move in four directions; N, S, E, W; we want to add up and down as possible directions*

- Quais as classes envolvidas?
- *Which classes do we need to change? Which changes do we need to do?*
  - Room e Game
  - Classe Game – vários sítios onde se enumeram *exits*, em cada um adicionar 2 casos novos (pelo menos…)
  - *At least two new cases...*
    - **MAU DESENHO/BAD DESIGN**

Desenho de classes – Acoplamento
Class Design - Coupling

- Utilizar HashMap para armazenar saídas (exits), em vez de variáveis separadas
- *Use a HashMap to store the exits, rather than separate variables*
  - » Qualquer nº de saídas
  - » *Any number of exists*
  - » Alteração na forma como uma sala (room) guarda a informação internamente sobre as salas vizinhas
  - » *This is a change in the way a room stores information internally abut neighboring rooms*
    - Mudança que afecta **a implementação** de Room (**como** a informação da saída está armazenada) **e não a interface** (**o que** a sala armazena)
    - *This is a change that should affect only **the implementation** of the Room class (**how** the exit information is stored) and **not the interface** (**what** the room stores)*

Acoplamento
Coupling

- Quando apenas a implementação de uma classe muda, as outras classes não são afectadas – caso de baixo grau de acoplamento
- *Ideally, when only the implementation of a class changes, other classes should not be affected – this would be a case of loose coupling*
  - Neste caso, é impossível – mudando apenas a classe Room, para utilização de um HashMap levaria a que a classe Game não compilasse – esta classe faz várias referências às variáveis de saída das salas – **Forte acoplamento** – **MAU DESENHO**
  - *In our example, this does not work. If we remove the exit varibles in teh Room class and replace them with a HashMap, the Game class will not compile any more – it makes numerous references to the room's exit variables – **Tight coupling** – **BAD DESIGN***

public String getExitString() – classe Room

```
{
// devolve uma string que descreve as saídas da sala, por exemplo, Exits:
    north west
   String exitString ="Exits: ";
   if(northExit !=null)
      exitString += "north ";
   if(eastExit !=null)
      exitString += "east ";
   if(southExit !=null)
      exitString += "south ";
   if(westExit !=null)
      exitString += "west ";
   return exitString;
}
```

private void printLocationInfo()

```
   {
   System.out.println("You are " + currentRoom.getDescription());
   System.out.println(currentRoom.getExitString());
 }
```


HashMap

- **Map** – colecção de pares de objectos chave/valor
  - Uma entrada é, não um objecto, mas um par de objectos
  - Utiliza-se o objecto chave para aceder ao objecto valor
    - » Exemplo: lista telefónica

- **HashMap** – implementação particular de um Map
  - Métodos *put* – insere uma entrada no mapa
    e *get* – acede ao valor de uma dada chave

Exemplo:

```
HashMap listaTelefonica = new HashMap();
listaTelefonica.put ("ana", "912345678");
 listaTelefonica.put ("joão", "969876543");

String numero = (String) listaTelefonica.get("ana");
```

Class Room melhorada com HashMap
Class Room improved with HashMap

```
import java.util.HashMap;
class Room
{
    private String description;
     private HashMap<String, Room> exits; // armazena as saídas desta
                                                    sala.

    /**
     * Cria uma sala chamada "description". Inicialmente não tem saídas.
     * "description" é algo como "numa cozinha" ou "num jardim"
     */

    public Room (String description)
    {
        this.description = description;
        exits = new HashMap<String, Room>();
    }
```


Class Room melhorada com HashMap
Class Room improved with HashMap

```
/**
     * Define as saídas desta sala. Cada direcção ou leva a outra sala
     * ou é null (nenhuma saída ali).
     */
    public void setExits (Room north, Room east, Room south, Room west)
    {
        if(north != null)
            //   northExit = north;
            exits.put("north", north);
        if(east != null)
            //   eastExit = east;
            exits.put("east", east);
        if(south != null)
            //   southExit = south;
            exits.put("south", south);
        if(west != null)
            //   westExit = west;
            exits.put("west", west);
    }
```

Ainda pode melhorar

Class Room melhorada com HashMap
Class Room improved with HashMap

```
* Devolve a sala onde fica se vai desta sala na direcção "direction". Se não há
  * nenhuma sala nesta direcção, devolve null
 */
        public Room getExit(String direction)
       {
          //     if (direction.equals("north"))
          //         return northExit;
          //     if (direction.equals("east"))
          //         return eastExit;
          //     if (direction.equals("south"))
          //         return southExit;
          //     if (direction.equals("west"))
          //         return westExit;
          //     return null;
          return (Room) exits.get(direction);
      }
      /**

   * Devolve a descrição da sala (a que foi definida no constructor).
   */
  public String getDescription()
  {
     return description;
  }
```


- A única parte onde existe informação sobre as 4 saídas é no método

public void setExit (Room north, Room east, Room south, Room west)

método faz parte da interface de Room – qualquer alteração poderá afectar outras classes devido ao acoplamento

- Reduzir acoplamento ao mínimo

Poderá ser substituido por:

```
public void setExit (String direccao, Room vizinho)
   {
      exits.put (direccao, vizinho);
   }
```

Em vez de escrever lab.setExits(outside, office, null, null);
Escreve lab.setExit ("north", outside);
lab.setExit ("east", office);

Já é possível utilizar qualquer direcção

Implementar as alterações necessárias para que Room possa ter mais saídas

```
import java.util.HashMap;

class Room - MELHORADA
{
    private String description;
    private HashMap<String, Room> exits;// armazena saídas desta sala.

  public Room (String description)
  {
    this.description = description;
    exits = new HashMap<String, Room>();
  }

public void setExit(String direccao, Room vizinho)
  {
      exits.put (direccao, vizinho);
  }

public String getDescription()
  {
    return description;
  }

  public Room getExit(String direccao)
  {
    return exits.get(direccao);
 }

}
```



```
Class Game - MELHORADA
 private void createRooms()
   {
      Room outside, theatre, pub, lab, office, cellar;

      // cria as salas
      outside = new Room("outside the main entrance of the university");
      theatre = new Room("in a lecture theatre");
      pub = new Room("in the campus pub");
      lab = new Room("in a computing lab");
      office = new Room("in the computing admin office");

      // inicializa saídas da sala
//       outside.setExits(null, theatre, lab, pub);
         outside.setExit("east", theatre);
         outside.setExit("south", lab);
         outside.setExit("west", pub);
//       theatre.setExits(null, null, null, outside);
         theatre.setExit("west", outside);
//       pub.setExits(null, outside, null, null);
         pub.setExit("east", outside);
//       lab.setExits(outside, office, null, null);
         lab.setExit ("north", outside);
                        lab.setExit ("east", office);
//       office.setExits(null, null, null, lab);
         office.setExit("west", lab);
      currentRoom = outside;  // começa jogo em outside
   }
```

Desenho orientado por responsabilidade
Responsability-driven design

- Uso apropriado de encapsulamento
  - Reduz acoplamento
  - Pode reduzir significativamente o trabalho necessário para alterar aplicações

Mas

não é o único factor que influencia o grau de acoplamento:

- Desenho orientado por responsabilidade

Encapsulation is not the only factor that influences the degree of coupling

Another aspect: Responsability-driven design

- Pretendemos adicionar uma nova sala – uma cave por baixo do escritório
  - Informação das salas está apenas na classe Room – única responsável por fornecer esta informação

Adicionar uma nova sala – uma cave por baixo do escritório

```
private void createRooms()
  {
     Room outside, theatre, pub, lab, office, cellar;

     // cria as salas
     outside = new Room("outside the main entrance of the university");
     theatre = new Room("in a lecture theatre");
     pub = new Room("in the campus pub");
     lab = new Room("in a computing lab");
     office = new Room("in the computing admin office");
     cellar = new Room("in the cellar");

     // inicializa saídas das salas
        outside.setExit("east", theatre);
        outside.setExit("south", lab);
        outside.setExit("west", pub);
        theatre.setExit("west", outside);
        pub.setExit("east", outside);
        lab.setExit ("north", outside);
        lab.setExit ("east", office);
        office.setExit("west", lab);
        cellar.setExit("up", office);
        office.setExit("down", cellar);
     currentRoom = outside;  // start game outside
  }
```

Método melhorado para encontrar os nomes das saídas de uma sala
a improved version of getExitString()

```
private String getExitString()
   {
// devolve uma string que descreve as saídas da sala, por exemplo,
Exits: north west
// return a description of the room's exits, for examplee,
Exits: north west
          String returnString = "Exits:";
      Set<String> chaves = exits.keySet();
      for(String exit : chaves) {
         returnString += " " + exit;
      }
      return returnString;
   }
```

As chaves no HashMap são os nomes das saídas

the keys in the HashMap are the names of the exits

Coesão/cohesion

- Coesão de métodos / *methods cohesion*
  - Responsável por uma única tarefa
  - *responsible for one and only one well defined task*
- Coesão de classes/ *classes cohesion*
  - Representa apenas uma entidade bem definida
  - *Represents only one well defined entity*
- Principais benefícios da coesão no desenho/ *Main benefits*:
  - Reutilização / *reuse*
  - Legibilidade / *readability*