BUSINESS LOGIC DESIGN

Chapter 5

Software Engineering
Computer Science School
DSIC – UPV

Goals

- Understand the software design as a set of objects that interact with each other and manage their own state and operations.
- Learn how to derive a design model from a class diagram.
- Learn how to derive methods from sequence diagrams.

References

- Weitzenfeld, A., Ingeniería del Software OO con UML. Java e Internet. Thomson, 2005
- Stevens, P., Pooley, R. Utilización de UML en Ingeniería del Software con Objetos y Componentes. Addison-Wesley Iberoamericana 2002.
- . . .

Contents

Introduction

Conceptual Modeling *(Analysis)*

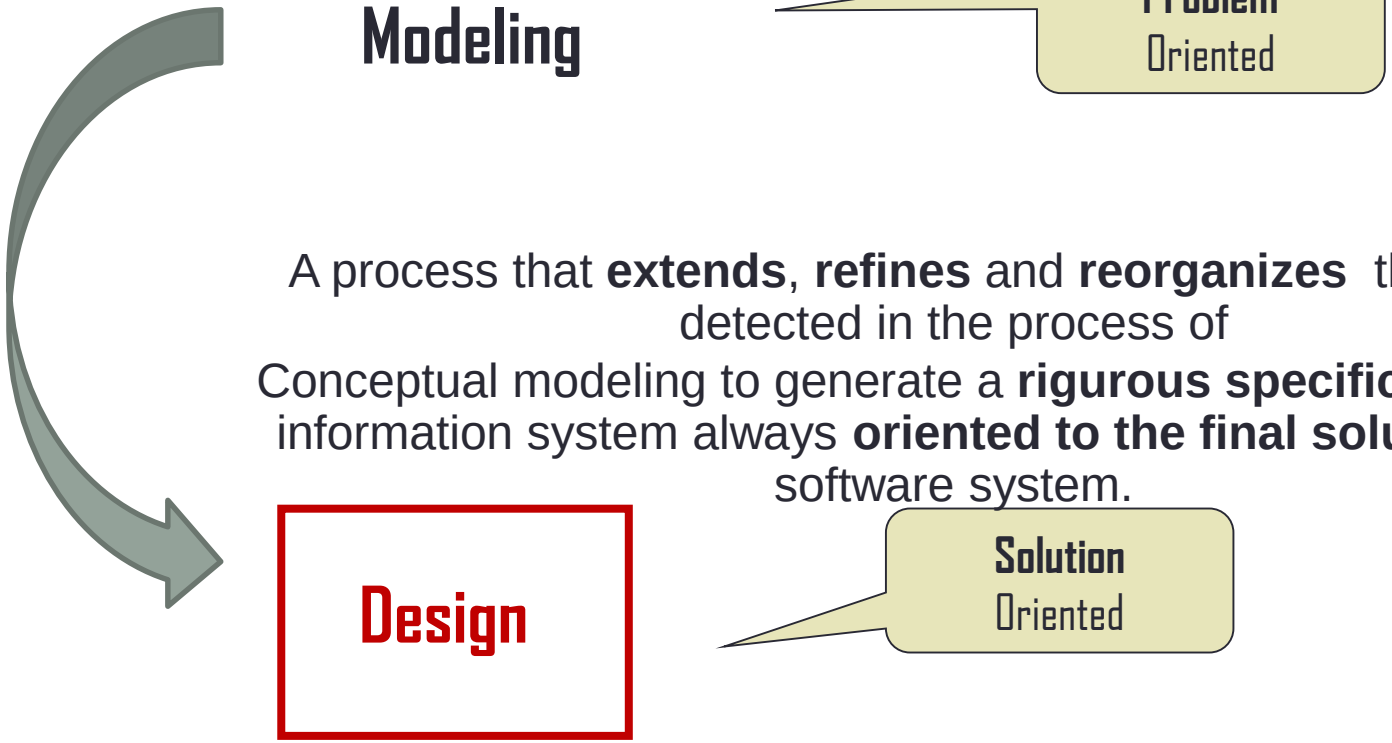
It is the process of constructing a **model** / of a detailed specification of
A problem of the real world we are confronted with.
It does not **contain** ***design and implementation*** elements

Modeling = Design?

NO

Introduction

Modeling *vs.* Design



The design adds the development environment and the implementation language as elements to consider.

OBJECTS DESIGN

Objects Design

- Input: Conceptual Modeling – **Class diagram**

** Refine Analysis class diagram

- Output: Design – **C# Design**

Design of Classes

Design of Associations

Design of Aggregations

Design of Specializations

Objects Design

** Refine analysis class diagram

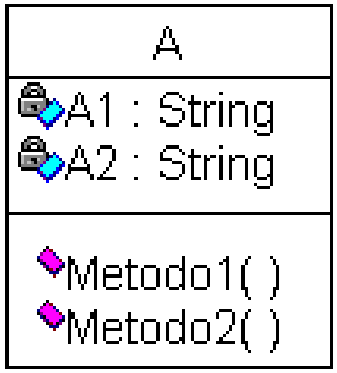
Design decisions

- ✓ Create new classes
- ✓ Remove/Join classes
- ✓ Create new relationships
- ✓ Modify existing relationships
  - ✓ Restrict navegability
- ✓ . . .

Design Class Diagram

Design Patterns. Classes (1/2)

Conceptual Modeling



Design

```
class A
{
 private String A1;
 private String A2;

 public int Metodo1(){…}
 public String Metodo2() {…}

 public void setA1(string a) {…}
 public void setA2(string a) {…}
 public String getA1() {…}
 public String geA2() {…}

}
```

Classes (2/2)

design

```
public int Metodo1()
{
   ...
   return ...
}

public String Metodo2()
{
   ...
   return ...
}

 public void setA1(string a){ A1=a;}

 public void setA2(string a){ A2=a;}


 public String getA1(){return A1;}

 public String getA2(){return A2;}
```

Design Patterns. Associations (1/10)

1-to-1 Relationship

Conceptual Modeling

design

```
class A
{
   private B Rb;
   public void setRb(B vB){…}
   public B    getRb(){…}
}
```

```
class B
{
   private A Ra;
   public void setRa(A vA){…}
   public A    getRa(){…}
}
```

Associations (2/10)

1-to-1 Relationship

Design

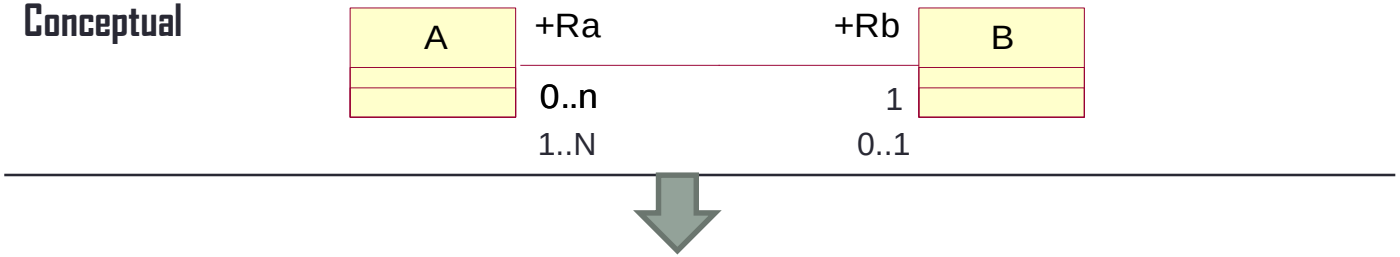
```
public void setRb(B vB)
{
   Rb=vB;
}

public void setRa(A vA)
{
   Ra=vA;
}
```

```
public B getRb()
{
   return Rb;
}

public A getRa()
{
   return Ra;
}
```

Associations (3/10)

1-to-Many Relationship

Modelado Conceptual



Design

```
class A
{
...   // same pattern as before with cardinality 1
}

class B
{
 private ICollection<A> collectionRa; // (Interface Collections)
 public A getA(string identifier_A){…}
 public void add_A(A objetoA){…}
 public void remove_A(A objetoA){…}
 // or remove_A(string identifier_A){…}
}
```

```
// Methods manipulating
// the A Collection
```

Associations (4/10)

```
class B
{
 private ICollection<A> collectionRa; // if it is List: List,
LinkedList

public void add_Ra(A objectA)
{
    collectionRa.Add(objectA)
}

public void remove_Ra(A objectA)
{
    collectionRa.Remove(ObjectA)
}
 . . .

]
```

SortedList, Stack, Queue, Dictionary, SortedDictionary, HashSet…

Associations (5/10)

The design of the **Associations**
Will be as defined depending on the max cardinality being 1 or N.

Associations (6/10)

Many-to-Many Relationship

Conceptual Modeling

Design

```
class A
{
 private ICollection<B> ColectionOfB //Set, Queue, Map, …  A []
 public B get_B(string identifier_B){…}
 public void add_B(B objectB){…}
 public void remove_B(C objectB){…}
}

class B
{
 private ICollection<A> CollectionOfA //Set, Queue, Map, …  A []
 public A get_A(string identificador_A){…}
 public void add_A(A objectA){…}
 public void remove_A(A objectA){…}

}
```

Associations (7/10)

1-1 Association (Association Class)

Conceptual Modeling

Design

```
class A
{
   private C The_C;
   public void setC(C vC){…}
   public C getC(){…}
}
```

```
class B
{
 private C The_C;
   public void setC(C vC){…}
   public C getC(){…}
}
```

Associations (8/10)

1-1 Association (Association Class)

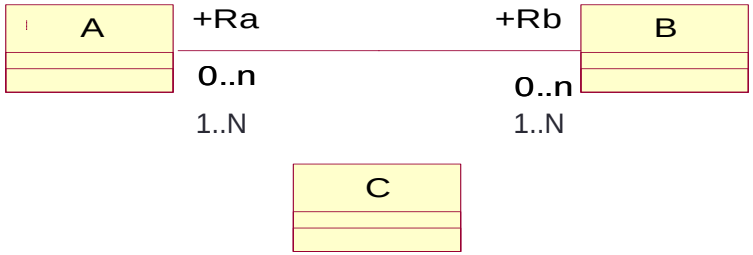
Design

```
class C
{
   private A Ref_A;
   private B Ref_B;

   public void setA(A vA){…}
   public A getA(){…}

   public void setB(B vB){…}
   public B getB(){…}
}
```

Associations (9/10)

Many-to-Many Association (Association Class)

Modelado Conceptual



Design

```
class A
{
 private ICollection<C> CollectionOfC; //Set, Queue, Map, …  A []
 public C get_C(string identifier_C){…}
 public void add_C(C objectC){…}
 public void remove_C(C objectC){…}
}

class B
{
 private ICollection<C>  CollectionOfC; //Set, Queue, Map, …  A []
 public C get_C(string identifier_C){…}
 public void add_C(C objectC){…}
 public void remove_C(C objectC){…}
}
```

Associations (10/10)

Many-to-Many Association (Association Class)

Design

```
class C
{
   private A Ref_A;
   private B Ref_B;

   public void setA(A vA){…}
   public A getA(){…}

   public void setB(B vB){…}
   public B getB(){…}
}
```

Design Patterns. Aggregation/Composition (1/2)

1-1 Aggregation

Conceptual Modeling

Design

```
class A
{
        private   B Rb;
        public void setRb(B vB){…}
        public B getRb(){…}
}

class B
{
        private   A Ra;
        public void setRa(A vA){…}
        public A getRa(){…}
}
```

Aggregation/Composition (2/2)

Aggregation 1-Many

Conceotual Modeling

Aggregation Many-Many

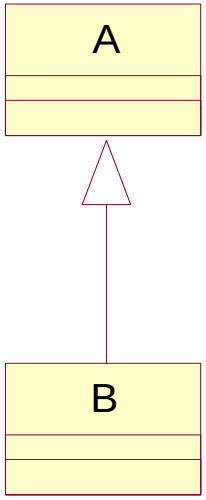
Conceptual Modeling

```
....  // same patterns already discussed
```

Specialization/Generalization

Simple Specialization

Conceptual Modeling



Design

```
class A
{
 ...
}

class B : A
{
 ...
}
```

DESIGN OF MESSAGES

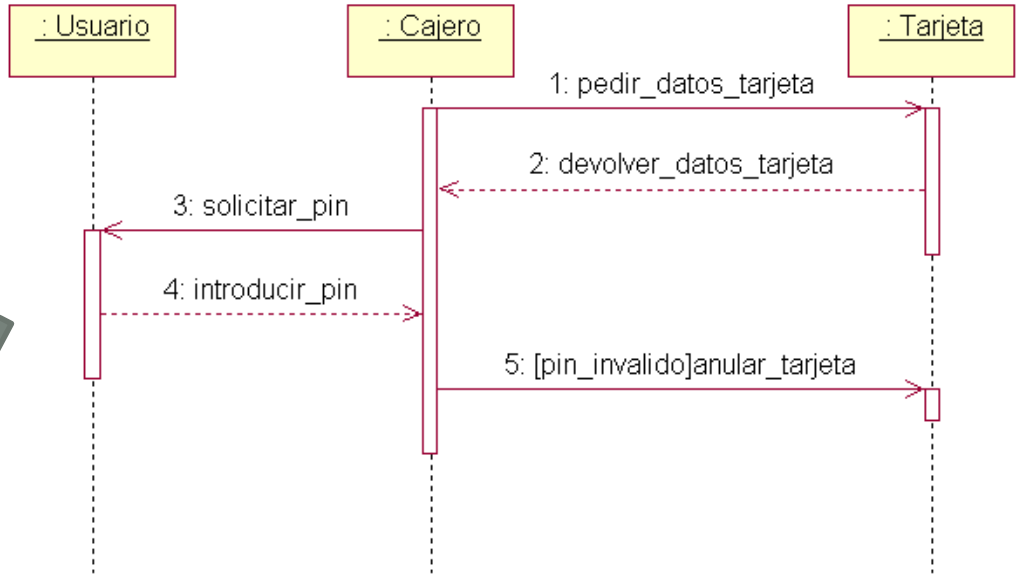
Design of Messages

- *Input:* **Sequence Diagram**
  - Identify communication within the system
- *Output:* Obtain operations (**methods**) of classes

- If an object receives a message → its class will offer a method to serve the message
- If an object sends a message → the invocation of the method of the destination object will be in the body of a method of the sender object.

Design of Messages

Scenario: Pin Update




```
class ATM
{
 private . . .
. . .
 public void change_pin()
  { . . .
    ObjTarjeta.get_card_data(. .  .);
    . . .
    if (pin_invalid){
    ObjTarjeta.cancel_card (. . .);}

  }}
```

```
class Card
{
   private . . .;

   . . .
   public void get_card_data(. . .){…}
   public void cancel_card(. . .){…}

}
```