

Programación Orientada a Objetos

Interacción entre objetos

CEIS

2020-1

Agenda

0.0

Visión

Desarrollo POOB

Requisitos

Diseño

Construcción + Pruebas

Refactorización

Un error en producción

Principios POOB

MDD

BDD

SOLID

Fuentes POOB

Agenda

0.0

Visión

Desarrollo POOB

Requisitos

Diseño

Construcción + Pruebas

Refactorización

Un error en producción

Principios POOB

MDD

BDD

SOLID

Fuentes POOB

SRS

STUDENT REGISTRATION SYSTEM (SRS) CASE STUDY: SRS REQUIREMENTS SPECIFICATION

We have been asked to develop an automated Student Registration System (SRS). This system will enable students to register online for courses each semester, as well as track a student's progress toward completion of his or her degree.

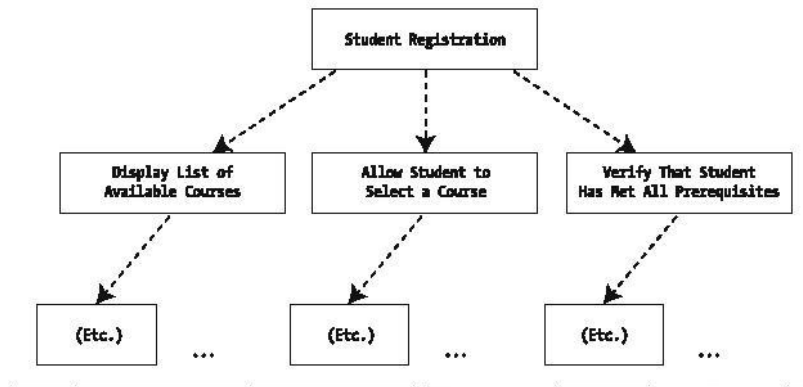
Visión

Descomposición funcional

¿QUÉ DEBE HACER?

Visión

Descomposición funcional



¿QUÉ DEBE HACER?

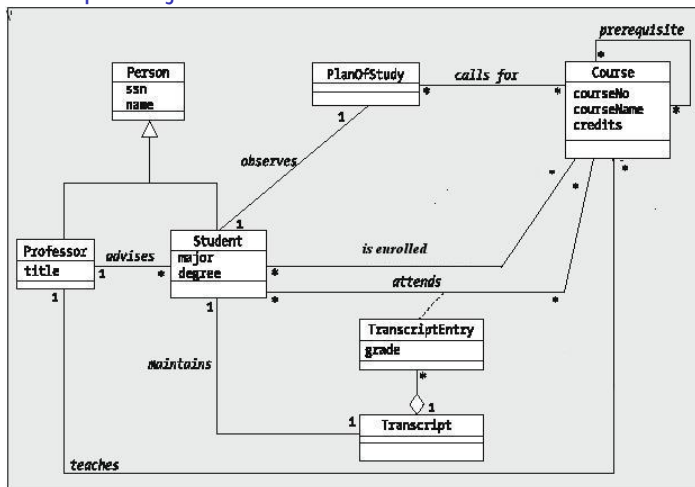
Visión

Orientado por objetos

¿QUÉ DEBE CONOCER?

Visión

Orientado por objetos



¿QUÉ DEBE CONOCER?

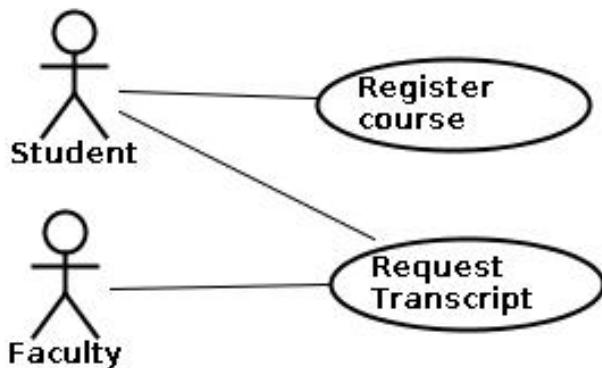
Visión

Orientado por objetos

¿Qué debe hacer?

Visión

Orientado por objetos



¿Qué debe hacer?

Agenda

0.0

Visión

Desarrollo POOB

Requisitos

Diseño

Construcción + Pruebas

Refactorización

Un error en producción

Principios POOB

MDD

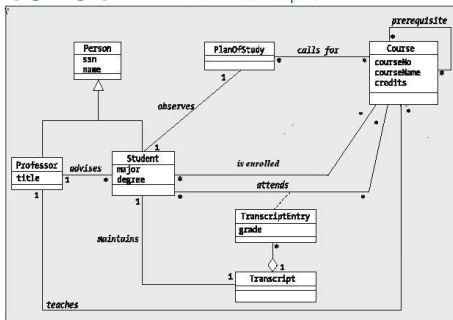
BDD

SOLID

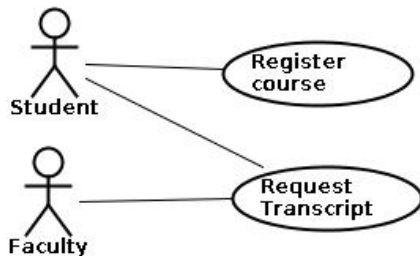
Fuentes POOB

Requisitos

CONOCER. Modelo de conceptos.

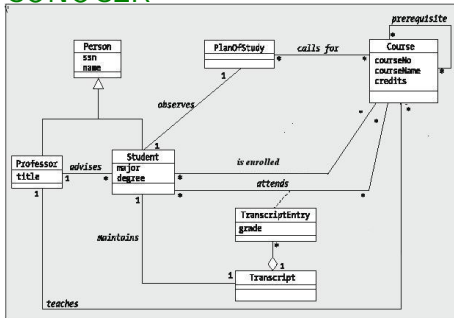


hacer. Modelo de casos de uso.

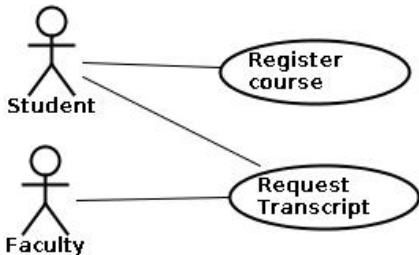


Requisitos

CONOCER



hacer

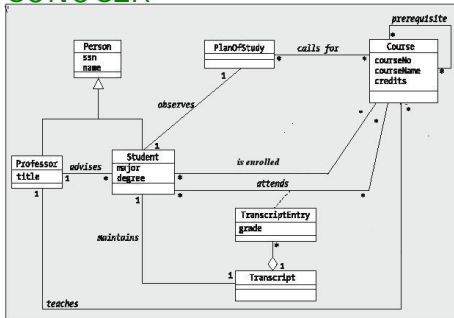


¿Ciclos?

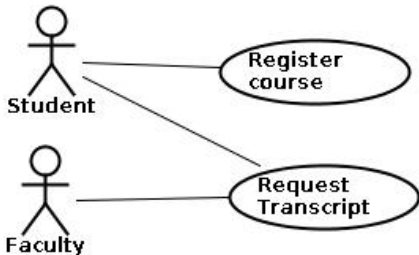
- ▶ ¿Qué modelo sirve para dividir?
- ▶ ¿Cuáles son los posibles ciclos?
- ▶ ¿En qué orden se abordarían?

Requisitos

CONOCER



hacer



¿Ciclos?

- ▶ ¿Qué modelo sirve para dividir?
- ▶ ¿Cuáles son los posibles ciclos?
- ▶ ¿En qué orden se abordarían?

Primer ciclo: un estudiante se registra en un curso

Agenda

0.0

Visión

Desarrollo POOB

Requisitos

Diseño

Construcción + Pruebas

Refactorización

Un error en producción

Principios POOB

MDD

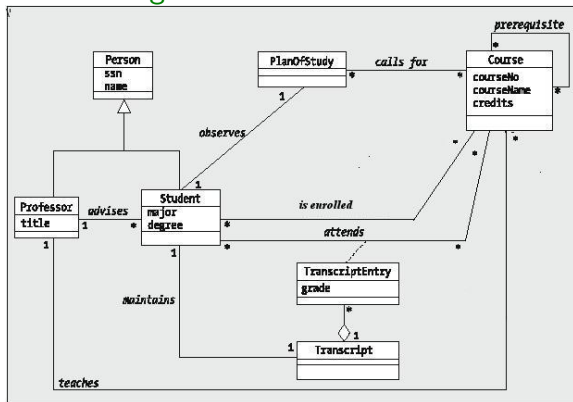
BDD

SOLID

Fuentes POOB

Colaboración

Un estudiante se registra en un curso

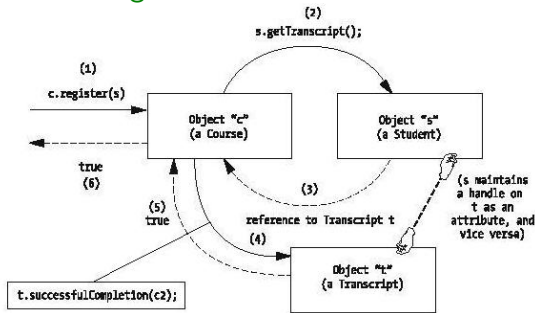


Preguntas de diseño

1. ¿Quién puede ser el responsable? **SOLID** Single Responsibility Principle
2. ¿Quiénes le deben colaborar?
3. ¿Cómo lo podrían hacer?

Colaboración

Un estudiante se registra en un curso

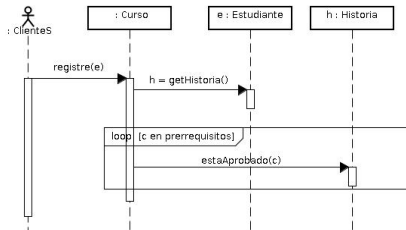


Preguntas de diseño

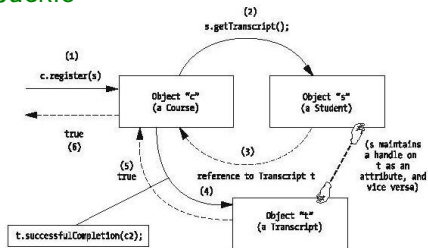
1. ¿Quién puede ser el responsable? **SOLID** Single Responsibility Principle
2. ¿Quiénes le deben colaborar?
3. ¿Cómo lo podrían hacer?

Colaboración

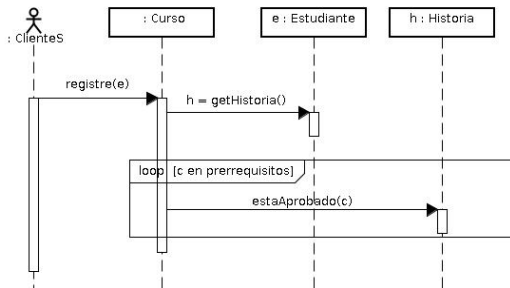
UML



Jackie

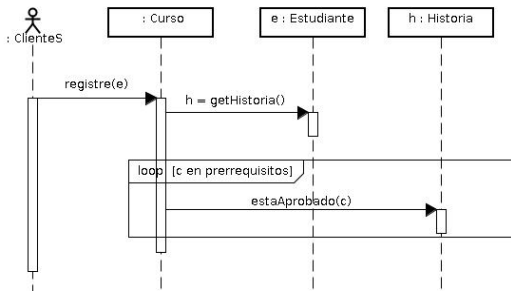


Visibilidad



¿Cómo un objeto puede ver a otro?

Visibilidad



¿Cómo un objeto puede ver a otro?

1. Lo tiene como atributo

De Atributo

2. Le llega como parámetro en un método

De Parametro

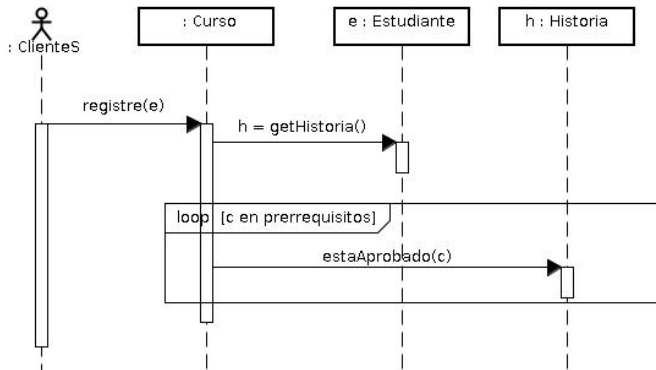
3. El lo crea o lo pide a otro objeto que conoce

Local

4. Es un objeto que todos pueden ver (es global)

Colaboración

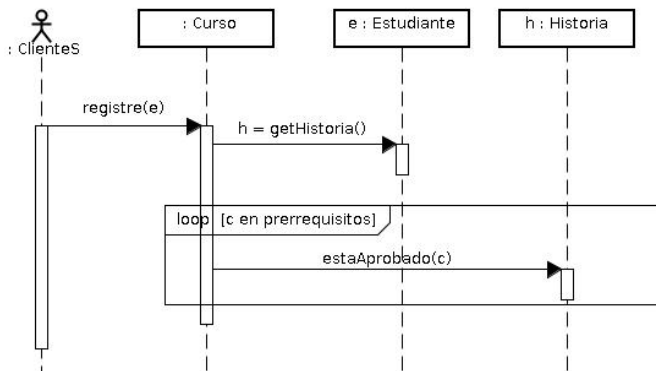
Registrarse



¿?

1. ¿Qué conoce el Course? ¿Cómo?
2. ¿Qué conoce el Student? ¿Cómo?

A Clases



1. ¿Qué clases tenemos en el diseño?

EN ZONA 1

2. ¿Qué atributos tenemos?

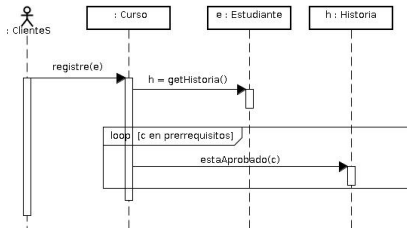
EN ZONA 2 - EN RELACIONES

3. ¿Qué métodos tenemos?

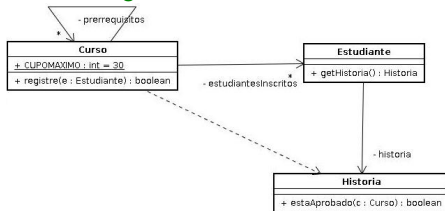
EN ZONA 3

Modelos de diseño

UML. Diagrama de secuencia



UML. Diagrama de clases

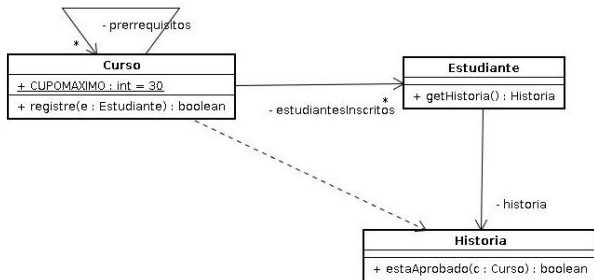


Especificando en java

Para cada clase

- Naturaleza, información e invariante

Comentario inicial



Especificando en java

Para cada clase

- Naturaleza, información e invariante

Comentario inicial

¿Especificación?

Curso

1. Naturaleza:

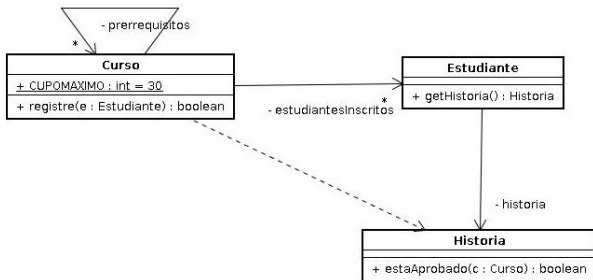
¿A quién representa?

2. Información:

¿Qué información tiene?

3. Invariante :

¿Qué condición debe cumplir siempre?



Especificando en java

Documentación

Class Curso

java.lang.Object

└ **Curso**

public class **Curso**

Representa un curso

(prerrequisitos, cupoMaximo, estudiantesInscritos)

CUPOMAXIMO > 0 y **cupoMaximo** >=#**estudiantesInscritos**

Field Summary

static int	<u>CUPOMAXIMO</u>
------------	-----------------------------------

Código

```
import java.util.ArrayList;
/**
Representa un curso <br>
(prerrequisitos, cupoMaximo, estudiantesInscritos) <br>
<b>CUPOMAXIMO</b> > 0 y <b>cupoMaximo</b> >=#<b>estudiantesInscritos</b>
*/
public class Curso {

    public static final int CUPOMAXIMO=30;
    private ArrayList<Curso> prerrequisitos;
    private ArrayList<Estudiante> estudiantesInscritos;
```

Especificando en java

Para cada método

- ▶ Objetivo

Comentario inicial

- ▶ Parámetros

@param

- ▶ Retorno (Si retorna)

@return

Especificando en java

Para cada método

► Objetivo

Comentario inicial

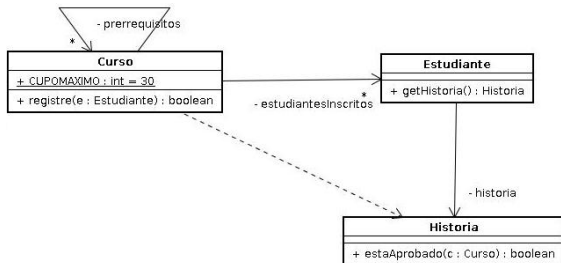
► Parámetros

@param

► Retorno (Si retorna)

@return

Curso



¿Especificación de registre?

1. Precondición:

¿Condiciones para poder registrar a un estudiante?

2. Poscondición:

¿Condición después de hacer el registro ?

INVARIANTE DE CLASE

Especificando en java

Código

```
/**
 * Si es posible, registra un nuevo estudiante al curso.
 * @param s es el estudiante a adicionar
 * @return Si el estudiante se logró registrar o no.
 * Las condiciones de registro son: <ol>
 * <li> Existe cupo en el curso
 * <li> El estudiante no se ha registrado al curso
 * <li> El estudiante cumple con los requisitos
 * </ol>
 */
public boolean registre(Estudiante s) {
    return false;
}
```

Documentación

registre

public boolean **registre**(Estudiante s)

Si es posible, registra un nuevo estudiante al curso.

Parameters:

s - es el estudiante a adicionar

Returns:

Si el estudiante se logró registrar o no. Las condiciones de registro son:

1. Existe cupo en el curso
2. El estudiante no se ha registrado al curso
3. El estudiante cumple con los requisitos

Agenda

0.0

Visión

Desarrollo POOB

Requisitos

Diseño

Construcción + Pruebas

Refactorización

Un error en producción

Principios POOB

MDD

BDD

SOLID

Fuentes POOB

Probando en java

Documentación

registre

```
public boolean registre(Estudiante s)
```

Si es posible, registra un nuevo estudiante al curso.

Parameters:

s - es el estudiante a adicionar

Returns:

Si el estudiante se logró registrar o no. Las condiciones de registro son:

1. Existe cupo en el curso
2. El estudiante no se ha registrado al curso
3. El estudiante cumple con los requisitos

Prueba

The screenshot shows an IDE with a UML diagram on the left and Java code on the right. The UML diagram illustrates the relationships between `CursoTest`, `Historia`, `Estudiante`, and `Curso`. `CursoTest` has a dependency on `Historia` and `Estudiante`. `Estudiante` has a dependency on `Curso`. The Java code on the right is a test class `CursoTest` with two test methods: `deberiaPermitirInscribirUnEstudianteSiEsPosible` and `noDeberiaPermitirInscribirSiNoHayCupo`.

```
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CursoTest {

    @Test
    public void deberiaPermitirInscribirUnEstudianteSiEsPosible() {
        Estudiante e = Estudiante.get("12546");
        Curso c = Curso.get("MBDA");
        assertTrue("Debe inscribirlo", c.registre(e));
    }

    @Test
    public void noDeberiaPermitirInscribirSiNoHayCupo() {
        ...
    }
}
```

Probando en java

Documentación

registre

```
public boolean registre(Estudiante s)
```

Si es posible, registra un nuevo estudiante al curso.

Parameters:

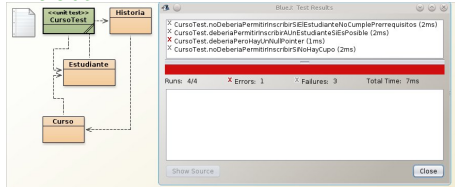
s - es el estudiante a adicionar

Returns:

Si el estudiante se logró registrar o no. Las condiciones de registro son:

1. Existe cupo en el curso
2. El estudiante no se ha registrado al curso
3. El estudiante cumple con los requisitos

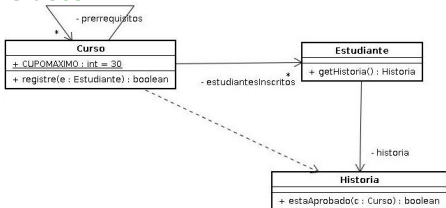
Prueba



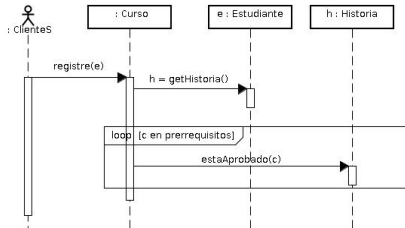
Errores vs Fallos

Modelado UML - JAVA

Clases



Colaboración



Documentación

registre

```
public boolean registre(Estudiante s)
```

Si es posible, registra un nuevo estudiante al curso.

Parameters:

`s` - es el estudiante a adicionar

Returns:

Si el estudiante se logró registrar o no. Las condiciones de registro son:

1. Existe cupo en el curso
2. El estudiante no se ha registrado al curso
3. El estudiante cumple con los requisitos

¿Código?

ArrayList

Method Summary	
boolean	<code>add(E o)</code> Appends the specified element to the end of this list.
boolean	<code>contains(Object o)</code> Returns true if this list contains the specified element.
E	<code>get(int index)</code> Returns the element at the specified position in this list.
int	<code>size()</code> Returns the number of elements in this list.

Probando en java

Documentación

registre

```
public boolean registre(Estudiante s)
```

Si es posible, registra un nuevo estudiante al curso.

Parameters:

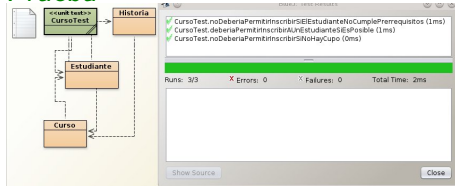
s - es el estudiante a adicionar

Returns:

Si el estudiante se logró registrar o no. Las condiciones de registro son:

1. Existe cupo en el curso
2. El estudiante no se ha registrado al curso
3. El estudiante cumple con los requisitos

Prueba



Agenda

0.0

Visión

Desarrollo POOB

Requisitos

Diseño

Construcción + Pruebas

Refactorización

Un error en producción

Principios POOB

MDD

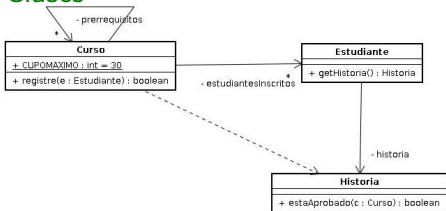
BDD

SOLID

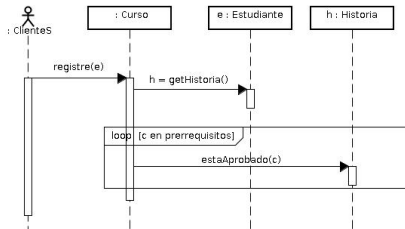
Fuentes POOB

Modelado UML - JAVA

Clases



Colaboración



: (No es un buen diseño

Alto acoplamiento

- ▶ ¿Cuál sería un mejor diseño?
- ▶ ¿Cuáles son los cambios en código?
- ▶ ¿Qué pasa con las pruebas?

Probando en java

Documentación

registre

```
public boolean registre(Estudiante s)
```

Si es posible, registra un nuevo estudiante al curso.

Parameters:

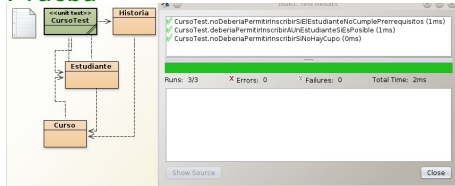
s - es el estudiante a adicionar

Returns:

Si el estudiante se logró registrar o no. Las condiciones de registro son:

1. Existe cupo en el curso
2. El estudiante no se ha registrado al curso
3. El estudiante cumple con los requisitos

Prueba



Agenda

0.0

Visión

Desarrollo POOB

Requisitos

Diseño

Construcción + Pruebas

Refactorización

Un error en producción

Principios POOB

MDD

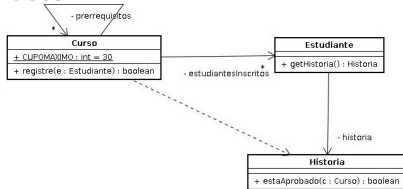
BDD

SOLID

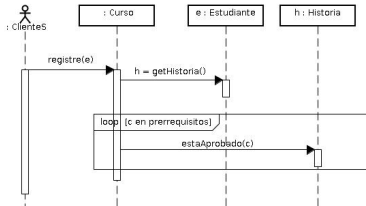
Fuentes POOB

Modelado UML - JAVA

Clases



Colaboración



Documentación

registre

public boolean **registre**(Estudiante s)

Si es posible, registra un nuevo estudiante al curso.

Parameters:

s - es el estudiante a adicionar

Returns:

Si el estudiante se logró registrar o no. Las condiciones de registro son:

1. Existe cupo en el curso
2. El estudiante no se ha registrado al curso
3. El estudiante cumple con los requisitos

Inscribió un estudiante que ya la había aprobado. ¿QUÉ HACER?

- ▶ Ajustar la especificación
- ▶ Escribir nuevas pruebas
- ▶ Revisar el diseño
- ▶ Modificar el código
- ▶ Ejecutar todas las pruebas.

Agenda

0.0

Visión

Desarrollo POOB

Requisitos

Diseño

Construcción + Pruebas

Refactorización

Un error en producción

Principios POOB

MDD

BDD

SOLID

Fuentes POOB

MDD-Desarrollo Dirigido Por Modelos



Agenda

0.0

Visión

Desarrollo POOB

Requisitos

Diseño

Construcción + Pruebas

Refactorización

Un error en producción

Principios POOB

MDD

BDD

SOLID

Fuentes POOB

BDD-Desarrollo Dirigido Por Comportamiento



Agenda

0.0

Visión

Desarrollo POOB

Requisitos

Diseño

Construcción + Pruebas

Refactorización

Un error en producción

Principios POOB

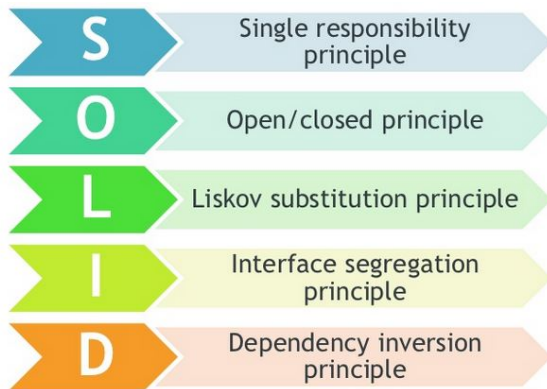
MDD

BDD

SOLID

Fuentes POOB

SOLID- Principios básicos



S : Primer tercio

O : Segundo tercio

LID: CVDS

Fuentes POOB

```
/**
 * Draw a given shape onto the canvas.
 * @param referenceObject an object to define identity for this shape
 * @param color           the color of the shape
 * @param shape           the shape object to be drawn on the canvas
 */
// Note: this is a slightly backwards way of maintaining the shape
// objects. It is carefully designed to keep the visible shape interfaces
// in this project clean and simple for educational purposes.
public void draw(Object referenceObject, String color, Shape shape){
    objects.remove(referenceObject); // just in case it was already there
    objects.add(referenceObject);    // add at the end
    shapes.put(referenceObject, new ShapeDescription(shape, color));
    redraw();
}
```

“Code is more often read than written”

- ▶ ¿Es verdad? ¿Por qué?
- ▶ ¿Qué es el código? ¿Qué son las fuentes?

Fuentes POOB

```
/**
 * Draw a given shape onto the canvas.
 * @param referenceObject an object to define identity for this shape
 * @param color           the color of the shape
 * @param shape           the shape object to be drawn on the canvas
 */
// Note: this is a slightly backwards way of maintaining the shape
// objects. It is carefully designed to keep the visible shape interfaces
// in this project clean and simple for educational purposes.
public void draw(Object referenceObject, String color, Shape shape){
    objects.remove(referenceObject); // just in case it was already there
    objects.add(referenceObject);    // add at the end
    shapes.put(referenceObject, new ShapeDescription(shape, color));
    redraw();
}
```

“Code is more often read than written”

Elementos

- ▶ Documentación: ¿qué?
- ▶ Código: ¿cómo?
- ▶ Comentarios: ¿por qué?

En lenguajes imperativos