

Programación orientada a objetos

Patrones

CEIS

2025-1

Patrones

Introducción

Otra historia

Definición

Inventario

Evaluativos

Básicos

Reutilización

Extensibilidad

Cierre

Patrones

Introducción

Otra historia

Definición

Inventario

Evaluativos

Básicos

Reutilización

Extensibilidad

Cierre

Otra historia

Desarrollador ECI



Consultor en arquitectura
y platatormas

¿ Cómo hago para eliminar el
tiempo de conexión a la base de
datos en cada invocación http, y
tener así tiempos de respuesta más
bajos?

FACIL!!, Haga un **singleton**,
que contenga un **resource
pooling** que **adapte**
a clases tipo **abstract
factory** de conexiones, e
implemente un **observer**
para notificar el estado de
la conexión.



Desarrollador

Solitario

Problema

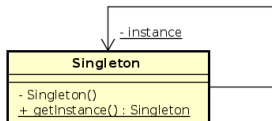
¿Cómo tener una única instancia de una clase con un sólo punto de acceso?

Solución

¿?

Solitario

Diseño



Código

```
public class Singleton {  
  
    private static Singleton instance = null;  
  
    private Singleton() {  
    }  
  
    public static Singleton getInstance() {  
        if (instance == null) {  
            instance = new Singleton();  
        }  
        return instance;  
    }  
}
```

Solitario

Problema

¿Cómo tener una única instancia de una clase con un sólo punto de acceso?

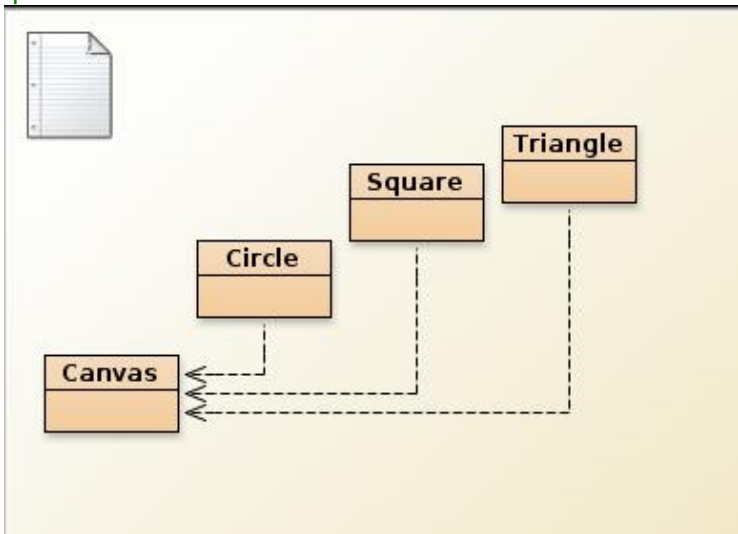
Solución

Definir un método de clase que devuelva el solitario

Referencia

GoF

Shapes



¿Quién es un solitario?

Canvas

```
public class Canvas
{
    private static Canvas canvasSingleton;

    public static Canvas getCanvas()
    {
        if(canvasSingleton == null) {
            canvasSingleton = new Canvas("BlueJ Shapes Demo", 300, 300,
                                         Color.white);
        }
        canvasSingleton.setVisible(true);
        return canvasSingleton;
    }
}
```

Canvas

```
/**
 * Create a Canvas.
 * @param title title to appear in Canvas Frame
 * @param width the desired width for the canvas
 * @param height the desired height for the canvas
 * @param bgColour the desired background colour of the canvas
 */
private Canvas(String title, int width, int height, Color bgColour){
    frame = new JFrame();
    canvas = new CanvasPane();
    frame.setContentPane(canvas);
    frame.setTitle(title);
    canvas.setPreferredSize(new Dimension(width, height));
    backgroundColour = bgColour;
    frame.pack();
    objects = new ArrayList <Object>();
    shapes = new HashMap <Object,ShapeDescription>();
}
```

Canvas

```
public void draw()
{
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color, new Ellipse2D.Double(xPosition, yPosition,
                                                         diameter, diameter));
        canvas.wait(10);
    }
}
```

¿Problema?

Patrones

Definición

Un patrón es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlos en nuevas situaciones y discusiones sobre sus puntos fuertes y débiles.

Catalogos

[8 – 11] **GRASP** [2 – 40] **GOF**

- ▶ Patrones evaluativos
- ▶ Patrones básicos
- ▶ Patrones de reutilización
- ▶ Patrones de extensibilidad
- ▶ Patrones estructurales

Catalogos

GRASP: General Responsibility Assignment Software Patterns

- ▶ Patrones evaluativos
 - 1 Alta cohesión
 - 2 Bajo acoplamiento
- ▶ Patrones básicos
 - 3 Experto
 - 4 Creador
 - 5 No hables con extraños
 - 6 Controlador
- ▶ Patrones de reutilización
 - 7 Polimorfismo
- ▶ Patrones de extensibilidad
 - 8 Variaciones protegidas
- ▶ Patrones estructurales

Catalogos

GOF: Gang of Four

- ▶ Patrones evaluativos
 - 1 Alta cohesión
 - 2 Bajo acoplamiento
- ▶ Patrones básicos
 - 3 Experto
 - 4 Creador
 - 5 No hables con extraños
 - 6 Controlador
- ▶ Patrones de reutilización
 - 7 Polimorfismo
 - 10 Plantilla
- ▶ Patrones de extensibilidad
 - 8 Variaciones protegidas
- ▶ Patrones estructurales
 - 9 Solitario

Patrones

Introducción

Otra historia

Definición

Inventario

Evaluativos

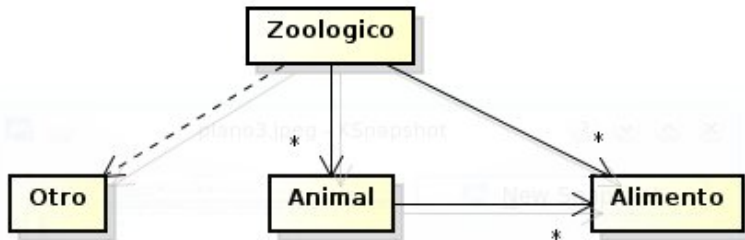
Básicos

Reutilización

Extensibilidad

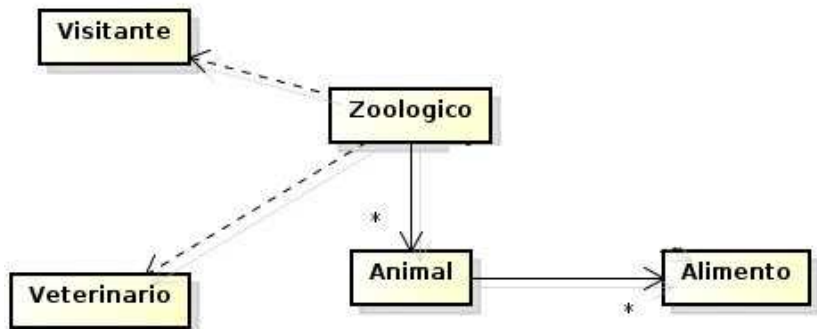
Cierre

Zoologico



¿Problemas?

Zoologico



GRASP

Problema

Decidir ¿cómo mantener controlable la complejidad?

Solución

Asignar las responsabilidades de manera que se mantenga **alta cohesión**.

Nombre

Alta cohesión

Problema

Decidir ¿ cómo dar soporte a poca dependencia y a una mayor reutilización?

Solución

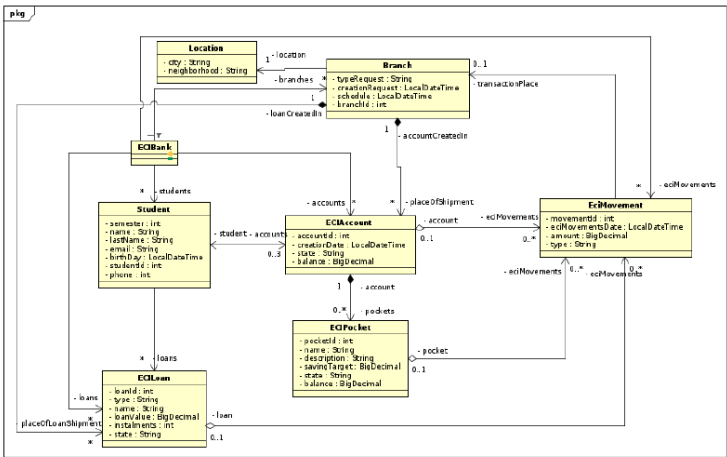
Asignar las responsabilidades de manera que se mantenga **bajo acoplamiento**.

Nombre

Bajo acoplamiento

GRASP

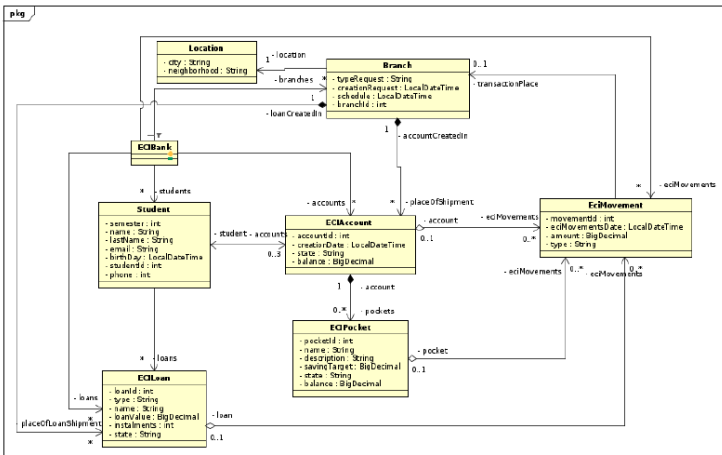
Parcial



► ¿Cuál es el componente con menos cohesión? ¿Qué justifica su existencia?

GRASP

Parcial



- ¿Cuál es el componente con menos cohesión? ¿Qué justifica su existencia?
- ¿Cuál es componente más acoplado? ¿Es posible bajar el nivel de acoplamiento?

Patrones

Introducción

Otra historia

Definición

Inventario

Evaluativos

Básicos

Reutilización

Extensibilidad

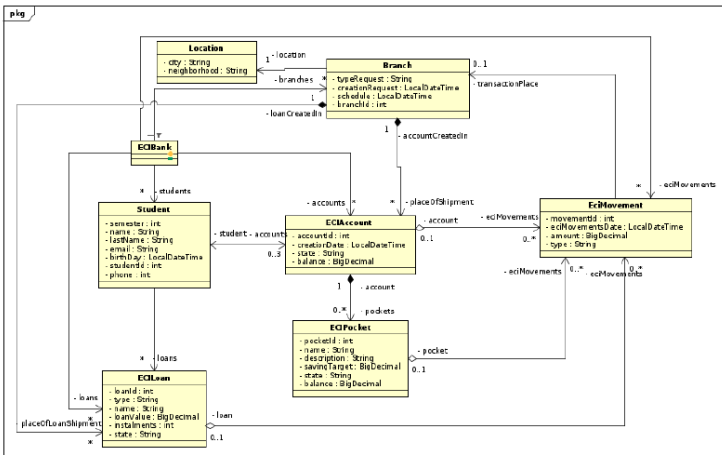
Cierre

Problema

Decidir ¿ quién debe asumir una responsabilidad?

GRASP

Parcial



¿Quién debe ser responsable?

Consultar el número de cuentas creadas cada mes durante el último año
(Diagrama de secuencia)

GRASP

Problema

Decidir ¿ quién debe asumir una responsabilidad?

Solución

Asignar la responsabilidad al **experto** en información: la clase que posee la información necesaria para cumplir con la responsabilidad.

Nombre

Experto

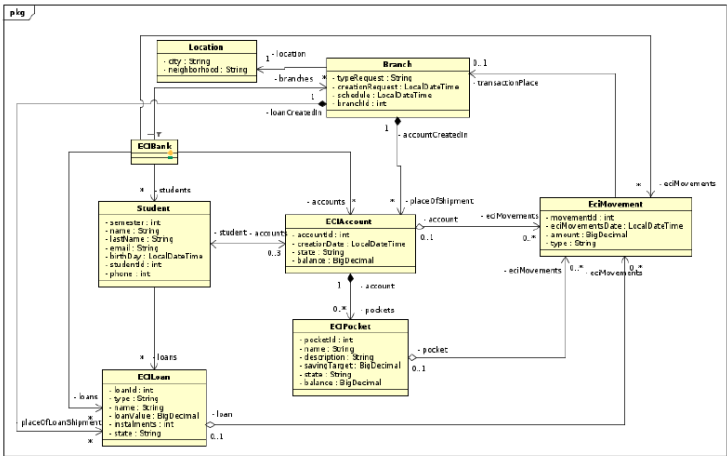
GRASP

Problema

Decidir ¿ quién debe crear una instancia?

GRASP

Parcial



¿Quién debe crear?

► Adicionar una cuenta (studentId) [El estado inicial es PROCESSING]

¿Quién crea el objeto? (Diagrama de secuencia)

GRASP

Problema

Decidir ¿ quién debe crear una instancia?

Solución

Asignar a la clase B la responsabilidad de crear una instancia de A, si se cumplen una de las siguientes condiciones:

- ▶ B contiene a A
- ▶ B agrega a A
- ▶ B tiene los datos de inicialización de A

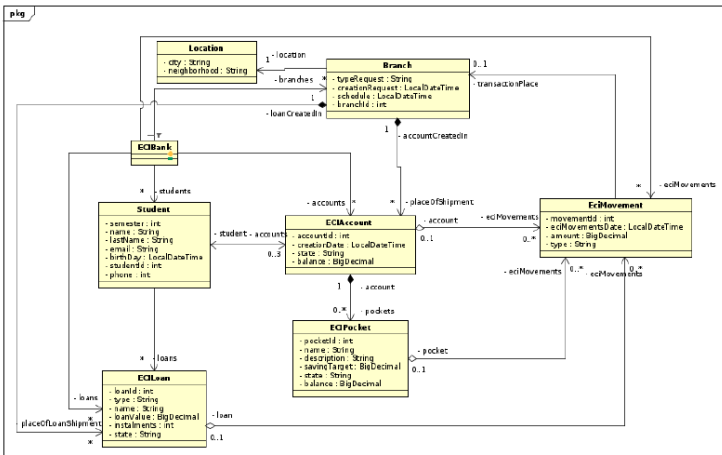
Referencia

GRASP

Problema

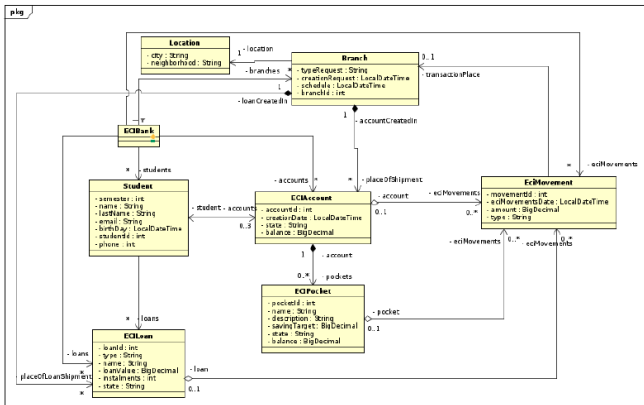
¿Quién, para evitar conocer a objetos indirectos?

Parcial



¿Quiénes son dos extraños?

► ¿Qué pasa si se hablan directamente? ¿Qué otra alternativa hay?



¿Quiénes no debería hablar?

- Consultar el número total de movimientos de un estudiante

Diagrama de secuencia

GRASP

Problema

¿Quién, para evitar conocer a objetos indirectos?

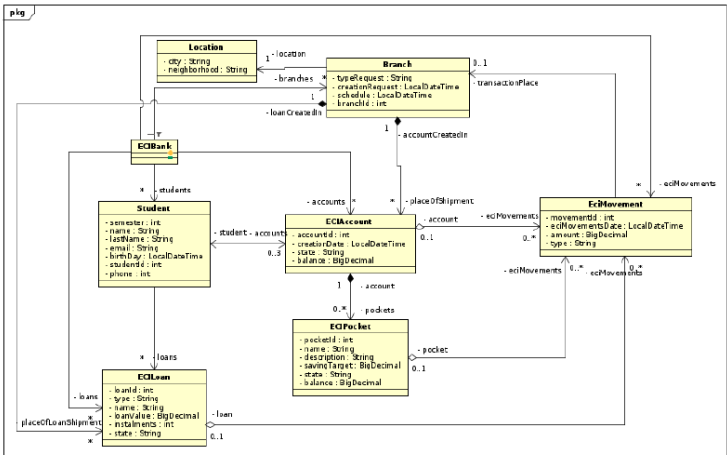
Solución

Asignar la responsabilidad a un objeto conocido por el cliente que conozca al objeto indirecto.

Nombre

No hables con extraños

Parcial



¿Diagrama de casos de uso?

GRASP

Problema

Decidir ¿ quién debe administrar un evento externos del sistema?

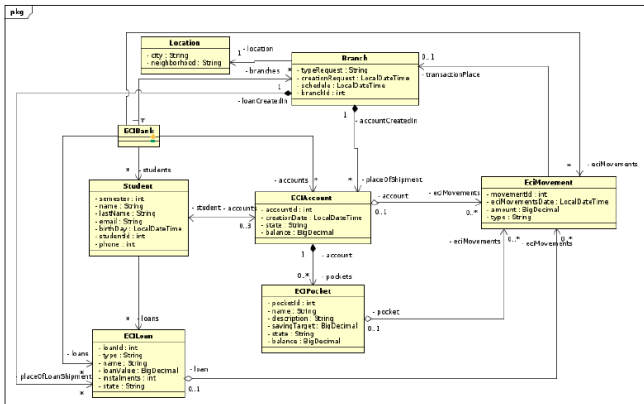
Solución

Asignar la responsabilidad de administrar los eventos externos del sistema a una clase que represente las siguientes opciones:

- ▶ El negocio o la organización global (fachada)
- ▶ El sistema global (fachada)
- ▶ Un ser animado del dominio que realice el trabajo (controlador de papeles)
- ▶ Una clase artificial que represente el caso de uso (controlador de caso de uso)

Nombre

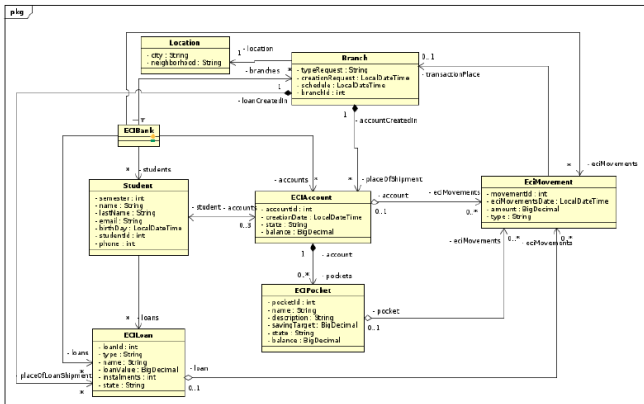
Controlador



Controladores. Alternativas

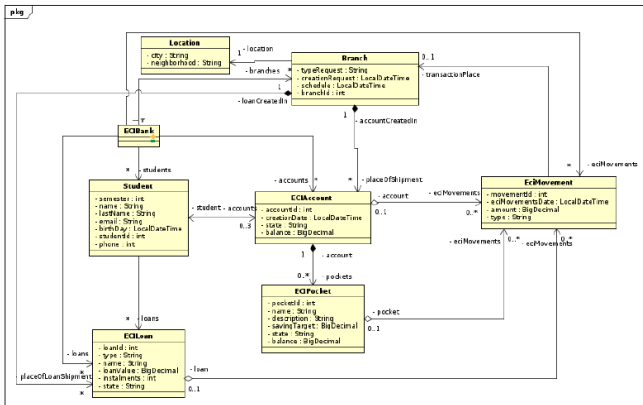
► Controlador fachada: ¿Quién?

¿Qué otro patrón tendría la fachada?



Controladores. Alternativas

- ▶ Controlador fachada: ¿Quién?
- ▶ Controladores de papeles: ¿Cuáles?



Controladores. Alternativas

- ▶ Controlador fachada: ¿Quién?
- ▶ Controladores de papeles: ¿Cuáles?
- ▶ Controladores de casos de uso: ¿Cuáles?

Patrones

Introducción

Otra historia

Definición

Inventario

Evaluativos

Básicos

Reutilización

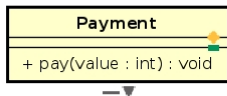
Extensibilidad

Cierre

Problema

¿Cómo manejar comportamientos diferentes basados en tipo?

Reutilización



```
public void pay(int value){  
    if (type==CASH){  
        //Code for a cash payment  
    } else if (type==CREDIT){  
        //Code for a credit payment  
    }  
}
```

¿Una propuesta mejor?

Problema

¿Cómo manejar comportamientos diferentes basados en tipo?

Solución

Defina una subclase para cada tipo para el que el comportamiento sea diferente y asígnele la responsabilidad usando métodos polimorficos.

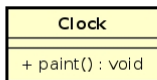
Nombre

Polimorfismo

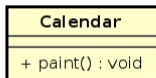
Problema

¿ Cómo implementar la parte invariable de un algoritmo una vez y dejar que las subclases implementen el comportamiento que puede variar ?

Reutilización



```
public void paint(){  
    backgroundClean();  
    //Code to paint the clock  
    //  
    //  
    frameRefresh();  
}
```



```
public void paint(){  
    backgroundClean();  
    //Code to paint the calendar  
    //  
    //  
    frameRefresh();  
}
```

Problema

¿ Cómo implementar la parte invariable de un algoritmo una vez y dejar que las subclasses implementen el comportamiento que puede variar ?

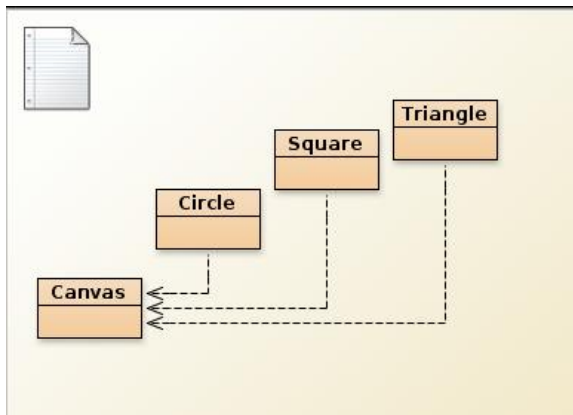
Solución

Crear una clase abstracta en la que se implementa el método plantilla que define el esqueleto de algoritmo. La parte variable corresponderá a métodos abstractos. En cada clase concreta implementar los métodos correspondientes a la parte variable del algoritmo.

Nombre

Plantilla

Reutilización



- ▶ ¿Dónde usamos polimorfismo?
- ▶ ¿Dónde usamos plantilla?

Reutilización

draw en Canvas

```
/**
 * 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();
}
```

draw en Circle

```
private void draw(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color,
            new Ellipse2D.Double(xPosition, yPosition,
                diameter, diameter));
        canvas.wait(10);
    }
}
```

draw en Rectangle

```
private void draw() {
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color,
            new java.awt.Rectangle(xPosition, yPosition,
                width, height));
        canvas.wait(10);
    }
}
```

¿draw en Shapes?

Reutilización

```
/*  
 * Return the shape of the figure  
 * @return the shape of the figure  
 */  
protected abstract Shape shape();  
/*  
 * Draw the figure with current specifications on screen.  
 */  
private void draw(){  
    if(isVisible) {  
        Canvas.getCanvas().draw(this, color, shape());  
        Canvas.getCanvas().wait(10);  
    }  
}
```

Patrones

Introducción

Otra historia

Definición

Inventario

Evaluativos

Básicos

Reutilización

Extensibilidad

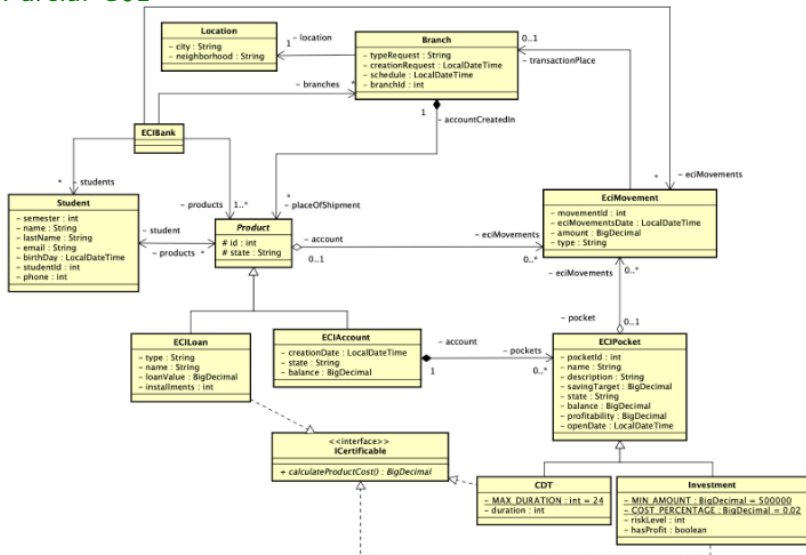
Cierre

Problema

¿ Cómo diseñar de manera que las variaciones de los componentes no tengan un impacto indeseable en otros componentes?

Extensibilidad

Parcial G01



¿Qué variaciones estamos permitiendo?

Problema

¿ Cómo diseñar de manera que las variaciones de los componentes no tengan un impacto indeseable en otros componentes?

Solución

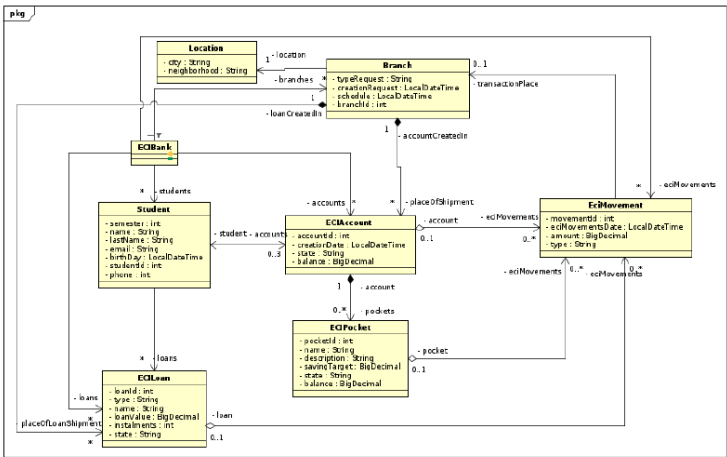
Identifique los puntos de variaciones; asigne responsabilidades para crear una interfaz estable alrededor de ellos.

Nombre

Variaciones protegidas

GRASP

Parcial

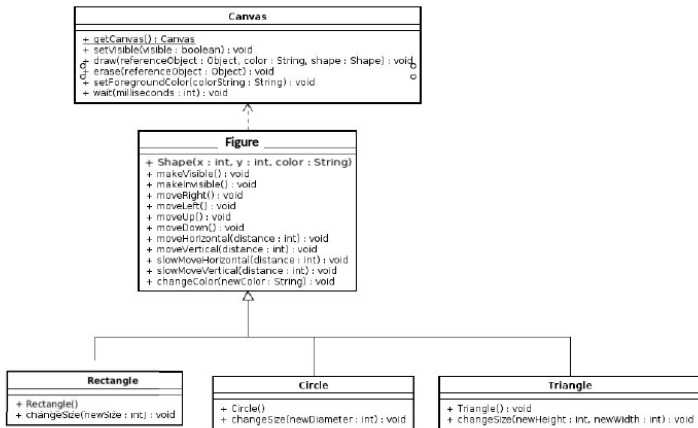


¿Qué hacer?

► Ofrecer diferente tipos de bolsillos.

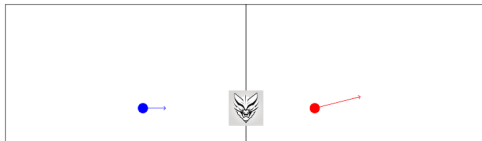
Por ejemplo: Generous (Premio de 5% al cumplir objetivo), Staggered (1% de aportes hacia cuenta), etc

Shapes



- ¿Dónde polimorfismo?
- ¿Dónde plantilla?
- ¿Hay variaciones protegidas?

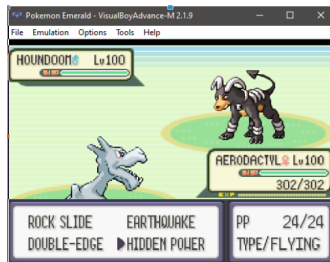
Proyecto inicial



- ▶ ¿Dónde polimorfismo?
- ▶ ¿Dónde plantilla?
- ▶ ¿Dónde variaciones protegidas?

Reutilización

Proyecto final



- ▶ ¿Qué puntos de variación queremos permitir?
- ▶ ¿Cómo se protegen efectivamente las variaciones?

Patrones

Introducción

Otra historia

Definición

Inventario

Evaluativos

Básicos

Reutilización

Extensibilidad

Cierre

Catalogos

[8 – 11] **GRASP** [2 – 40] **GOF**

- ▶ Patrones evaluativos
- ▶ Patrones básicos
- ▶ Patrones de reutilización
- ▶ Patrones de extensibilidad
- ▶ Patrones estructurales

Catalogos

[8 – 11] **GRASP** [2 – 40] **GOF**

- ▶ Patrones evaluativos
 - 1 Alta cohesión
 - 2 Bajo acoplamiento
- ▶ Patrones básicos
 - 3 Experto
 - 4 Creador
 - 5 No hables con extraños
 - 6 Controlador
- ▶ Patrones de reutilización
 - 7 Polimorfismo
- ▶ Patrones de extensibilidad
 - 8 Variaciones protegidas
- ▶ Patrones estructurales

Catalogos

[8 – 11] **GRASP** [2 – 40] **GOF**

- ▶ Patrones evaluativos
 - 1 Alta cohesión
 - 2 Bajo acoplamiento
- ▶ Patrones básicos
 - 3 Experto
 - 4 Creador
 - 5 No hables con extraños
 - 6 Controlador
- ▶ Patrones de reutilización
 - 7 Polimorfismo
 - 10 Plantilla
- ▶ Patrones de extensibilidad
 - 8 Variaciones protegidas
- ▶ Patrones estructurales
 - 9 Solitario