Programación Orientada a Objetos

Relaciones entre objetos

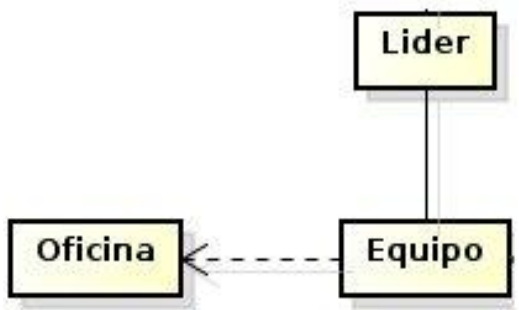
CEIS

2020-2

Agenda

Relaciones

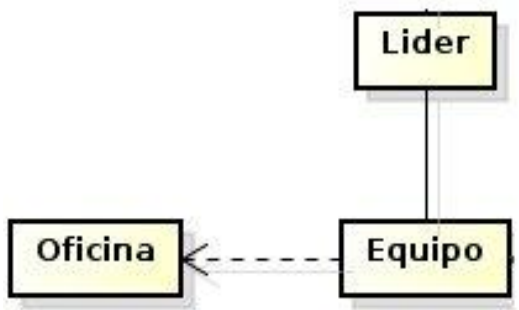
Unidad de proyectos



¿Qué leemos?

Relaciones

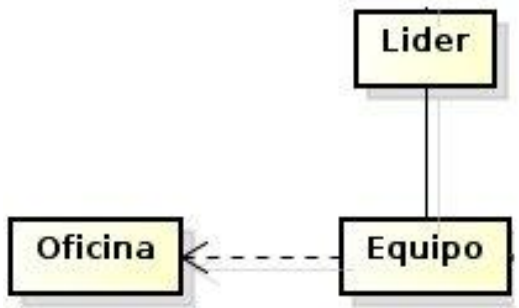
Unidad de proyectos



¿Tipos de relaciones?

Relaciones

Unidad de proyectos

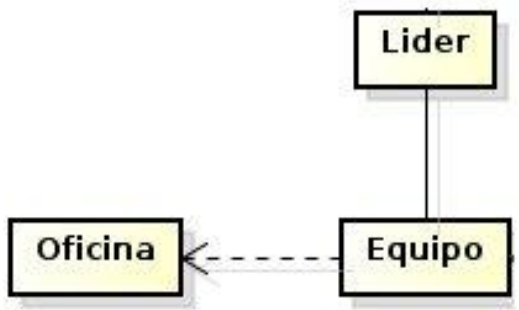


¿Tipos de relaciones?

Relaciones estructurales. Relaciones de comportamiento

Relaciones

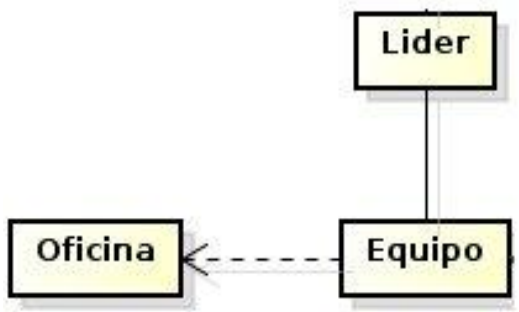
Unidad de proyectos



RELACIONES ESTRUCTURALES
Los equipos conocen su líder (líder) y el líder conoce sus equipos (equipos).

Relaciones

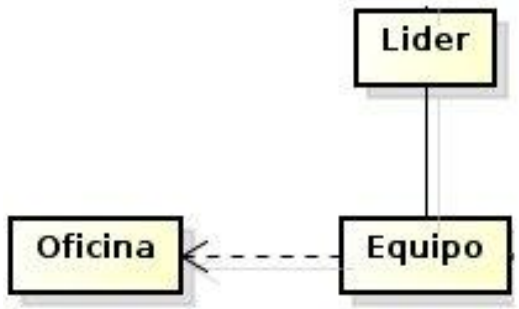
Unidad de proyectos



RELACIONES ESTRUCTURALES
Los equipos conocen su líder (líder) y el líder conoce sus equipos (equipos).
¿Atributos? ¿Visibilidad? ¿Roles?

Relaciones

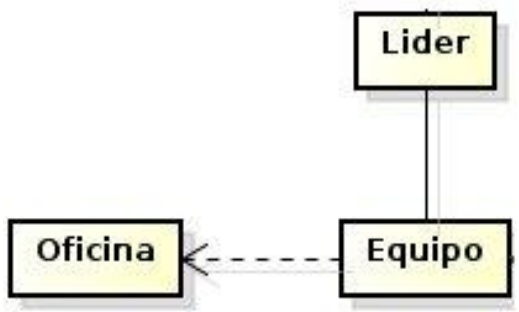
Unidad de proyectos



RELACIONES ESTRUCTURALES
Los equipos pueden tener un líder. Los lideres pueden trabajar con varios equipos.

Relaciones

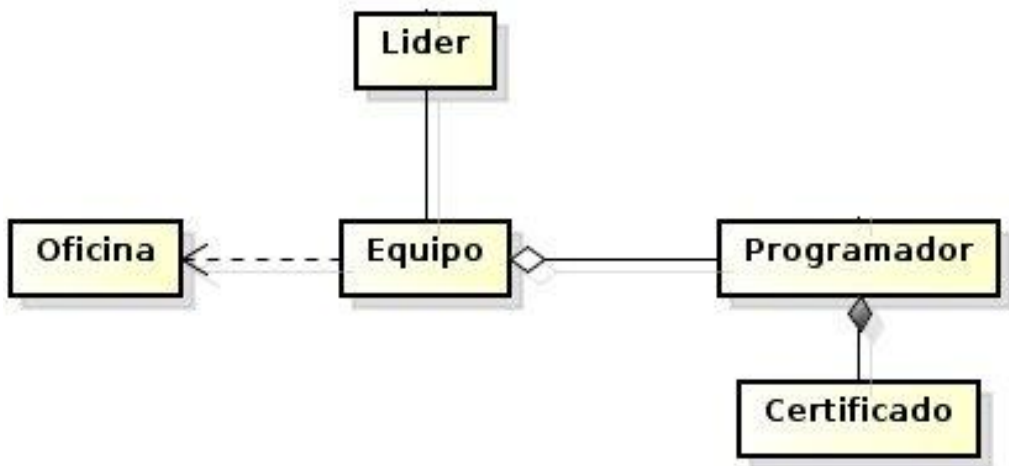
Unidad de proyectos



RELACIONES ESTRUCTURALES
Los equipos pueden tener un líder. Los lideres pueden trabajar con varios equipos.
¿Cardinalidades? 1:1 1:N M:N

Relaciones

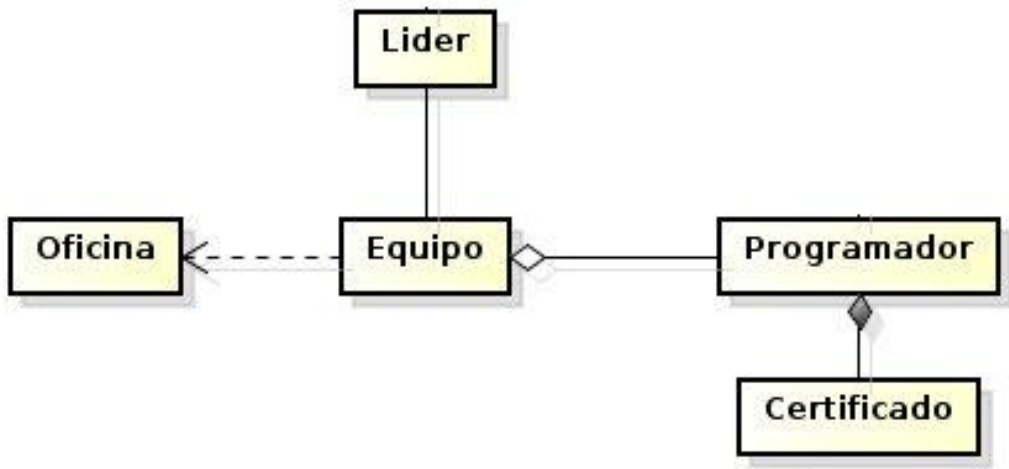
Unidad de proyectos



¿Qué leemos?

Relaciones

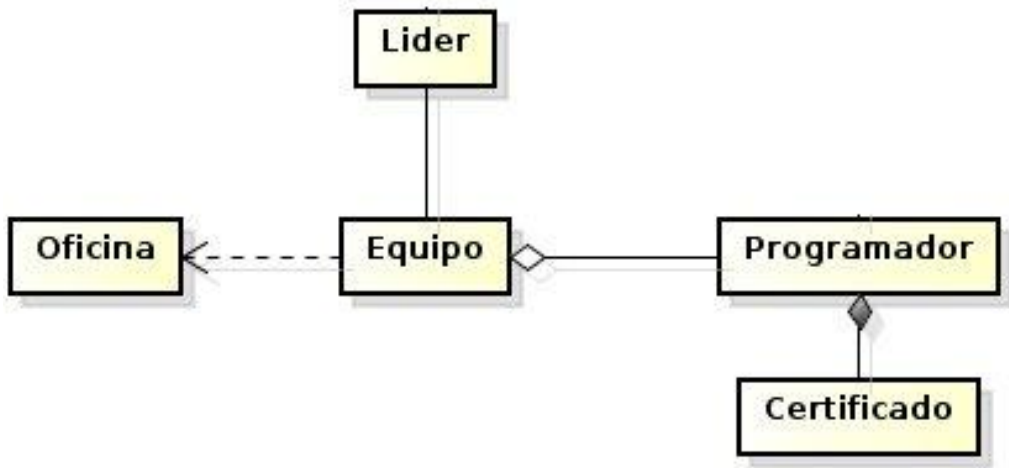
Unidad de proyectos



RELACIONES TODO-PARTE

Relaciones

Unidad de proyectos

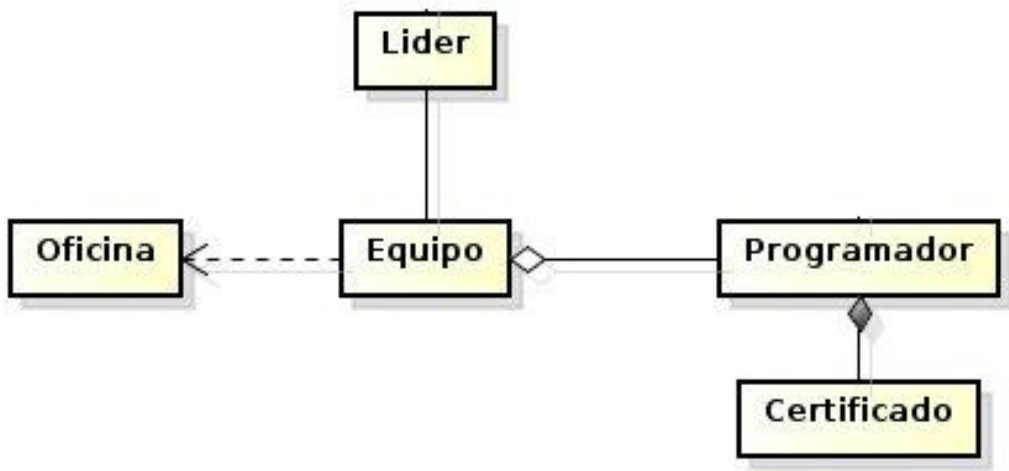


RELACIONES TODO-PARTE

¿Agregación? ¿Composición?

Relaciones

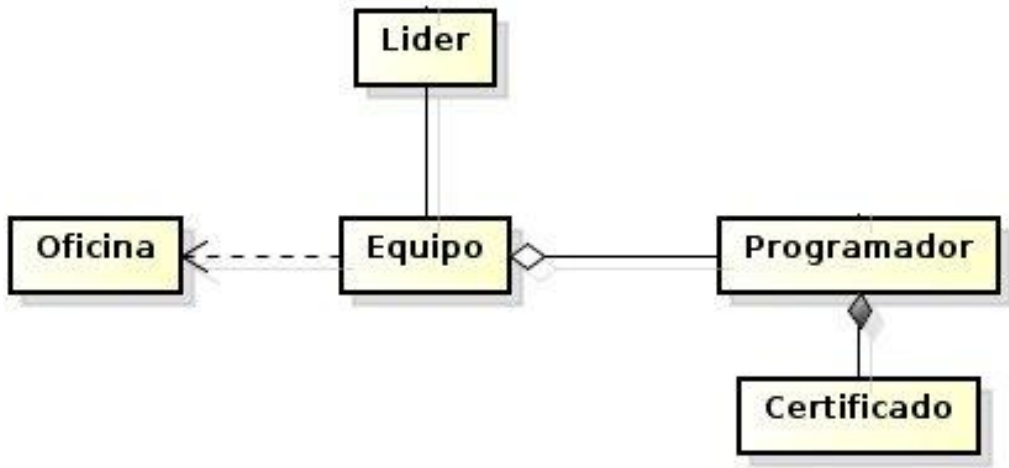
Unidad de proyectos



RELACIONES TODO-PARTE
Un equipo está compuesto de varios programadores, un programador pertenece a un único equipo. El equipo es quien conoce sus programadores.

Relaciones

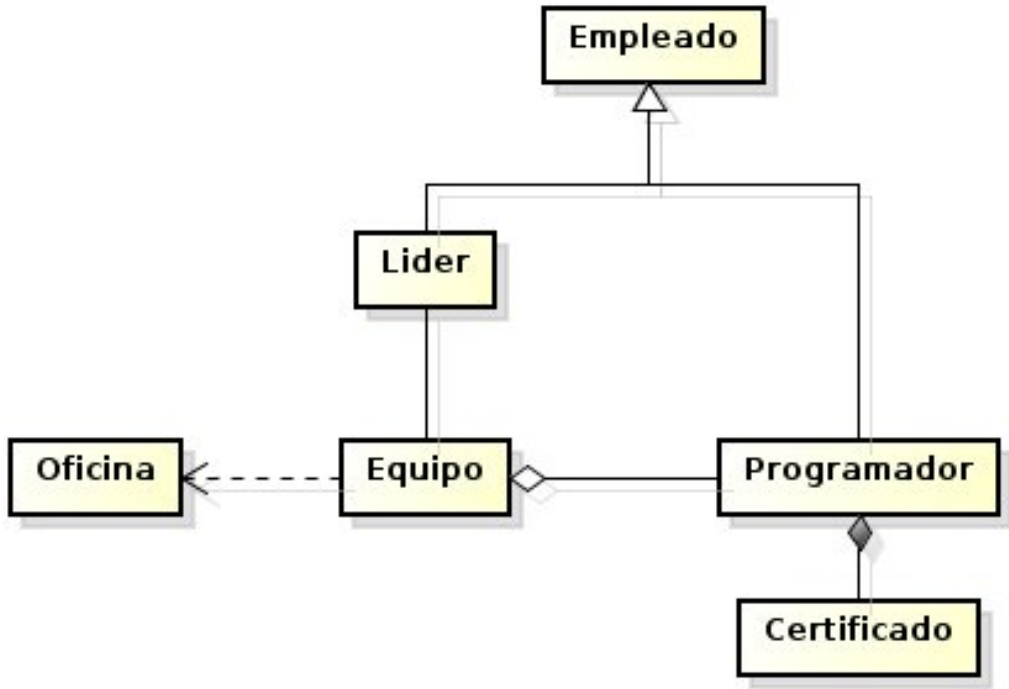
Unidad de proyectos



RELACIONES TODO-PARTE
Los certificados son de cada programador. El programador conoce sus certificados y el certificado su dueño.

Relaciones

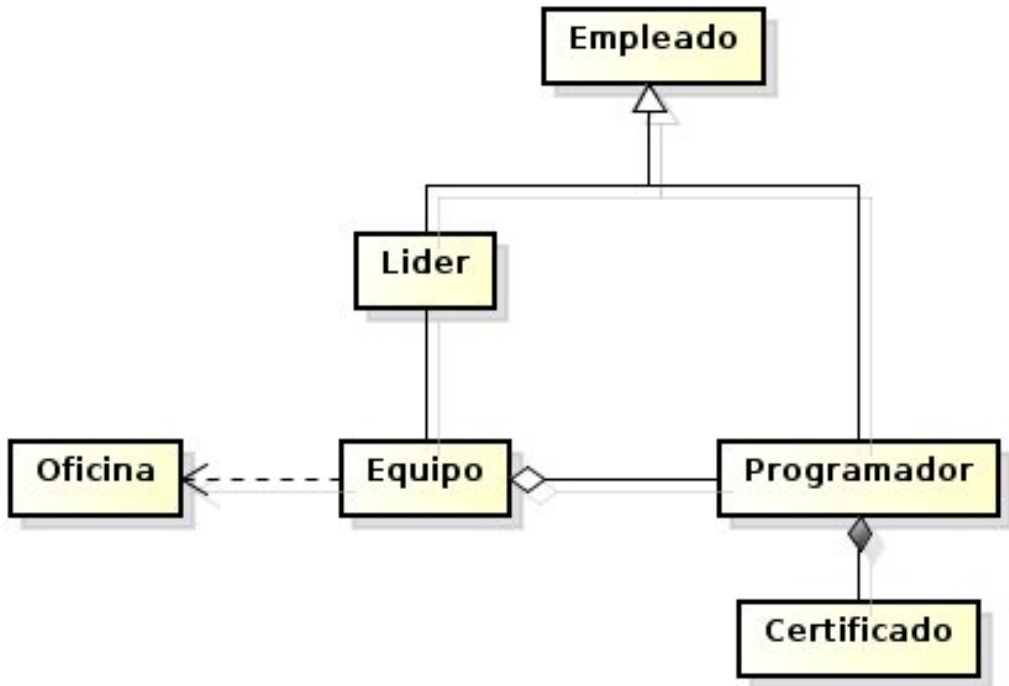
Unidad de proyectos



¿Qué leemos?

Relaciones

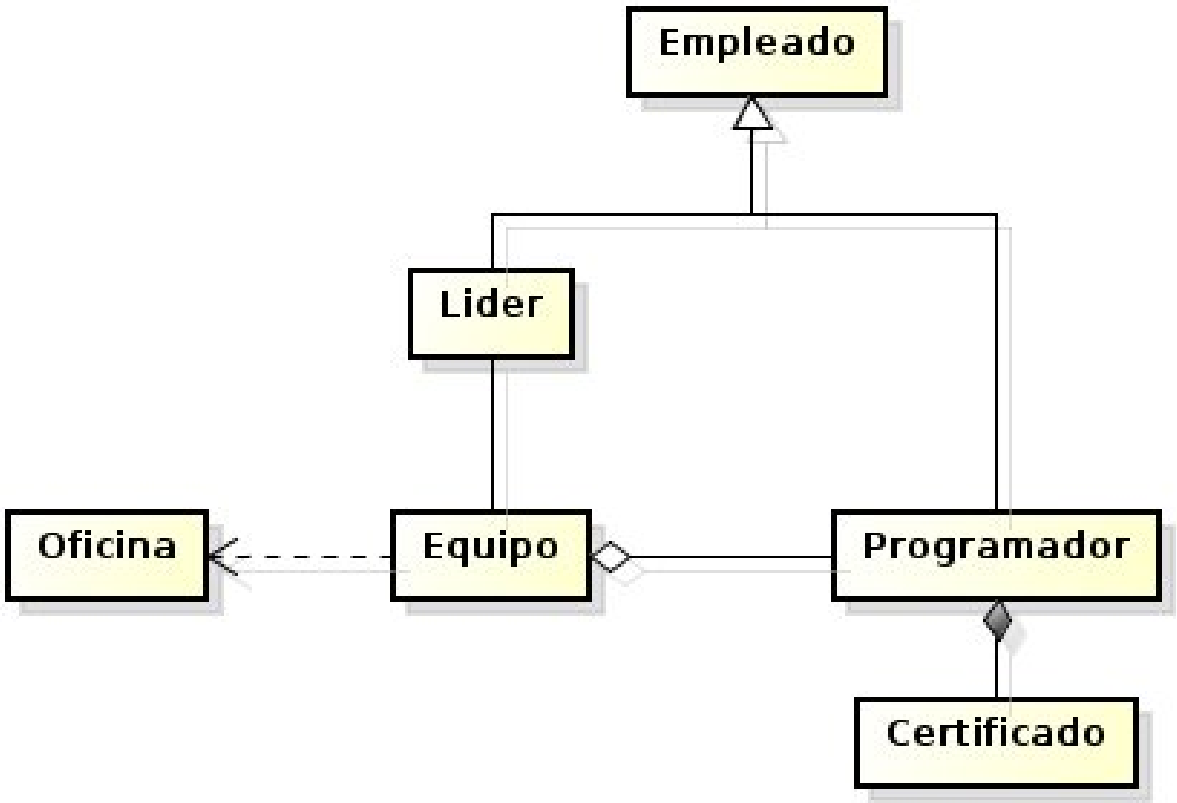
Unidad de proyectos



RELACIONES DE HERENCIA-Es Un

Agenda

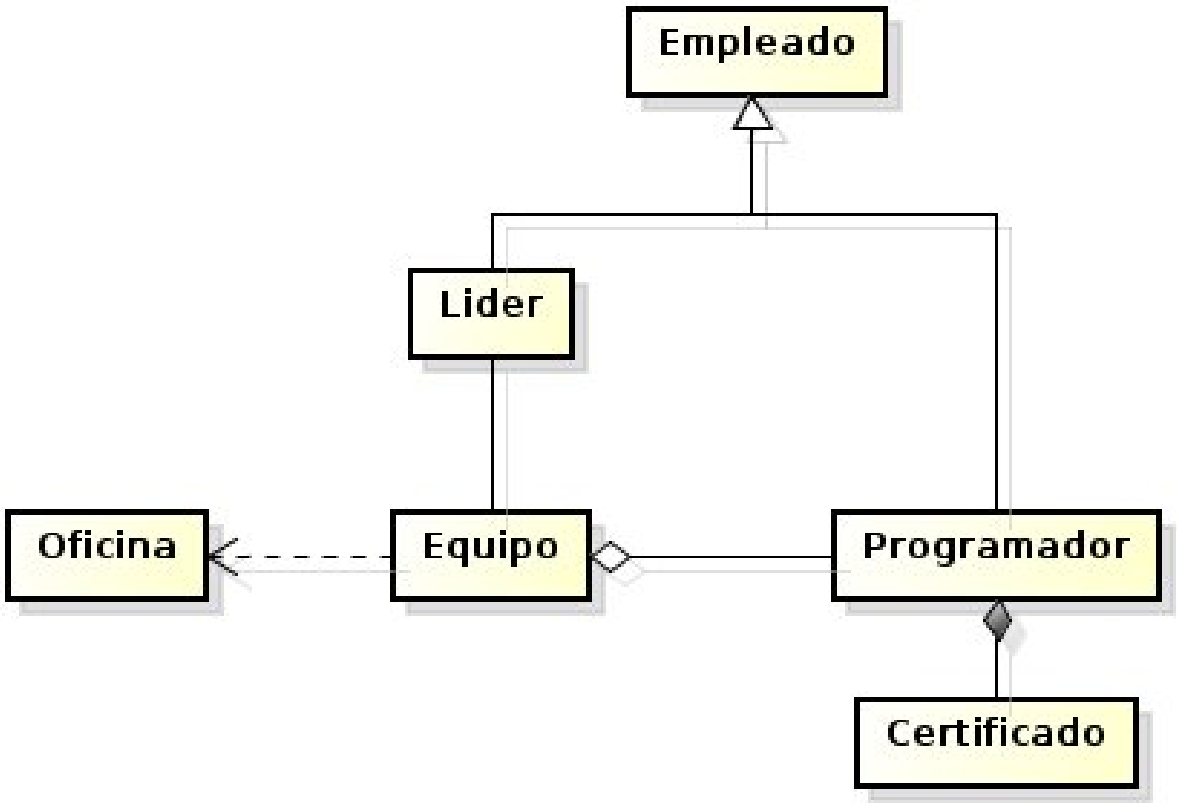
Herencia



¿Abstracción?

Las oficinas se encuentran en cinco sedes, cada una tiene un gerente

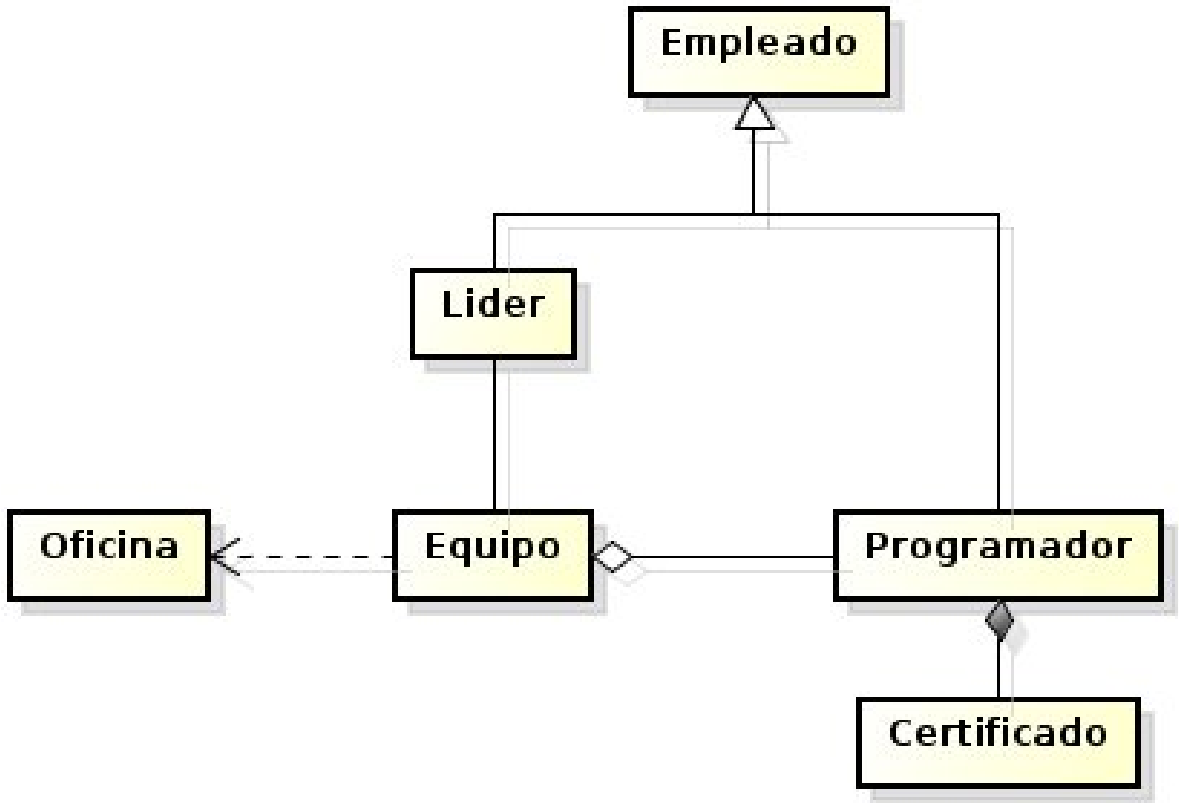
Herencia



¿Generalización?

Los lideres y programadores deben ser ingenieros de software

Herencia



¿Especialización?

Algunas oficinas son laboratorios

Ventajas

Ventajas

Reutilización de código.

Agenda

Definición

Estudiante

```
public class Student {
  private String name;
  private String major;
  // etc.

  public String getName() {
    return name;
  }

  public void setName(String n) {
      name = n;
  }



  // etc.
}
```

¿ Reversa?

Definición

Estudiante graduado

```
public class GraduateStudent extends Student {
  // Declare two new attributes above and beyond
  // what the Student class has already declared ...

  private String undergraduateDegree;
  private String undergraduateInstitution;

  // ... and accessor methods for each of these new attributes.

  public String getUndergraduateDegree {
      return undergraduateDegree;
  }

  public void setUndergraduateDegree(String s) {
      undergraduateDegree = s;
  }


  etc (5 más)

  // That's the ENTIRE GraduateStudent class declaration!
  // Short and sweet!
}
```

¿ Reversa?

Definición

Estudiante graduado

```
public class GraduateStudent extends Student {
  // Declare two new attributes above and beyond
  // what the Student class has already declared ...

  private String undergraduateDegree;
  private String undergraduateInstitution;

  // ... and accessor methods for each of these new attributes.

  public String getUndergraduateDegree {
      return undergraduateDegree;
  }

  public void setUndergraduateDegree(String s) {
      undergraduateDegree = s;
  }


  etc (5 más)

  // That's the ENTIRE GraduateStudent class declaration!
  // Short and sweet!
}
```

¿Qué métodos ofrece `GraduateStudent`?

Definición

Estudiante graduado

```
public class GraduateStudent extends Student {
  // Declare two new attributes above and beyond
  // what the Student class has already declared ...

  private String undergraduateDegree;
  private String undergraduateInstitution;

  // ... and accessor methods for each of these new attributes.

  public String getUndergraduateDegree {
      return undergraduateDegree;
  }

  public void setUndergraduateDegree(String s) {
      undergraduateDegree = s;
  }


  etc (5 más)

  // That's the ENTIRE GraduateStudent class declaration!
  // Short and sweet!
}
```

¿Qué atributos tiene un `GraduateStudent`?

Definición

Estudiante graduado

```
public class GraduateStudent extends Student {
  // Declare two new attributes above and beyond
  // what the Student class has already declared ...

  private String undergraduateDegree;
  private String undergraduateInstitution;

  // ... and accessor methods for each of these new attributes.

  public String getUndergraduateDegree {
      return undergraduateDegree;
  }

  public void setUndergraduateDegree(String s) {
      undergraduateDegree = s;
  }


  etc (5 más)

  // That's the ENTIRE GraduateStudent class declaration!
  // Short and sweet!
}
```

¿Qué atributos se pueden manipular en `GraduateStudent`?

Definición

Persona

```
public class Person {
  private String name;
  private String ssn;

  public Person(String n, String s) {

      setName(n);
      setSsn(s);

  }
}
```

¿ Reversa?

Definición

Persona

```
public class Person {
  private String name;
  private String ssn;

  public Person(String n, String s) {

      setName(n);
      setSsn(s);

  }
}
```

¿Qué podría cambiar?

Agenda

Creadores

Crear una estudiante

```
public class Student extends Person {
  private String major;

  public Student(String n, String s) {
    ...
        setName(n);
        setSsn(s);
        setMajor("UNDECLARED");
        emptyHistory();
  }

  public Student(String n, String s, String m) {
        setName(n);
        setSsn(s);
        setMajor(m);
        emptyHistory();

  }

}
```

¿REUTILIZANDO ?

Creadores

Crear una estudiante

```
public class Student extends Person {
  private String major;

  public Student(String n, String s) {
    ...
        this(n,s,"UNDECLARED");


  }
  public Student(String n, String s, String m) {
          super(n,s);
          setMajor(m);
          emptyHistory();

  }
}
```

Agenda

Sobreescritura

Escribir

```
1 public class Student extends Person {


    private String studentId;
    private String major     ;
    private double gpa;

    public void print() {
      System.out.println("Student Name:   " +      getName() + "\n" +
                         "Student No.:   " +      getStudentId() + "\n" -
                         "Major Field:   " +      getMajor () + "\n"
                         "GPA:   " +      getGpa());                    +
    }                                                                     +
}
```

Sobreescritura

Escribir

```
public class GraduateStudent extends Student {
  private String undergraduateDegree;
  private String undergraduateInstitution;

  public void print() {
       System.out.println("Student Name:  " +      getName() + "\n" +
                          "Student No.:  " +     getStudentId() + "\n" +
                          "Major Field:  " +      getMajorField() + "\n" +
                          "GPA:  " +      getGpa() + "\n" +
                          "Undergrad. Deg.:  " +      getUndergraduateDegree() +
                           "\n" + "Undergrad. Inst.:  " +
                               getUndergraduateInstitution());
    }
}
```

Sobreescritura

Escribir

```
public class GraduateStudent extends Student {
  private String undergraduateDegree;
  private String undergraduateInstitution;

  public void print() {
      System.out.println("Student Name:  " +      getName() + "\n" +
                         "Student No.:  " +     getStudentId() + "\n" +
                         "Major Field:  " +      getMajorField() + "\n" +
                         "GPA:  " +      getGpa() + "\n" +
                         "Undergrad. Deg.:  " +      getUndergraduateDegree() +
                          "\n" + "Undergrad. Inst.:  " +
                              getUndergraduateInstitution());
    }
}
```

`e.print()` ¿qué método ejecuta?

Sobreescritura

Escribir

```
public class GraduateStudent extends Student {
  private String undergraduateDegree;
  private String undergraduateInstitution;

  public void print() {
      System.out.println("Student Name:  " +      getName() + "\n" +
                         "Student No.:  " +     getStudentId() + "\n" +
                         "Major Field:  " +      getMajorField() + "\n" +
                         "GPA:  " +      getGpa() + "\n" +
                         "Undergrad. Deg.:  " +      getUndergraduateDegree() +
                          "\n" + "Undergrad. Inst.:  " +
                              getUndergraduateInstitution());
    }
}
```

¿REUTILIZANDO?

Sobreescritura

Escribir

```
1 public class Student extends Person {

    private String studentId;
    private String major     ;
    private double gpa;

   public void print() {
     System.out.println("Student Name:  " +      getName() + "\n" +
                        "Student No.:  " +      getStudentId() + "\n" -
                        "Major Field:  " +      getMajor () + "\n"
                        "GPA:  " +      getGpa());                    +
   }                                                                    +
 }
```

```
public class GraduateStudent extends Student {
  private String undergraduateDegree;
  private String undergraduateInstitution;

  public void print() {
       System.out.println("Student Name:  " +      getName() + "\n" +
                          "Student No.:  " +      getStudentId() + "\n" +
                          "Major Field:  " +      getMajorField() + "\n" +
                          "GPA:  " +      getGpa() + "\n" +
                          "Undergrad. Deg.:  " +      getUndergraduateDegree() +
                           "\n" + "Undergrad. Inst.:  " +
                               getUndergraduateInstitution());
    }
 }
```

Sobreescritura

Escribir

```
1 public class Student extends Person {


    private String studentId;
    private String major     ;
    private double gpa;

   public void print() {
     System.out.println("Student Name:  " +      getName() + "\n" +
                        "Student No.:  " +      getStudentId() + "\n" -
                        "Major Field:  " +      getMajor () + "\n"
                        "GPA:  " +      getGpa());
   }
 }
```

```
public class GraduateStudent extends Student {
  private String undergraduateDegree;
  private String undergraduateInstitution;

  public void print() {
       System.out.println("Student Name:  " +      getName() + "\n" +
                          "Student No.:  " +      getStudentId() + "\n" +
                          "Major Field:  " +      getMajorField() + "\n" +
                          "GPA:  " +      getGpa() + "\n" +
                          "Undergrad. Deg.:  " +      getUndergraduateDegree() +
                           "\n" + "Undergrad. Inst.:  " +
                               getUndergraduateInstitution());
    }
  }
```

`e.print()` ¿qué método ejecuta?

Sobreescritura

Escribir

```
1 public class Student extends Person {


    private String studentId;
    private String major     ;
    private double gpa;

   public void print() {
     System.out.println("Student Name:  " +      getName() + "\n" +
                        "Student No.:  " +      getStudentId() + "\n" -
                        "Major Field:  " +      getMajor () + "\n"
                        "GPA:  " +      getGpa());                    +
   }                                                                   +
 }
```

```
public class GraduateStudent extends Student {
  private String undergraduateDegree;
  private String undergraduateInstitution;

  public void print() {
       System.out.println("Student Name:  " +      getName() + "\n" +
                          "Student No.:  " +      getStudentId() + "\n" +
                          "Major Field:  " +      getMajorField() + "\n" +
                          "GPA:  " +      getGpa() + "\n" +
                          "Undergrad. Deg.:  " +      getUndergraduateDegree() +
                           "\n" + "Undergrad. Inst.:  " +
                               getUndergraduateInstitution());
    }
  }
```

¿REUTILIZANDO?

Sobreescritura

Escribir

```
public class GraduateStudent extends Student {

  public void print() {

    super.print();

    System.out.println("Undergrad. Deg.:  " + this.getUndergraduateDegree() + "\n" +
                       "Undergrad. Inst.:  " + this.getUndergraduateInstitution());
  }
}
```

Agenda

Acceso a características

Persona - Estudiante

```
public class Person {
   // etc.
   private int age;
}
```

```
public class Student extends Person {
  // Details omitted.

  public boolean isOver65( ) {
    if (age > 65) return true;
    else return false;
  }

  // Other details omitted.
}
```

¿Reversa?

Acceso a características

Persona - Estudiante

```
public class Person {
   // etc.
   private int age;
}
```

```
public class Student extends Person {
  // Details omitted.

  public boolean isOver65( ) {
    if (age > 65) return true;
    else return false;
  }

  // Other details omitted.
}
```

¿Problema? ¿Solución?

Acceso a caracteríticas

Persona - Estudiante

```
public class Person {
  // etc.
  protected int age;
}
```

```
public class Student extends Person {
  // Details omitted.

  public boolean isOver65( ) {
    if (age > 65) return true;
    else return false;
  }

  // Other details omitted.
}
```

¿Reversa?

Acceso a caracteríticas

Persona - Estudiante

```
public class Person {
  // etc.
  protected int age;
}
```

```
public class Student extends Person {
  // Details omitted.

  public boolean isOver65( ) {
    if (age > 65) return true;
    else return false;
  }

  // Other details omitted.
}
```

¿ *Refactoring* método isOver65?

Agenda

Finales

En clase

```
public final class GraduateStudent extends Student{

    ....

}
```

¿Qué indica `final class GraduateStudent` ?

Finales

En clase

```
public final class GraduateStudent extends Student{

    ....

}
```

```
public class PHDStudent extends GraduateStudent {

}
```

¿Qué pasa?

Finales

En clase

```
public final class GraduateStudent extends Student{

    ....

}
```

```
public class PHDStudent extends GraduateStudent {

}
```

cannot inherit from final GraduateStudent

Finales

En métodos

```
public class Student{

    protected double gpa;


    public final boolean isExcellent(){
        return (gpa>4.5);
    }
}
```

¿Qué indica `final isExcellent`?

Finales

En métodos

```
public class Student{

    protected double gpa;


    public final boolean isExcellent(){
        return (gpa>4.5);
    }
}
```

```
public final class GraduateStudent extends Student{

    public boolean isExcellent(){
        return (gpa>=4.0);
    }

}
```

¿Qué pasa?

Finales

En métodos

```
public class Student{

    protected double gpa;


    public final boolean isExcellent(){
        return (gpa>4.5);
    }
}
```

```
public final class GraduateStudent extends Student{

    public boolean isExcellent(){
        return (gpa>=4.0);
    }

}
```

Find: uate}\end{center} Prev Next Match Case Replace

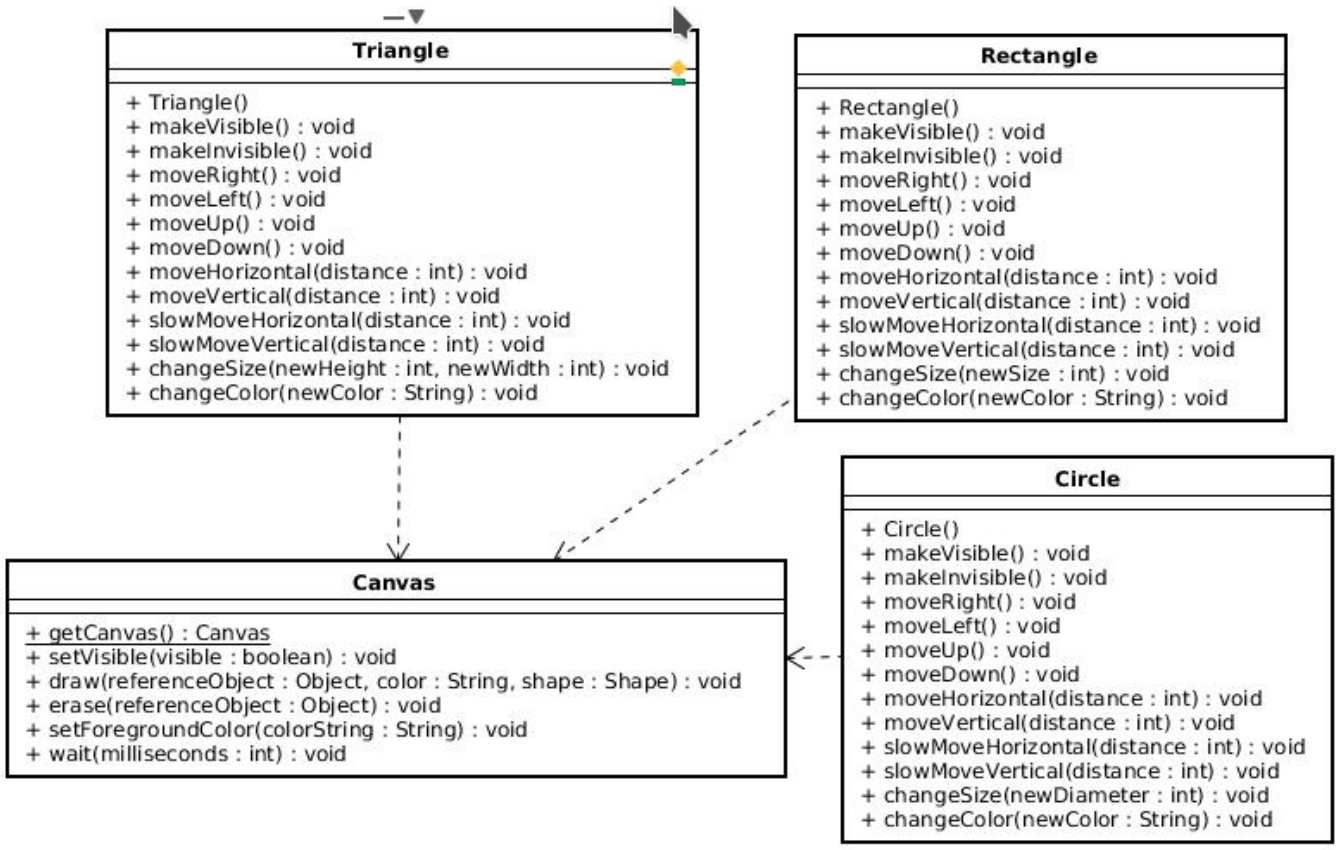
isExcellent() in GraduateStudent cannot override isExcellent() in Student; overr
idden method is final

Agenda

Shapes

Mejor estructura

- Simplificando



Shapes

¿Atributos?

Circle

```
public class Circle
{
    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;
```

Triangle

```
public class Triangle
{
    private int height;
    private int width;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;
```

Rectangle

```
public class Rectangle{

    private int height;
    private int width;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;
```

Shapes

¿Creadores?

Circle

```
public Circle()
{
    diameter = 30;
    xPosition = 20;
    yPosition = 60;
    color = "blue";
    isVisible = false;
}
```

Rectangle

```
public Rectangle(){
    height = 30;
    width = 40;
    xPosition = 70;
    yPosition = 15;
    color = "magenta";
    isVisible = false;
}
```

Triangle

```
public Triangle()
{
    height = 30;
    width = 40;
    xPosition = 50;
    yPosition = 15;
    color = "green";
    isVisible = false;
}
```

Shapes

¿Moverse lentamente?

Circle

```
public void slowMoveVertical(int distance)
{
    int delta;

    if(distance < 0)
    {
        delta = -1;
        distance = -distance;
    }
    else
    {
        delta = 1;
    }

    for(int i = 0; i < distance; i++)
    {
        yPosition += delta;
        draw();
    }
}
```

Rectangle

```
public void slowMoveHorizontal(int distance)
{
    int delta;

    if(distance < 0)
    {
        delta = -1;
        distance = -distance;
    }
    else
    {
        delta = 1;
    }

    for(int i = 0; i < distance; i++)
    {
        xPosition += delta;
        draw();
    }
}
```

Triangle

```
public void slowMoveHorizontal(int distance)
{
    int delta;

    if(distance < 0)
    {
        delta = -1;
        distance = -distance;
    }
    else
    {
        delta = 1;
    }

    for(int i = 0; i < distance; i++)
    {
        xPosition += delta;
        draw();
    }
}
```

Shapes

¿Pintarse?

Circle

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

Triangle

```
private void draw()
{
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        int[] xpoints = { xPosition, xPosition + (width/2), xPosition - (width/2) };
        int[] ypoints = { yPosition, yPosition + height, yPosition + height };
        canvas.draw(this, color, new Polygon(xpoints, ypoints, 3));
        canvas.wait(10);
    }
}
```

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);
    }
}
```

Shapes

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

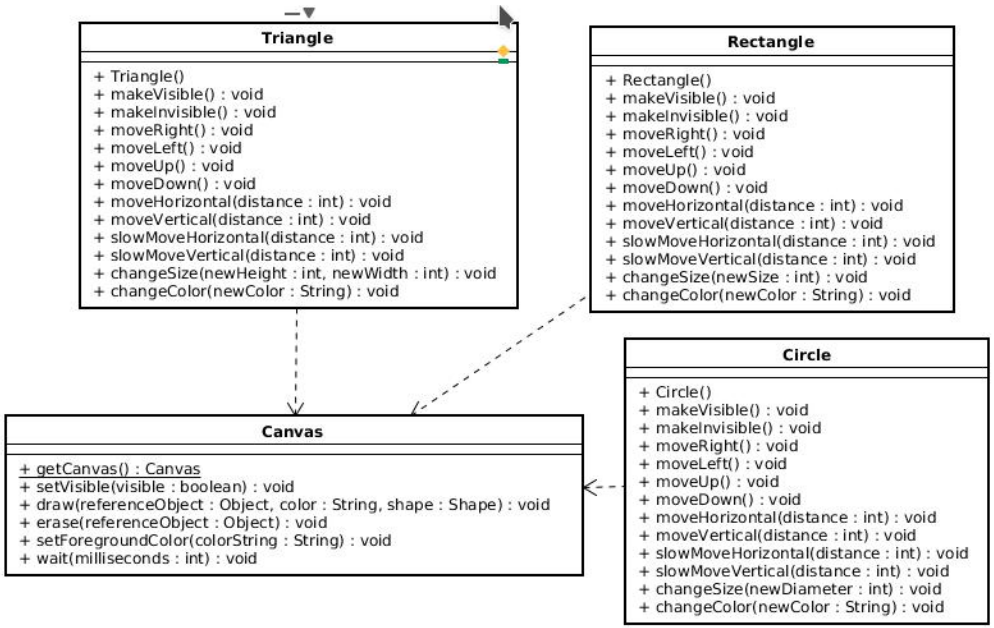
Agenda

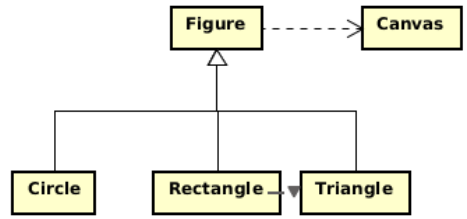
Extensión

Nueva figura

- Línea

shapes



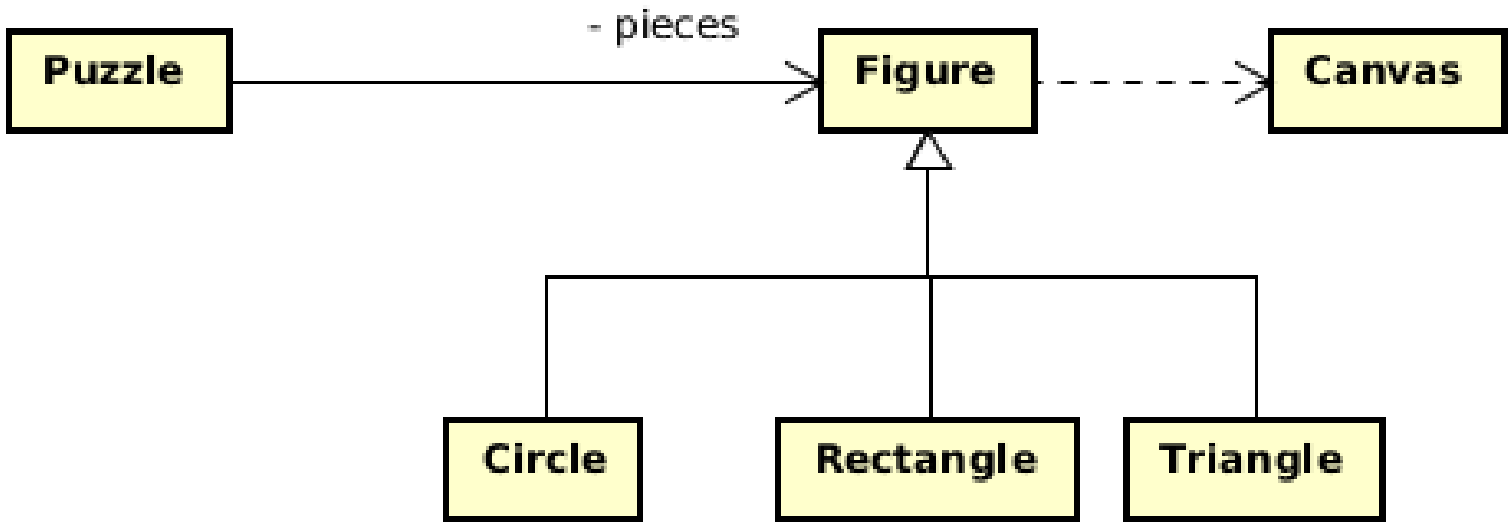


Agenda

Nueva clase

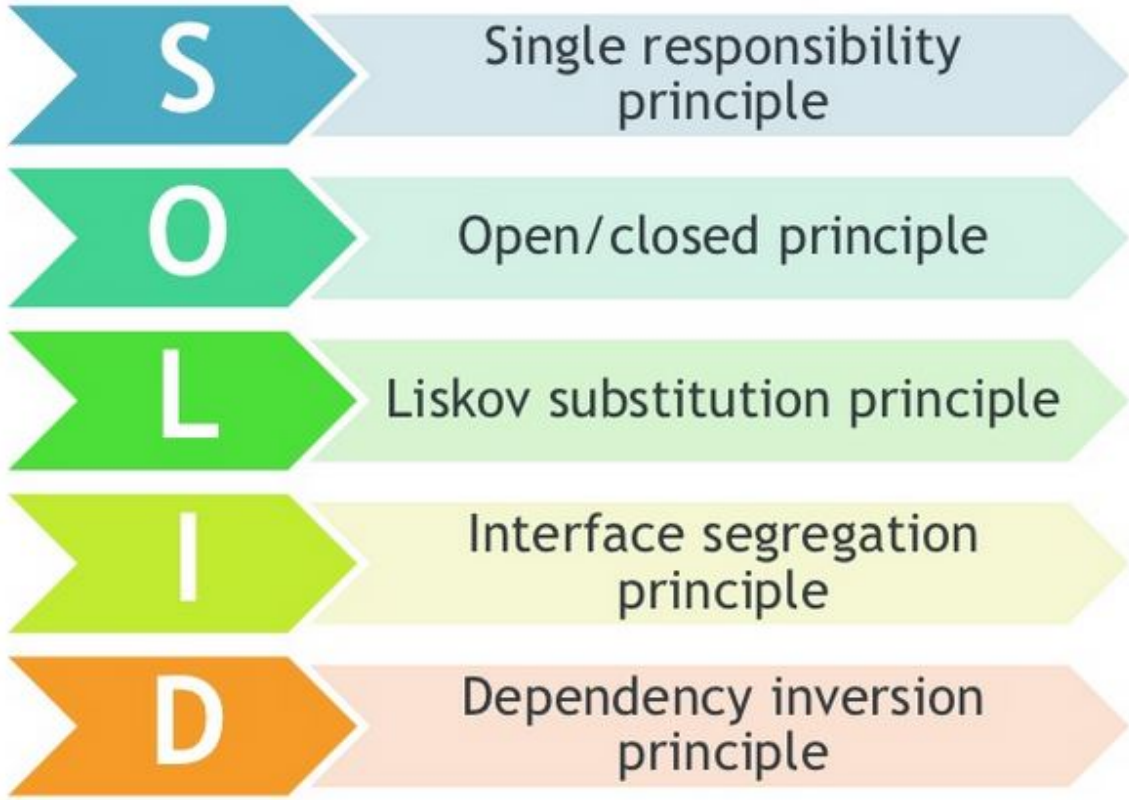
- hacerla visible
- calcular el area

Puzzle



Agenda

SOLID- Principios básicos



S : Primer tercio

O : Segundo tercio

LID: CVDS

Agenda

Batalla naval

Mejor estructura

- Simplificando