Programación Orientada a Objetos

Relaciones entre objetos

CEIS

2022-01

Agenda

Relaciones

Herencia

- Definición
- Creadores
- Sobreescritura
- Visibilidad - Mutabiliad
- Ventajas y Principios

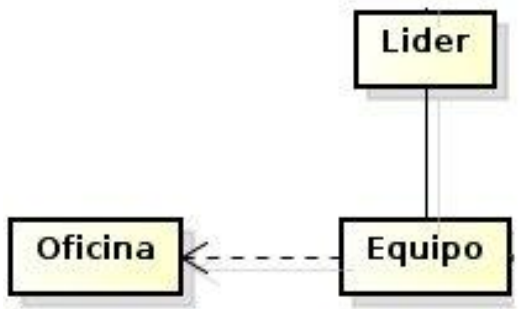
Caso: Shapes

- Estructura
- Atributos
- Métodos

Caso: Batalla naval

Relaciones

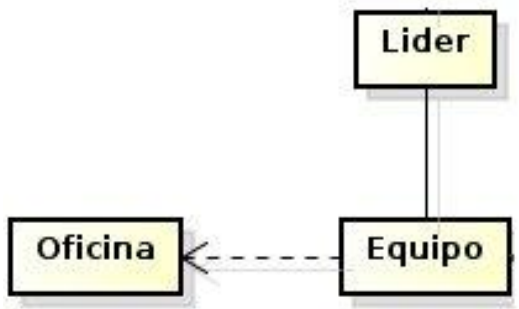
Unidad de proyectos



¿Qué leemos?

Relaciones

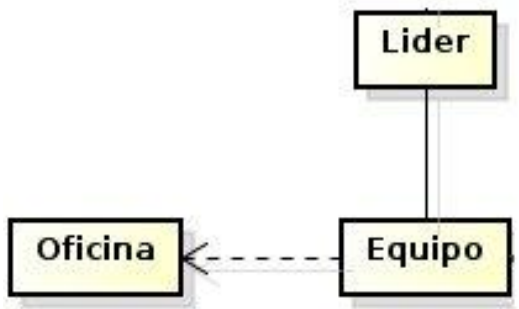
Unidad de proyectos



¿Tipos de relaciones?

Relaciones

Unidad de proyectos

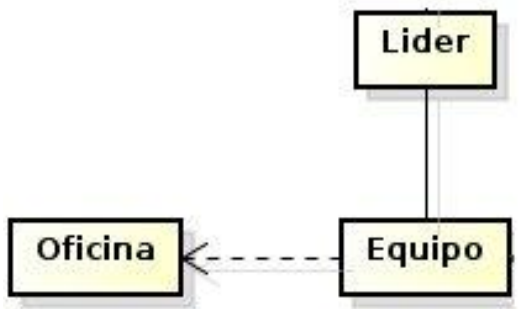


¿Tipos de relaciones?

Relaciones estructurales (atributo). Relaciones de comportamiento(uso)

Relaciones

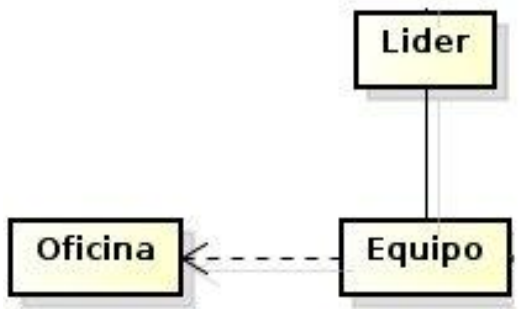
Unidad de proyectos



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

Relaciones

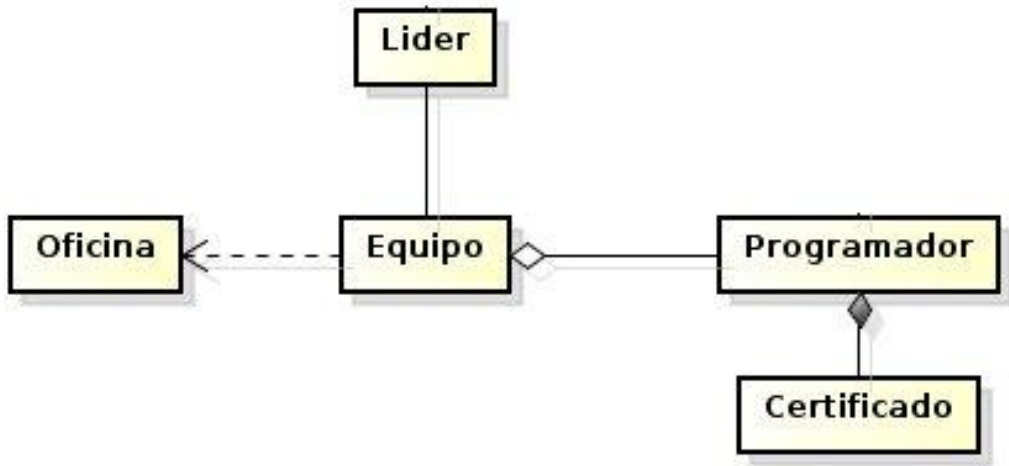
Unidad de proyectos



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

Relaciones

Unidad de proyectos

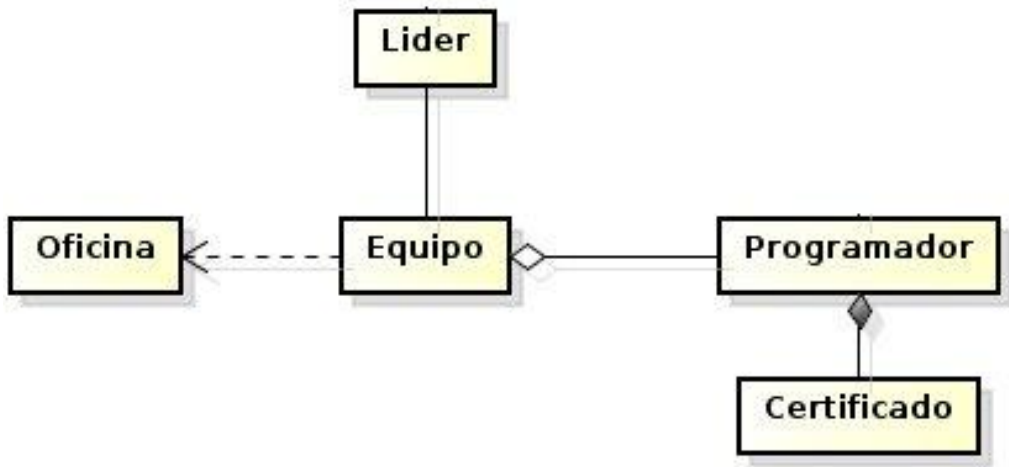


RELACIONES ESTRUCTURALES

¿Qué leemos?

Relaciones

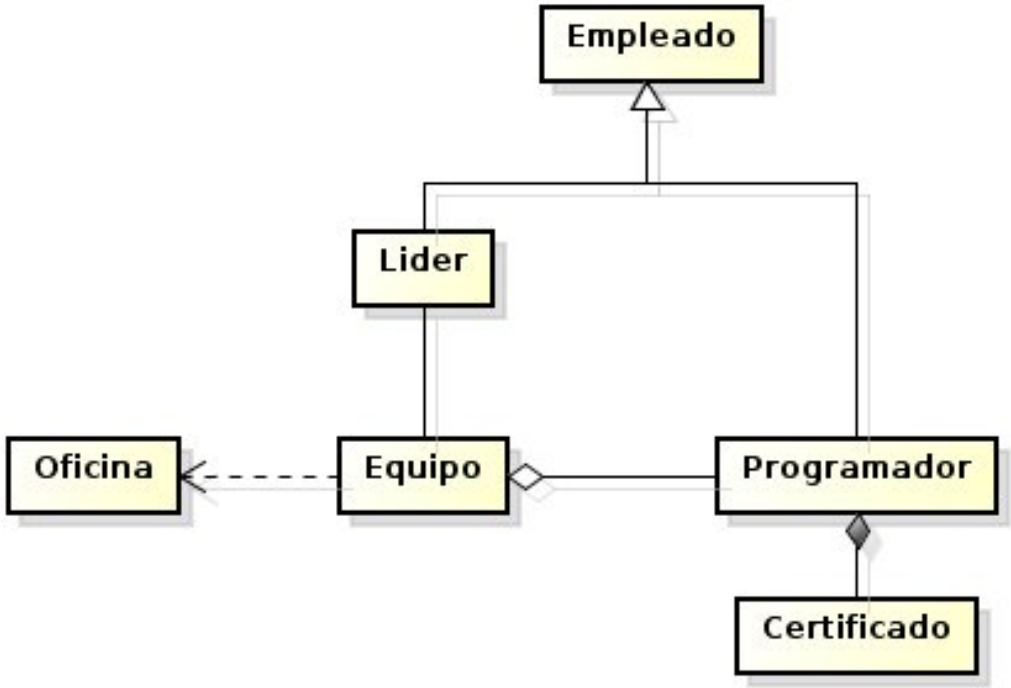
Unidad de proyectos



RELACIONES ESTRUCTURALES
RELACIONES TODO-PARTE
¿Agregación (debil)? ¿Composición (fuerte)?
Un equipo está compuesto de varios programadores, un programador pertenece a un único equipo. El equipo es quien conoce sus programadores.
Los certificados son de cada programador. El programador conoce sus certificados y el certificado su dueño.

Relaciones

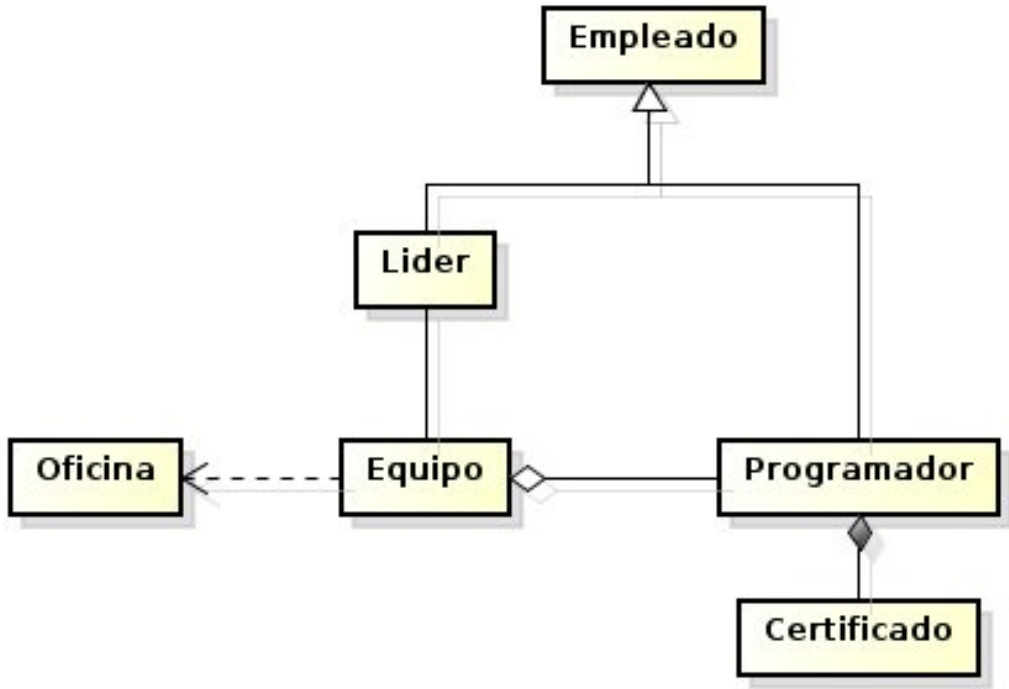
Unidad de proyectos



¿Qué leemos?

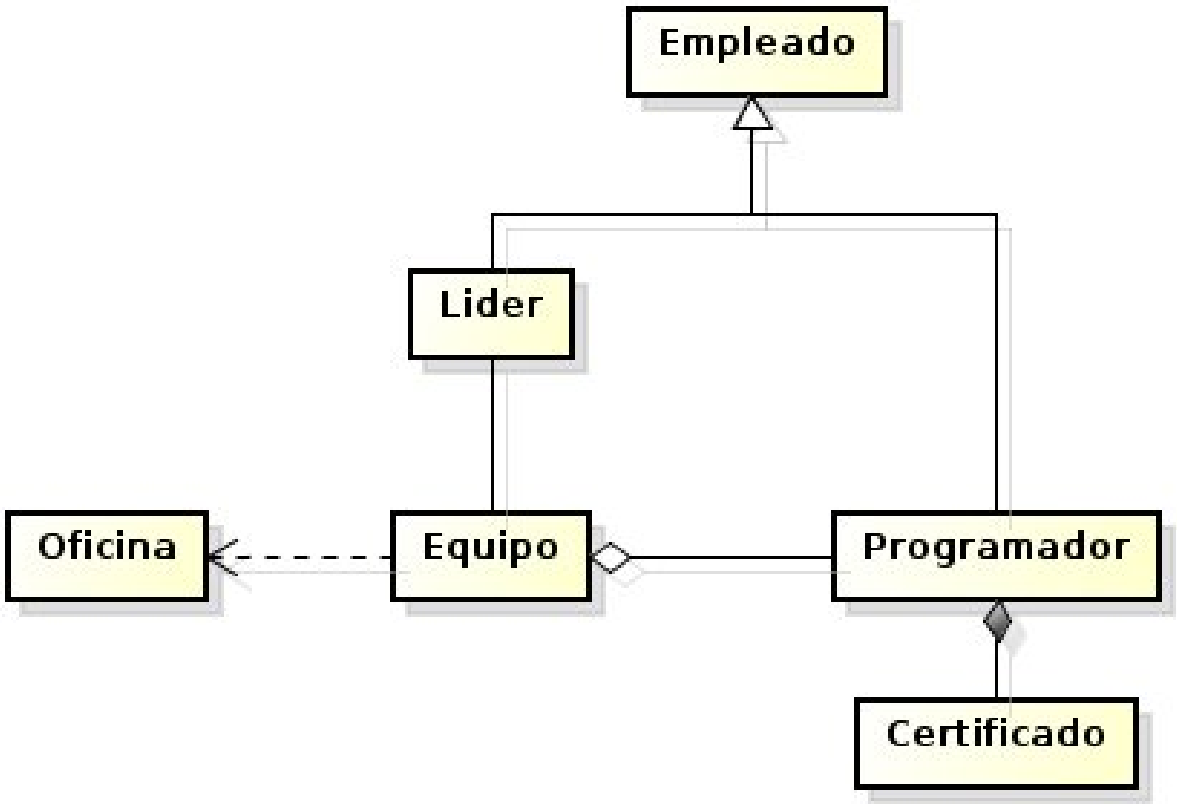
Relaciones

Unidad de proyectos



RELACIONES DE HERENCIA-Es Un

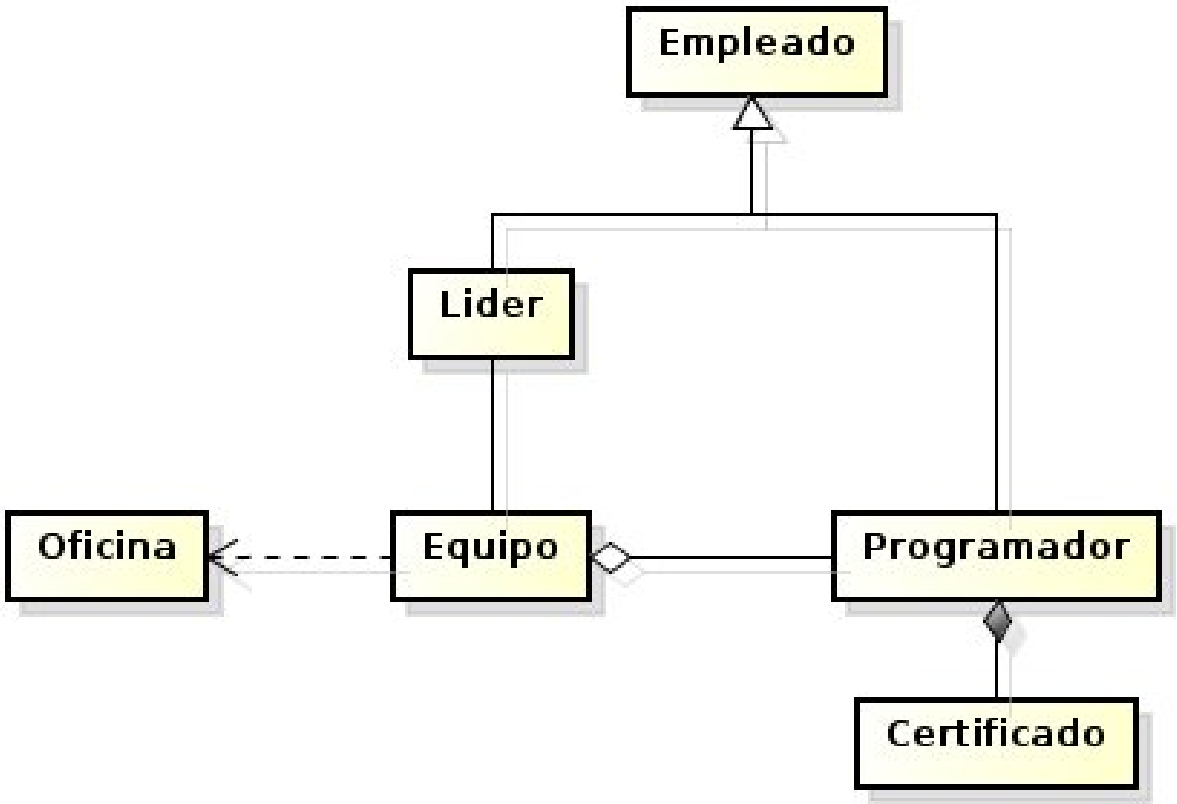
Herencia



¿Abstracción?

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

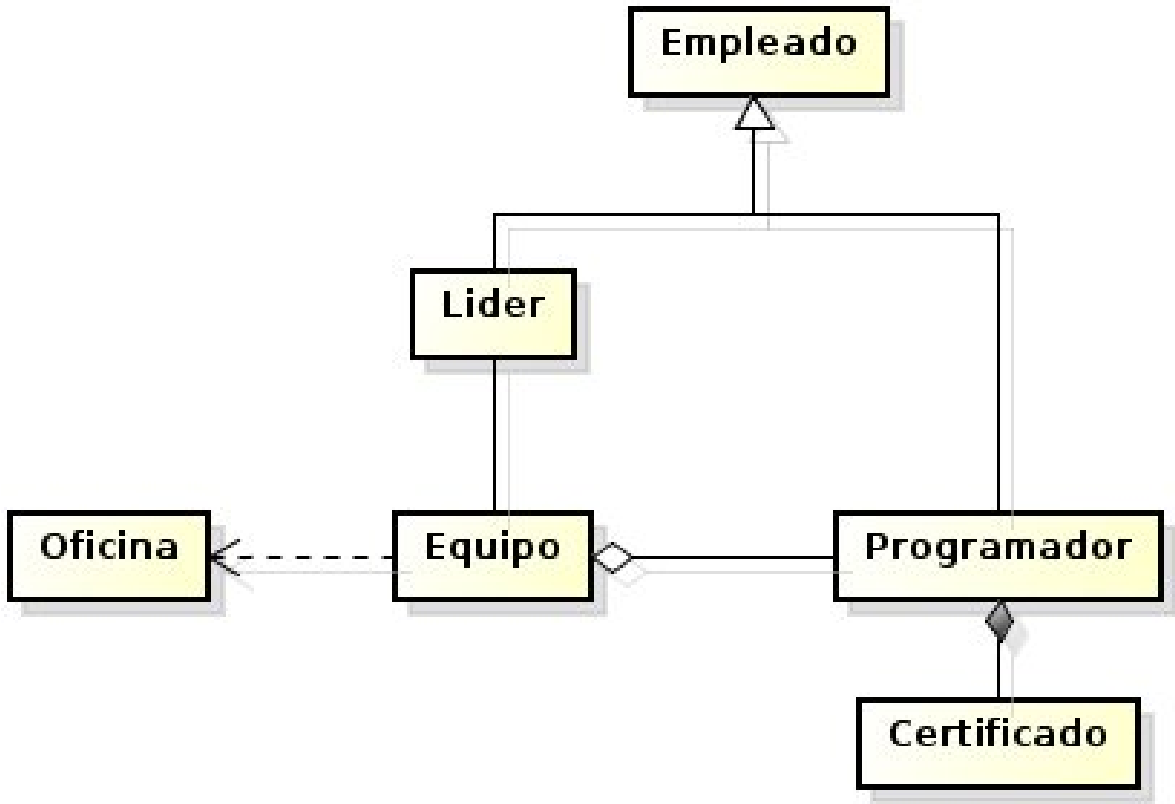
Herencia



¿Generalización?

Los lideres y programadores deben ser ingenieros de software

Herencia



¿Especialización?

Algunas oficinas son laboratorios

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.
}
```

¿Ingeniería reversa?

1. ¿Cómo quedaría el diseño? (Diagrama de clases)

Herencia

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;

   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!
}
```

1. ¿Cómo quedaría el diseño? (Diagrama de clases)

Herencia

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;

  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!
}
```

1. ¿Qué atributos tiene un `GraduateStudent`?
2. ¿Qué atributos se pueden usar en `GraduateStudent`?

Herencia

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;

   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!
}
```

1. ¿Qué métodos ofrece `GraduateStudent`?

Definición

Persona

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

  public Person(String n, String s) {
    .
    .
    .
  }
}
```

1. ¿Cómo quedaría el diseño? (Diagrama de clases)

Definición

Persona

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

  public Person(String n, String s) {

  }
}
```

1. ¿Cómo quedaría el diseño? (Diagrama de clases)
2. ¿Qué podríamos cambiar? (Diagrama de clases)

Agenda

Creadores

Crear un 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();
  }
}
```

Refactorizando

1. ¿Cómo sería un mejor código? Dos toques (Código)

Creadores

Crear un 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();
  }
}
```

```
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();

  }
}
```

Refactorizando

1. ¿Cómo sería un mejor código? Dos toques (Código)

Agenda

Sobreescritura

Imprimir un estudiante

```
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.:  " +      getUndergraduateDegre
                          "\n" + "Undergrad. Inst.:  " +
                              getUndergraduateInstitution());
    }
}
```

Explorando

1. ¿Cómo quedarían en diseño? (Diagrama de clases)
2. `e.print()` ¿Cúal método ejecuta?
3. ¿Cómo se podría refactorizar?

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());
      }
    }
```

```
public class GraduateStudent extends Student {

  public void print() {

    super.print();

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

Agenda

- Relaciones
- Herencia
  - Definición
  - Creadores
  - Sobreescritura
  - Visibilidad - Mutabiliad
  - Ventajas y Principios
- Caso: Shapes
  - Estructura
  - Atributos
  - Métodos
- Caso: Batalla naval

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.
}
```

1. ¿Cómo quedarían en diseño? (Diagrama de clases)
2. ¿Cuál es el problema? ¿Cuál es la solución?

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.
}
```

```
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.
}
```

1. ¿Cómo quedarían en diseño? (Diagrama de clases)
2. ¿Cuál es el problema? ¿Cuál es la solución?

Visibilidad

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.
}
```

Refactorizando

1. ¿Cómo sería un mejor código? (Código)

Finales

En clase

```
public final class GraduateStudent extends Student{

    ....

}
```

Analizando

1. ¿Qué indica `final class GraduateStudent`?

Finales

En clase

```
public final class GraduateStudent extends Student{

    ....

}
```

```
public class PHDStudent extends GraduateStudent {


}
```

Analizando

1. ¿Qué indica `final class GraduateStudent`?
2. ¿Qué pasaría?

Finales

En clase

```
public final class GraduateStudent extends Student{

    ....

}
```

```
public class PHDStudent extends GraduateStudent {

}
```

cannot inherit from final GraduateStudent

Finales

En clase

```
public class Student{

    protected double gpa;

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

Analizando

1. ¿Qué indica `final isExcellent`?

Finales

En clase

```
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);
    }

}
```

Analizando

1. ¿Qué indica `final isExcellent`?
2. ¿Qué pasaría?

Finales

En clase

```
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: ɹate}\end{center} Prev Next Match Case Replace

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

Agenda

- Relaciones
- Herencia
  - Definición
  - Creadores
  - Sobreescritura
  - Visibilidad - Mutabiliad
  - Ventajas y Principios
- Caso: Shapes
  - Estructura
  - Atributos
  - Métodos
- Caso: Batalla naval

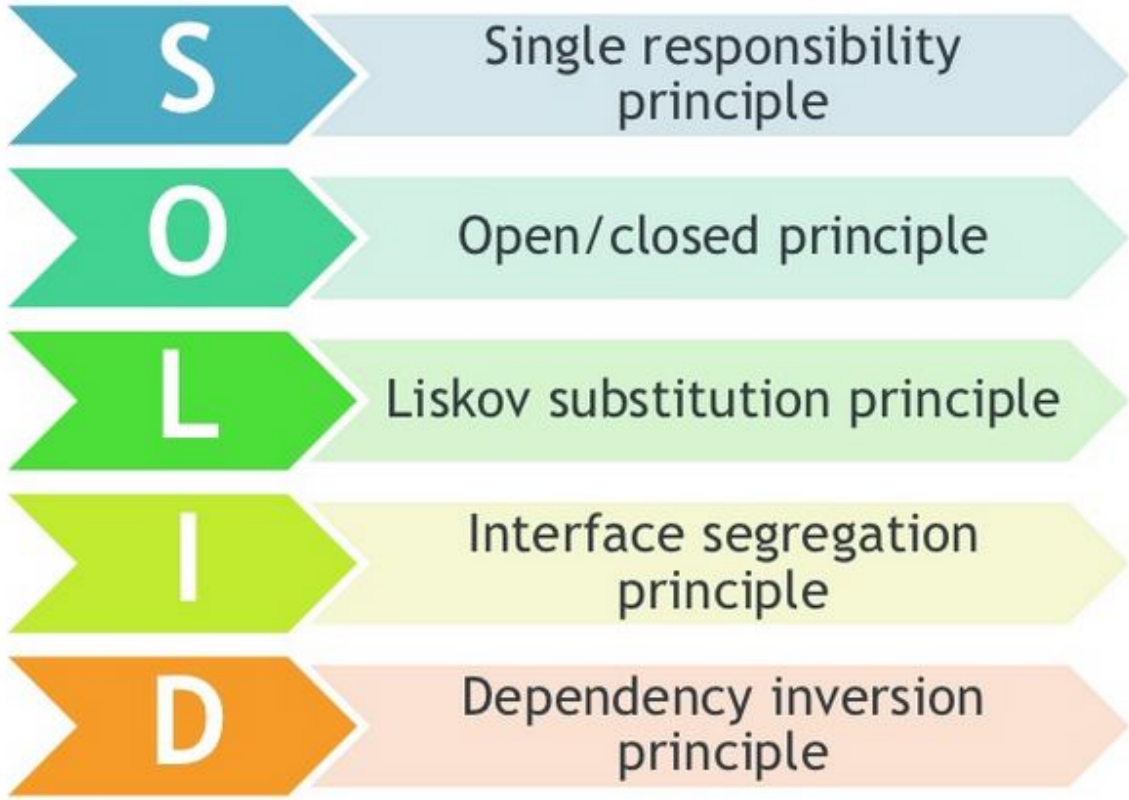
Ventajas

Ventajas

Reutilización

Extensibilidad

SOLID- Principios básicos



S : Primer tercio

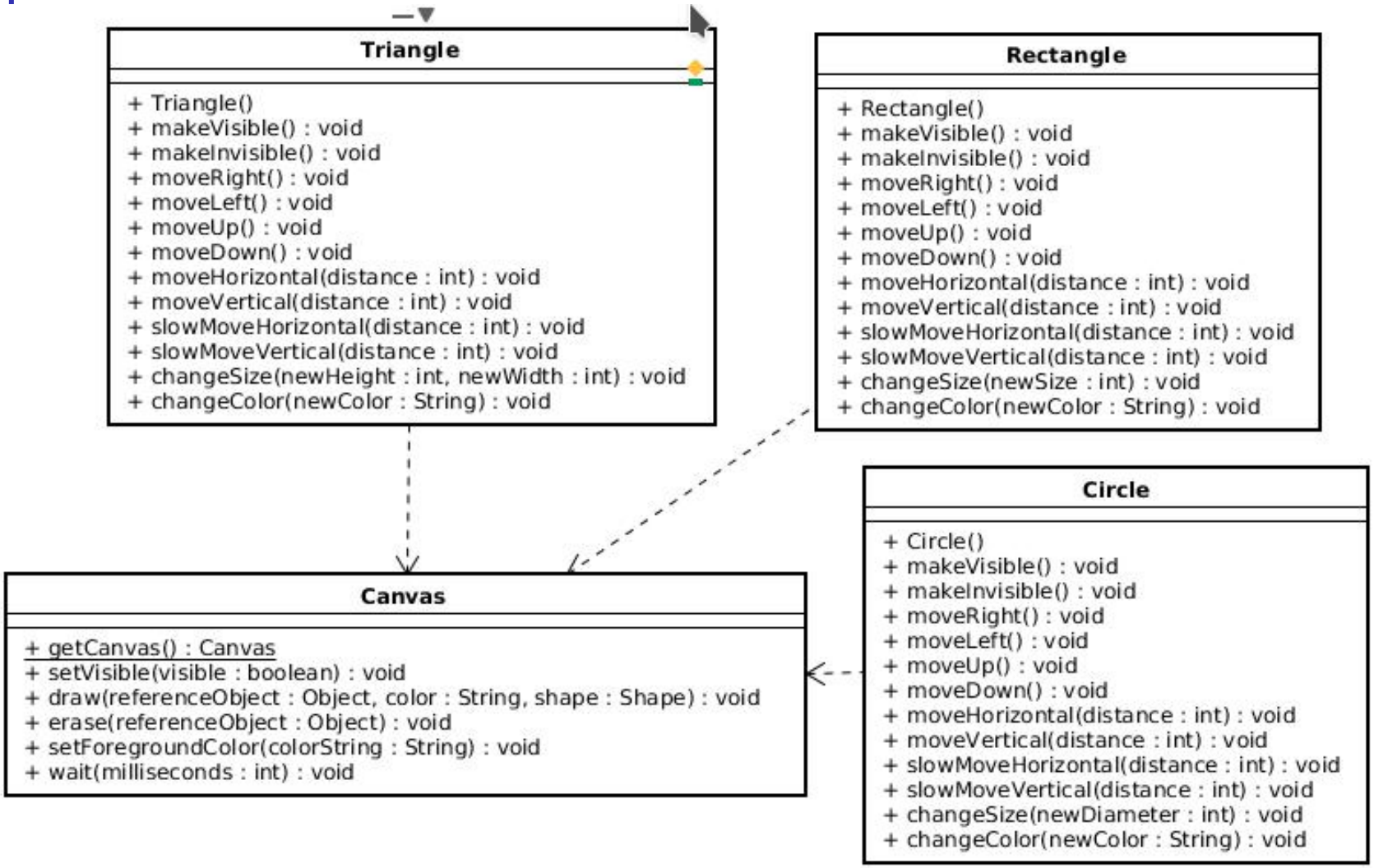
O : Segundo tercio

LID: CVDS

Agenda

- Relaciones
- Herencia
  - Definición
  - Creadores
  - Sobreescritura
  - Visibilidad - Mutabiliad
  - Ventajas y Principios
- Caso: Shapes
  - Estructura
  - Atributos
  - Métodos
- Caso: Batalla naval

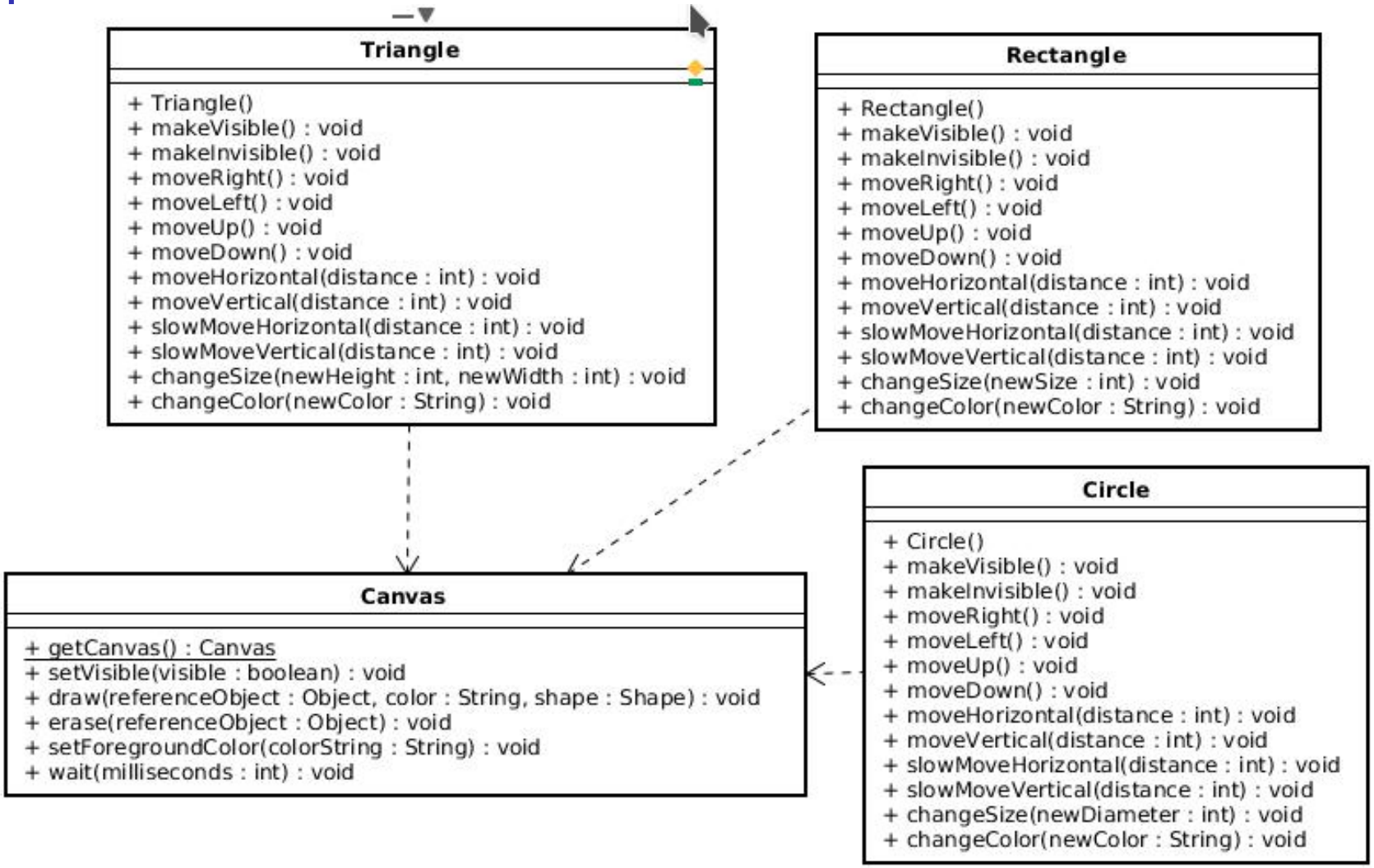
Shapes



¿Estructura?

1. ¿Cuál sería una mejor estructura? (Diagrama de clases sin atributos ni métodos)

Shapes



¿Estructura?

1. ¿Cuál sería una mejor estructura? (Diagrama de clases sin atributos ni métodos)
2. Idealmente, ¿cuántas clases debería usar a `Canvas`? (¿Cuál(es)?¿Por qué?)

Agenda

Shapes

Circle

```
public class Circle
{
    private int diameter;
    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;
```

Triangle

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

¿Atributos?

1. ¿Cuáles serían los atributos de la nueva clase? (Diagrama de clases con atributos)
2. ¿Cuáles atributos quedan en las subclases? (Diagrama de clases con atributos)

Agenda

Shapes

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;
}
```

¿Creadores?

1. ¿Cuál sería el creador de la superclase? (Código)
2. ¿Cómo quedarán los creadores de las subclases? (Código)

Shapes

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();
    }
}
```

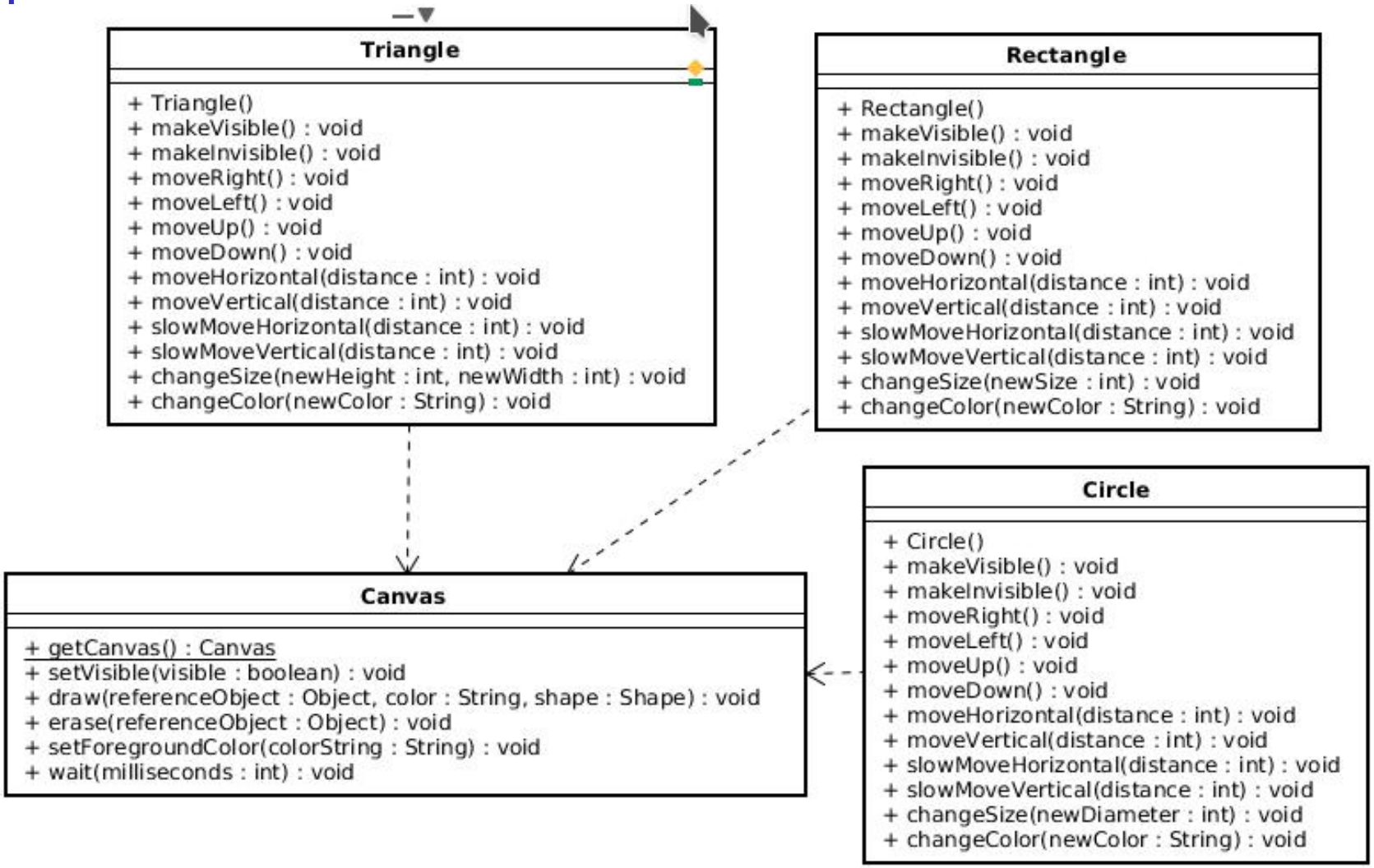
Triangle

```
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();
    }
}
```

¿Moverse lentamente?

1. ¿Dónde debería quedar este método? (Diagrama de clases)

Shapes



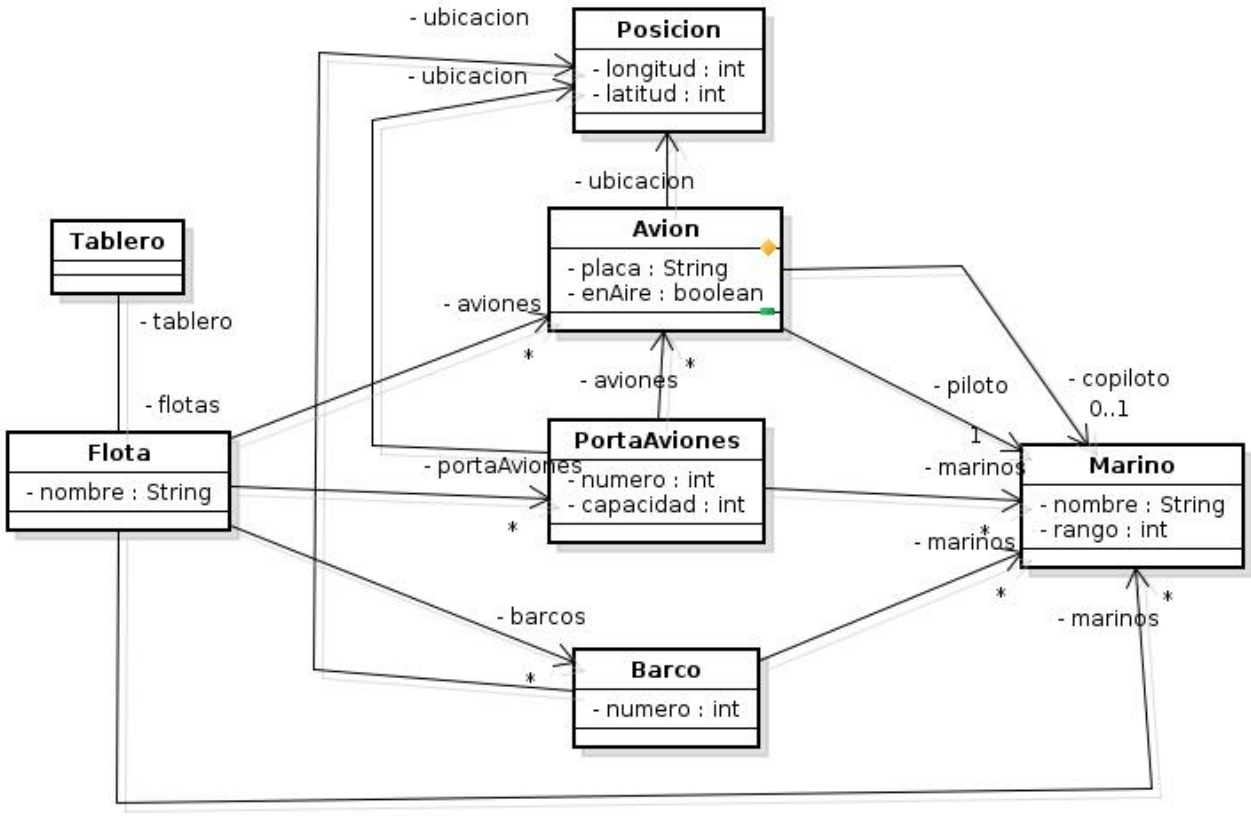
¿Estructura?

1. ¿Cuáles métodos podrian quedar en la superclase? (Diagrama de clases)
2. ¿Cuáles métodos deberían quedar en las subclases? (Digrama de clases)

Agenda

- Relaciones
- Herencia
  - Definición
  - Creadores
  - Sobreescritura
  - Visibilidad - Mutabiliad
  - Ventajas y Principios
- Caso: Shapes
  - Estructura
  - Atributos
  - Métodos
- Caso: Batalla naval

Batalla naval



¿Estructura?

1. ¿Cuál sería una mejor estructura? (Diagrama de clases con atributos y métodos)