# Programación Orientada a Objetos

## Lo abstracto y las interfaces

CEIS

2022-1

# Agenda

- Lo abstracto
  - Universidad
  - Otros ejemplos
- Interfaz
  - Universidad
  - Otros ejemplos
- Shapes
  - Refactorización
  - Extensión
  - Manipulando
  - Uso
- Batalla naval
  - Estructura
  - Métodos

# Agenda

# Lo abstracto

## Cursos

```
public abstract class Course {
    // Details omitted.
     public  void enrollStudent(Student s) {
         enrolledStudents.add(s);
     }
     public  final  void assignInstructor(Professor p) {
         setInstructor(p);
     }
     public void initializeCourse(Professor p, String s, String e) {
         assignInstructor(p);
         reserveClassroom( );
         establishCourseSchedule(s, e);
    }
    public abstract void establishCourseSchedule(String startDate, String endDate);
}
```

- ¿Qué se está diciendo? (de la clase, de los métodos)

  Reversa

# Lo abstracto

## Cursos

```
public abstract class Course {
    // Details omitted.
    public void enrollStudent(Student s) {
        enrolledStudents.add(s);
    }
    public final void assignInstructor(Professor p) {
        setInstructor(p);
    }
    public void initializeCourse(Professor p, String s, String e) {
        assignInstructor(p);
        reserveClassroom();
        establishCourseSchedule(s, e);
    }
    public abstract void establishCourseSchedule(String startDate, String endDate);
}
```

¿Qué métodos

- a) deben ser igual para todos?

# Lo abstracto

## Cursos

```
public abstract class Course {
    // Details omitted.
     public void enrollStudent(Student s) {
         enrolledStudents.add(s);
     }
     public final void assignInstructor(Professor p) {
         setInstructor(p);
     }
     public void initializeCourse(Professor p, String s, String e) {
         assignInstructor(p);
         reserveClassroom();
         establishCourseSchedule(s, e);
     }
     public abstract void establishCourseSchedule(String startDate, String endDate);
}
```

¿Qué métodos

- a) deben ser igual para todos? b) pueden ser igual para todos ?

# Lo abstracto

## Cursos

```
public abstract class Course {
    // Details omitted.
     public void enrollStudent(Student s) {
         enrolledStudents.add(s);
     }
     public final void assignInstructor(Professor p) {
         setInstructor(p);
     }
     public void initializeCourse(Professor p, String s, String e) {
         assignInstructor(p);
         reserveClassroom();
         establishCourseSchedule(s, e);
     }
     public abstract void establishCourseSchedule(String startDate, String endDate);
}
```

¿Qué métodos

- a) deben ser igual para todos? b) pueden ser igual para todos ?
- c) son totalmente diferentes para todos ?

# Lo abstracto

## Cursos

```
public abstract class Course {
    // Details omitted.
     public void enrollStudent(Student s) {
         enrolledStudents.add(s);
     }
     public final void assignInstructor(Professor p) {
         setInstructor(p);
     }
     public void initializeCourse(Professor p, String s, String e) {
         assignInstructor(p);
         reserveClassroom();
         establishCourseSchedule(s, e);
     }
     public abstract void establishCourseSchedule(String startDate, String endDate);
}
```

¿Qué métodos

- a) deben ser igual para todos? b) pueden ser igual para todos ?
- c) son totalmente diferentes para todos ?
- d) tienen algunas partes iguales para todos ?

# Lo abstracto

Cursos- ¿Crear?

```
Course c = new Course();  // Impossible!
```

Here's the error message:

```
Course is abstract; cannot be instantiated
```

No se permite crear un objeto de una clase abstracta

# Lo abstracto

## Cursos- ¿Crear?

```
Course c = new Course();  // Impossible!
```

Here's the error message:

```
Course is abstract; cannot be instantiated
```

No se permite crear un objeto de una clase abstracta

## Cursos - ¿Manipular?

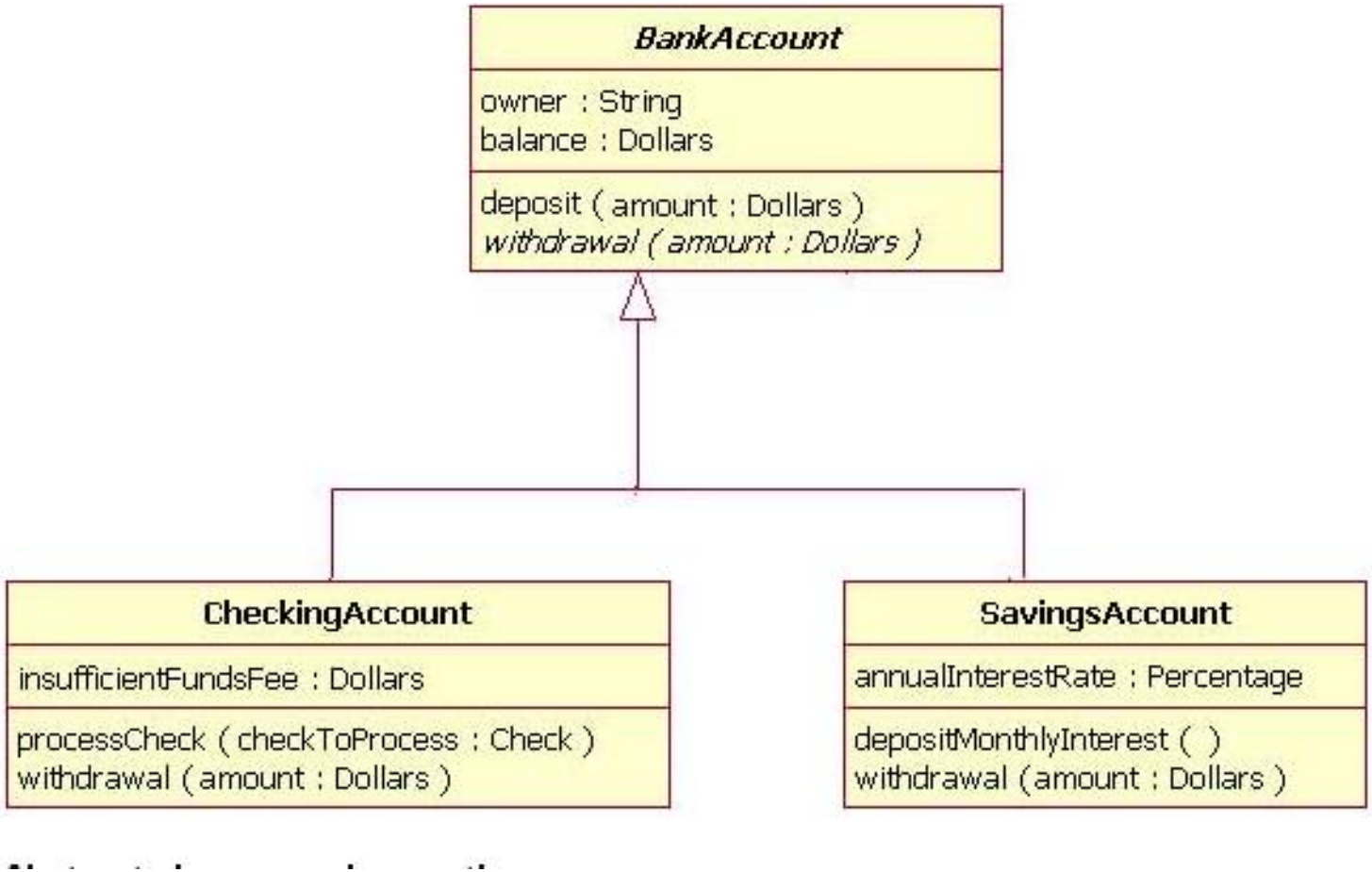
```
ArrayList<Course> courses = new ArrayList<Course>();
// Add a variety of different Course types to the collection.
courses.add(new LectureCourse());
courses.add(new LabCourse());
courses.add(new IndependentStudyCourse());
// etc.
for (Course c : courses) {
    // This next line of code is polymorphic.
    c.establishCourseSchedule("1/24/2005", "5/10/2005");
}
```

Se pueden almacenar, representa a sus subclases.

Se pueden usar los métodos, si todas las subclases lo tienen

# Agenda

# UML




- ¿Qué se está diciendo?

# Manual de referencia. Java.

```
abstract class Point {
        int x = 1, y = 1;
        void move(int dx, int dy) {
                x += dx;
                y += dy;
                alert();
        }
        abstract void alert();
}
abstract class ColoredPoint extends Point {
        int color;
}

class SimplePoint extends Point {
        void alert() { }
}
```

- ¿Qué se está diciendo?

  Reversa

# Agenda

# Personas y responsabilidades

## Para los que investigan

```java
import java.time.LocalDate;

public interface Investigator{

    boolean approveSyllabus(Syllabus s);

}
```

- ¿Qué se está diciendo ?

  Reversa

# Personas y responsabilidades

## Para los que investigan

```java
import java.time.LocalDate;

public interface Investigator{

    boolean approveSyllabus(Syllabus s);

}
```

- ¿Qué se está diciendo ?

  Reversa
- ¿Cómo digo que un estudiante de PHD puede investigar?

  ¿Qué implica lo anterior?

# Personas y responsabilidades

## Para los que enseñan

```java
import java.time.LocalDate;

public interface Teacher{

    final int WEEKEND = 50;

    void designedTextbook(TextBook t, Course c);

    Syllabus defineSyllabus(Course c);

    int getHourlyRate();

    default int teachingHourlyRate(int dayOfWeek){
        return (int)(getHourlyRate()*(2+(dayOfWeek>6 ? (WEEKEND/100): 0)));
    }
}
```

- ¿Qué se está diciendo ?

  Reversa

# Personas y responsabilidades

## Para los que enseñan

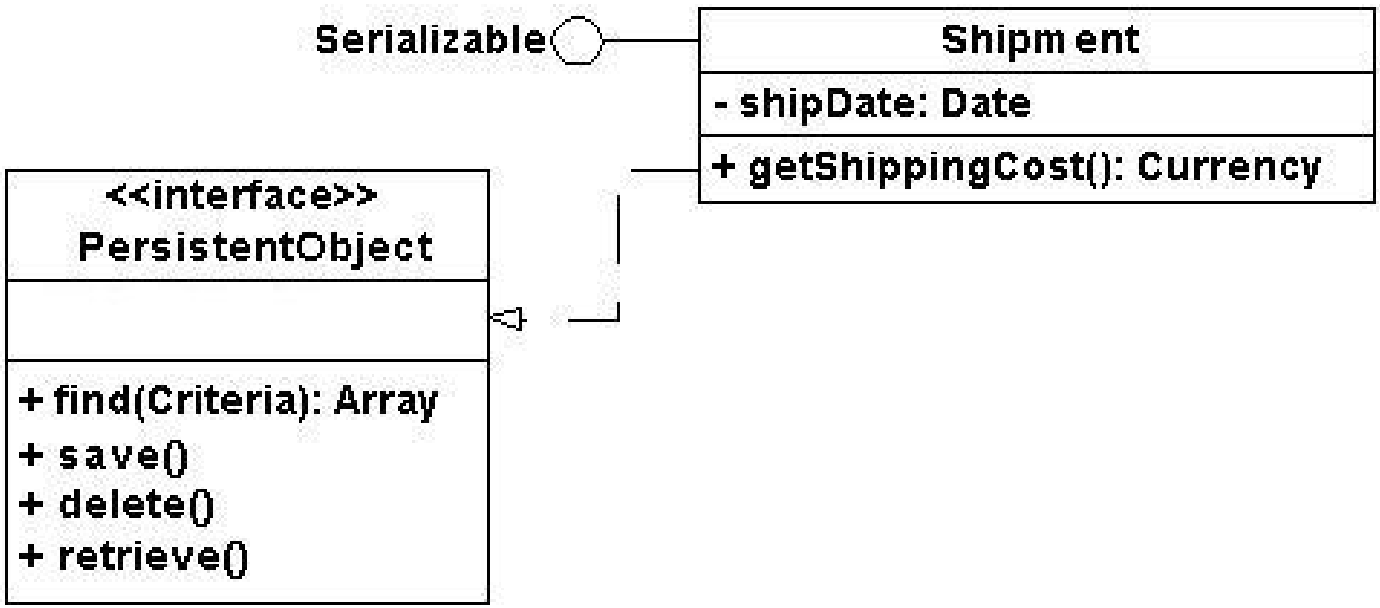
```
public class Professor extends Person implements Teacher,Investigator{
    private static final int HOURLY_RATE = 300000;

    public void designedTextbook(TextBook t, Course c){
        ...
    }

    public Syllabus defineSyllabus(Course c){
        ...
    }

    public boolean approveSyllabus(Syllabus s){
        ...
    }

    public int getHourlyRate(){
        return HOURLY_RATE;
    }

    public int todayClassPayment(int hours, int dayOfWeek){
        return teachingHourlyRate(dayOfWeek)*hours;
    }
}
```

- ¿Qué se está diciendo ?

  Reversa

# Agenda

# UML




- ¿Qué se está diciendo?

# Manual de referencia. Java.

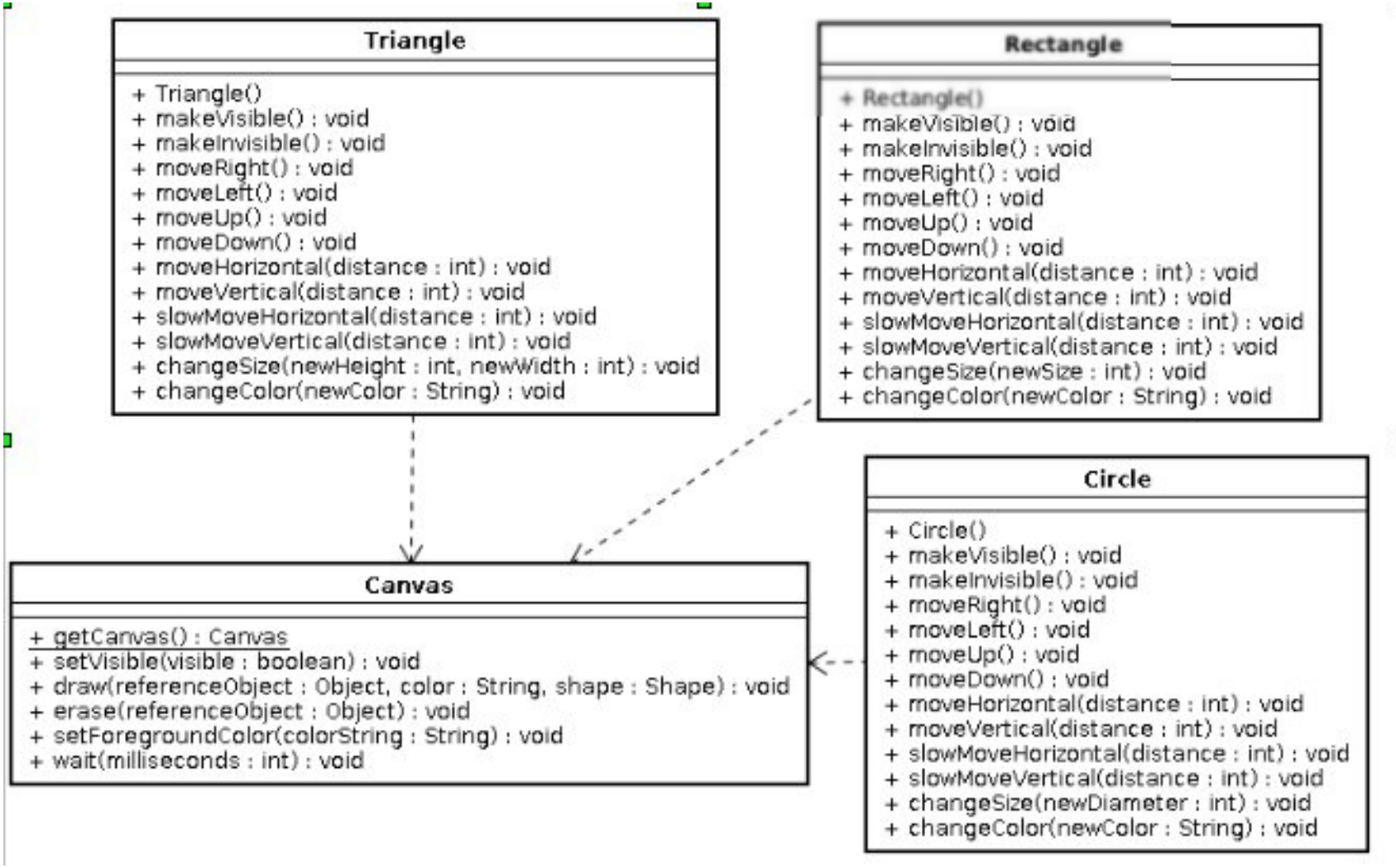
```
public interface Colorable {
        void setColor(byte r, byte g, byte b);
}
class Point { int x, y; }
class ColoredPoint extends Point implements Colorable {
        byte r, g, b;
        public void setColor(byte rv, byte gv, byte bv) {
                r = rv; g = gv; b = bv;
        }
}
class Test {
        public static void main(String[] args) {
                Point p = new Point();
                ColoredPoint cp = new ColoredPoint();
                p = cp;
                Colorable c = cp;
        }
}
```

- ¿Qué se está diciendo?

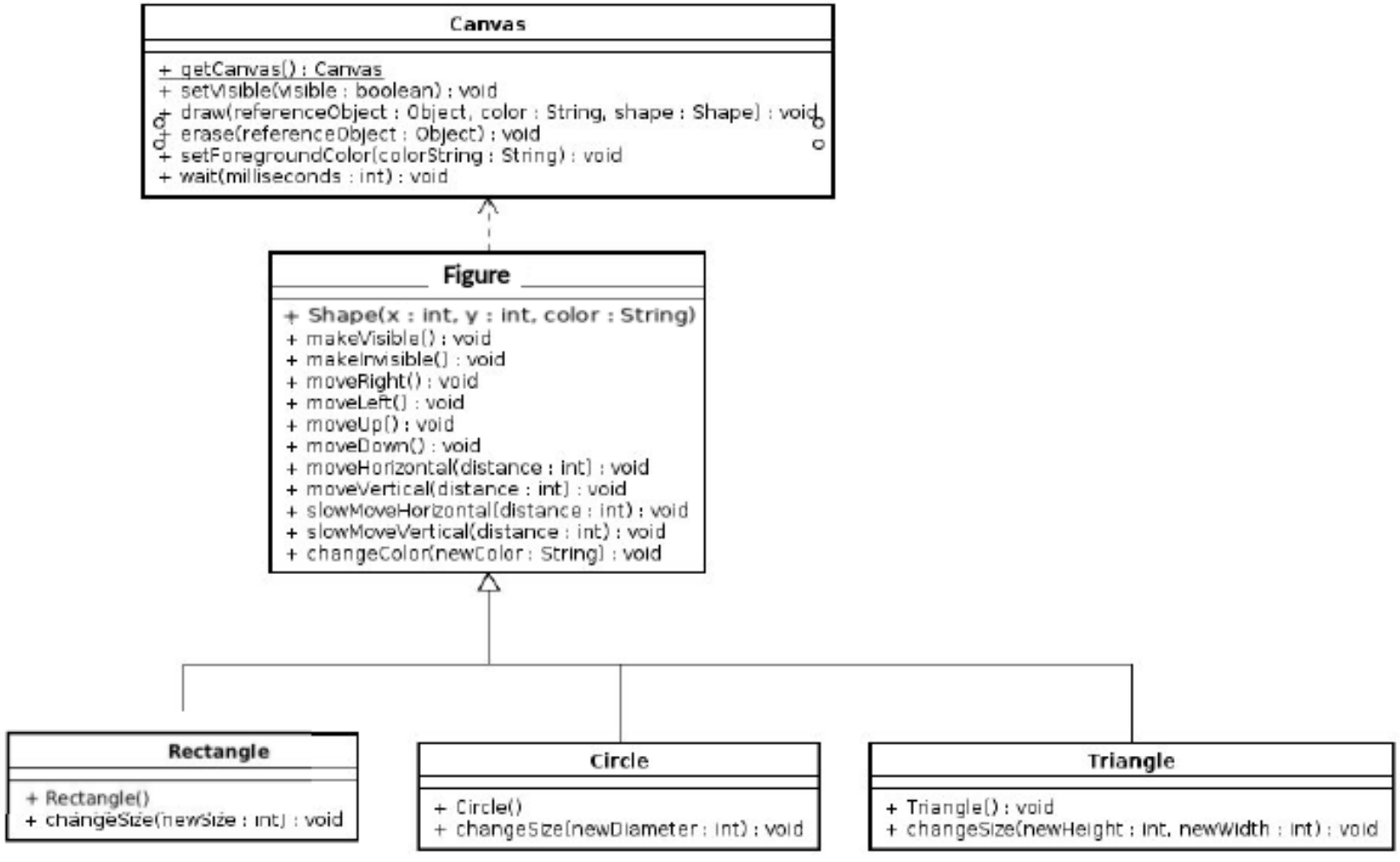
  Reversa

# Agenda

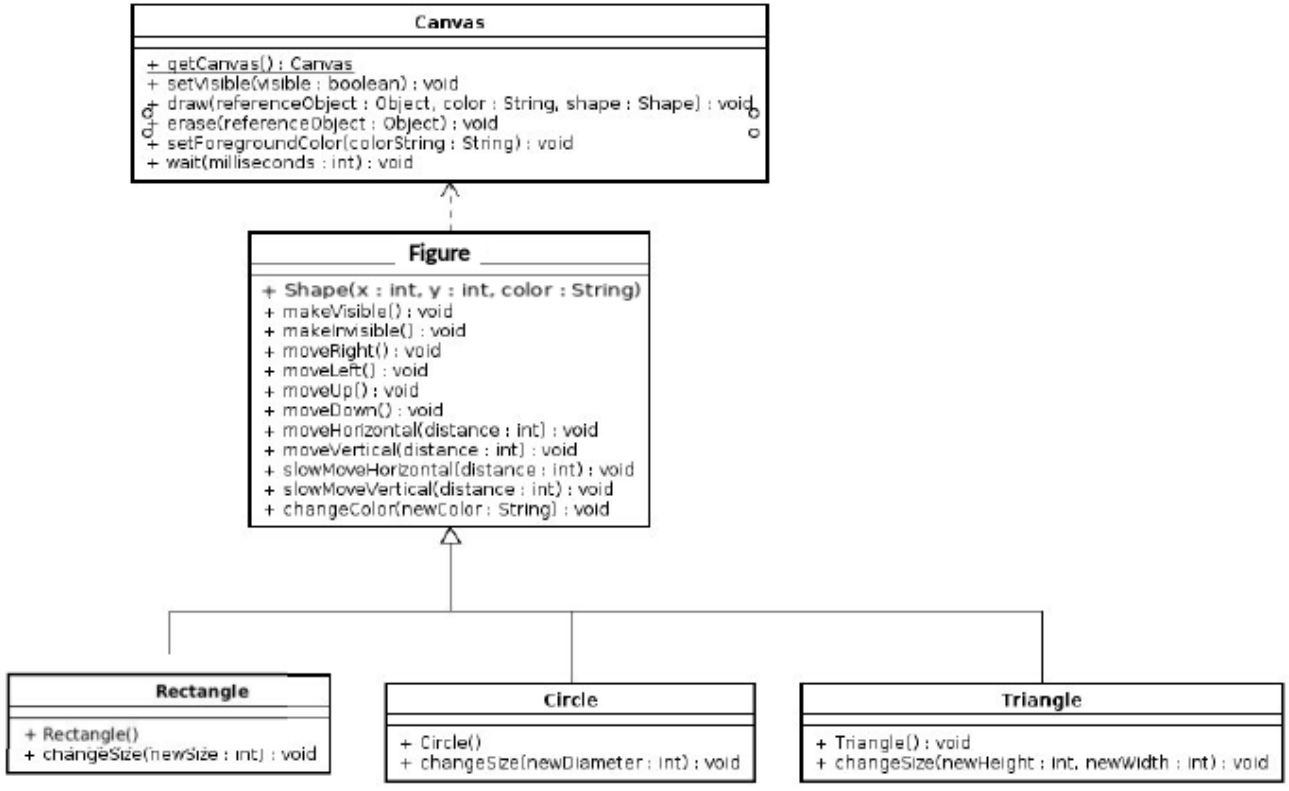
# Shapes


Triangle
+ Triangle()
+ makeVisible() : void
+ makeInvisible() : void
+ moveRight() : void
+ moveLeft() : void
+ moveUp() : void
+ moveDown() : void
+ moveHorizontal(distance : int) : void
+ moveVertical(distance : int) : void
+ slowMoveHorizontal(distance : int) : void
+ slowMoveVertical(distance : int) : void
+ changeSize(newHeight : int, newWidth : int) : void
+ changeColor(newColor : String) : void
Rectangle
+ Rectangle()
+ makeVisible() : void
+ makeInvisible() : void
+ moveRight() : void
+ moveLeft() : void
+ moveUp() : void
+ moveDown() : void
+ moveHorizontal(distance : int) : void
+ moveVertical(distance : int) : void
+ slowMoveHorizontal(distance : int) : void
+ slowMoveVertical(distance : int) : void
+ changeSize(newSize : int) : void
+ changeColor(newColor : String) : void
Canvas
+ getCanvas() : Canvas
+ setVisible(visible : boolean) : void
+ draw(referenceObject : Object, color : String, shape : Shape) : void
+ erase(referenceObject : Object) : void
+ setForegroundColor(colorString : String) : void
+ wait(milliseconds : int) : void
Circle
+ Circle()
+ makeVisible() : void
+ makeInvisible() : void
+ moveRight() : void
+ moveLeft() : void
+ moveUp() : void
+ moveDown() : void
+ moveHorizontal(distance : int) : void
+ moveVertical(distance : int) : void
+ slowMoveHorizontal(distance : int) : void
+ slowMoveVertical(distance : int) : void
+ changeSize(newDiameter : int) : void
+ changeColor(newColor : String) : void

# Shapes


Canvas
+ getCanvas() : Canvas
+ setVisible(visible : boolean) : void
+ draw(referenceObject : Object, color : String, shape : Shape) : void
+ erase(referenceObject : Object) : void
+ setForegroundColor(colorString : String) : void
+ wait(milliseconds : int) : void
Figure
+ Shape(x : int, y : int, color : String)
+ makeVisible() : void
+ makeInvisible() : void
+ moveRight() : void
+ moveLeft() : void
+ moveUp() : void
+ moveDown() : void
+ moveHorizontal(distance : int) : void
+ moveVertical(distance : int) : void
+ slowMoveHorizontal(distance : int) : void
+ slowMoveVertical(distance : int) : void
+ changeColor(newColor : String) : void
Rectangle
+ Rectangle()
+ changeSize(newSize : int) : void
Circle
+ Circle()
+ changeSize(newDiameter : int) : void
Triangle
+ Triangle() : void
+ changeSize(newHeight : int, newWidth : int) : void

# Shapes

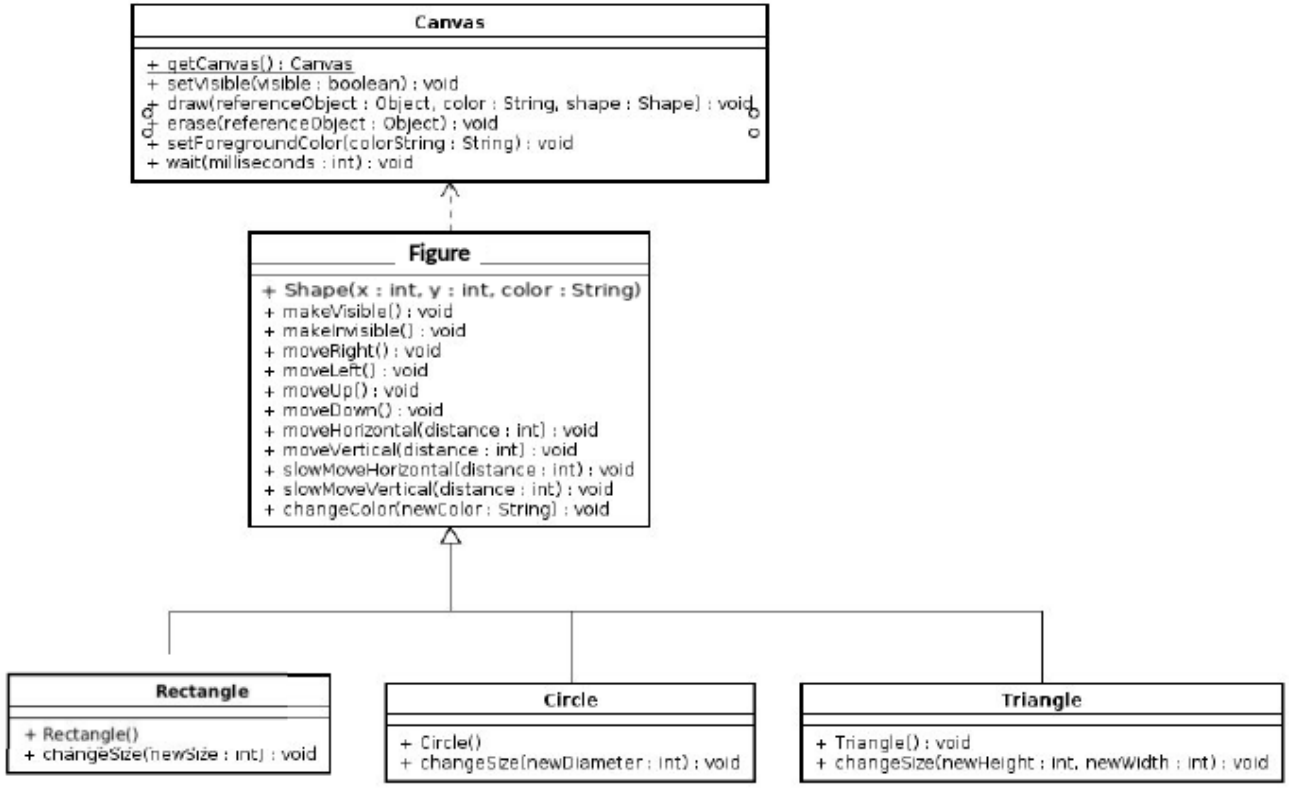



## Puliendo

¿Cómo ..

- podemos impedir que se creen figuras sin sentido?

# Shapes

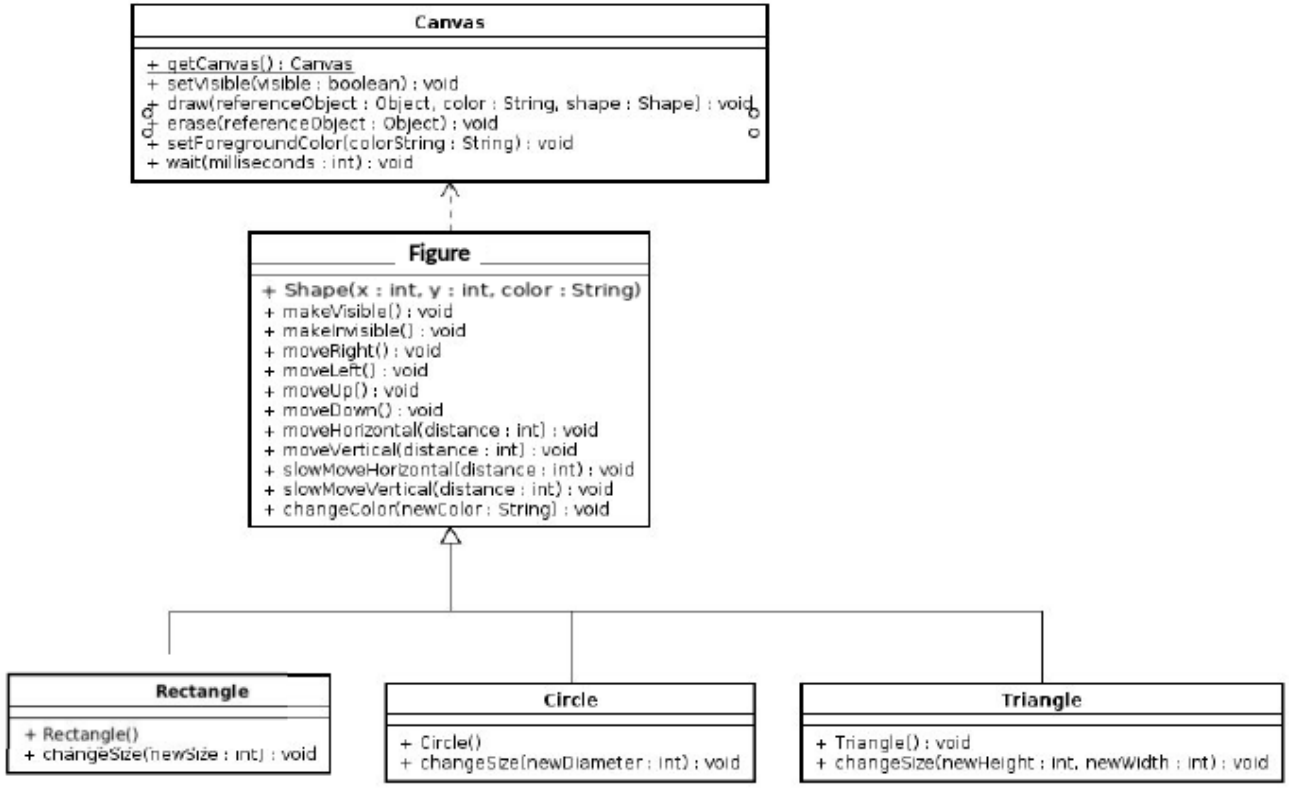



## Puliendo

¿Cómo ..

- podemos impedir que se creen subclases de `Circle`?

# Shapes

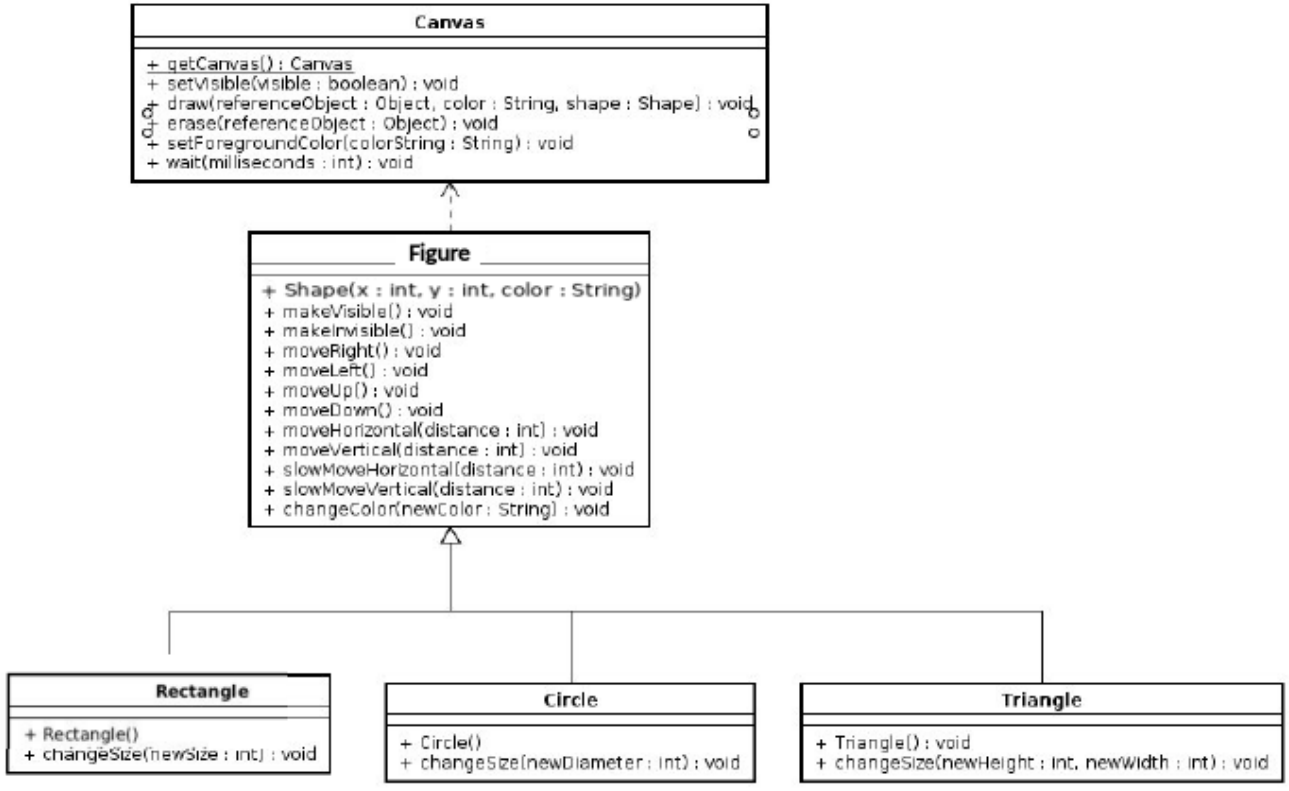



## Puliendo

¿Cómo ..

- podemos exigir que todas las figuras hagan zoom?

# Shapes

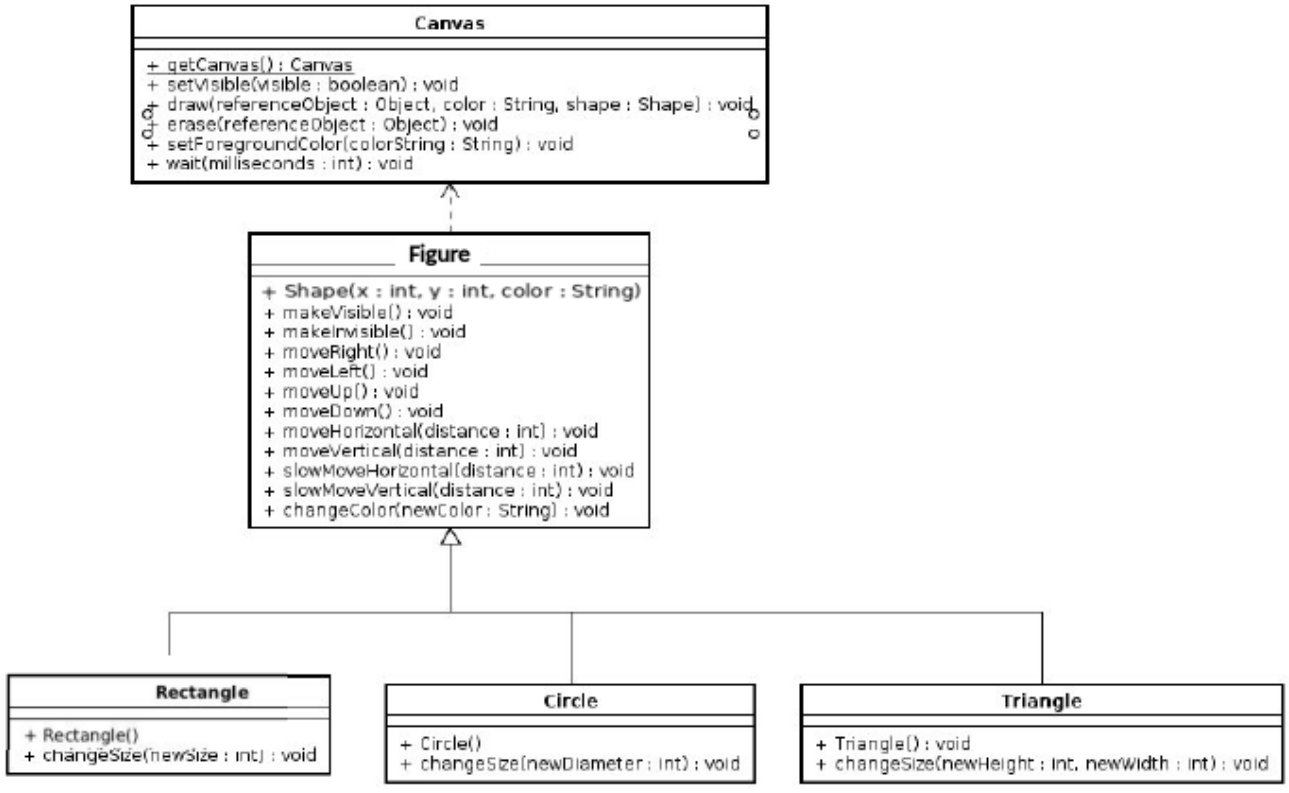



## Puliendo

¿Cómo ..

- podemos impedir que se cambie la forma en que las figuras cambian de color?

# Shapes




## Puliendo

¿Cómo ..

- podemos exigir que algunas figuras (por ahora rectángulo y triangulo) retornen cuadrados equivalentes?

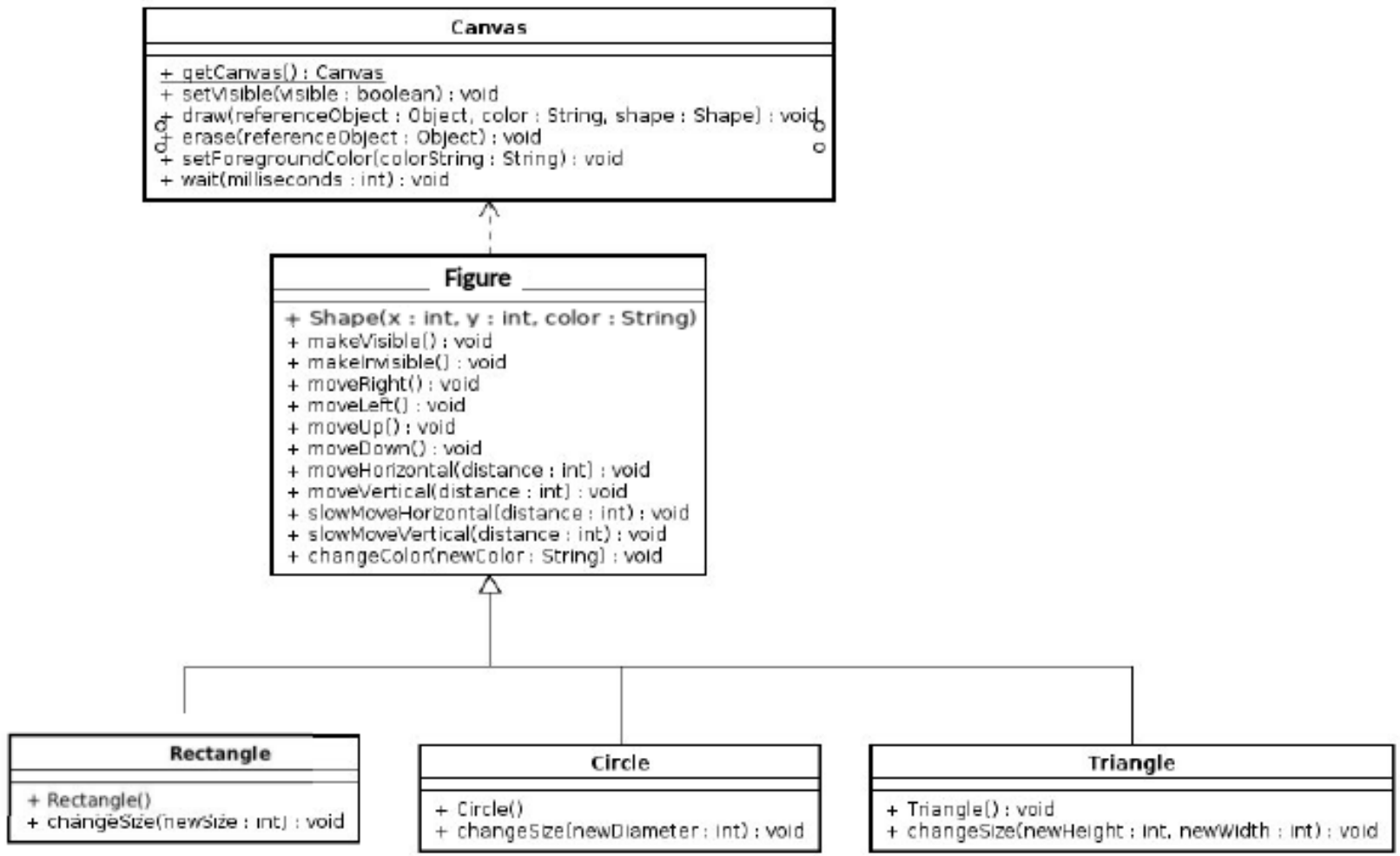
`Squarable - quadrature`

# Agenda

# Shapes

## Nueva figura

- Línea



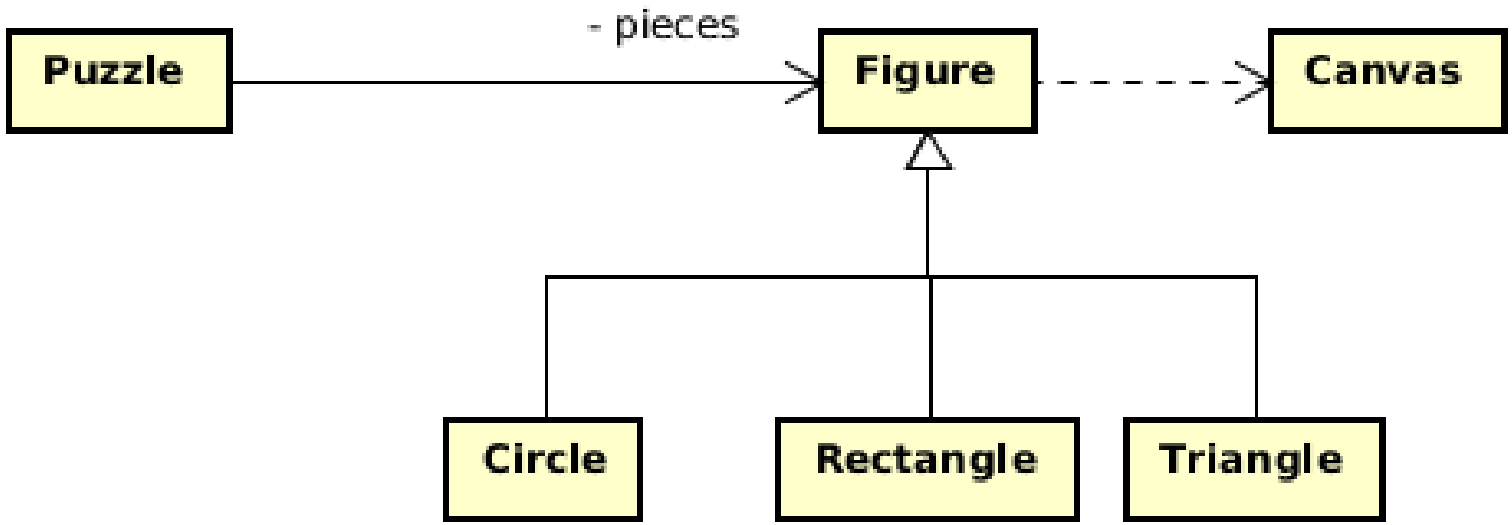
# Agenda

# Agenda

## Nueva clase

- makeVisible. Hacerlo visible
- area. Calcular el área

## Puzzle

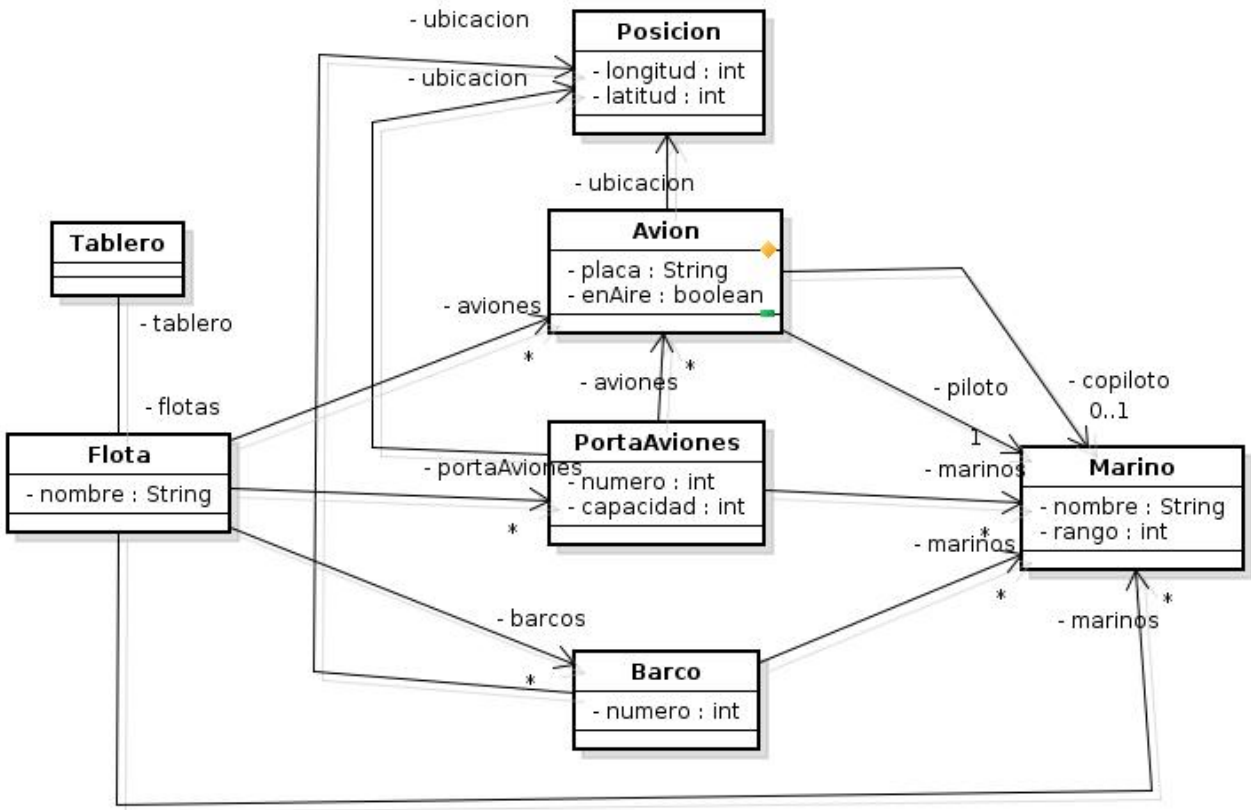


# Agenda

# Batalla naval

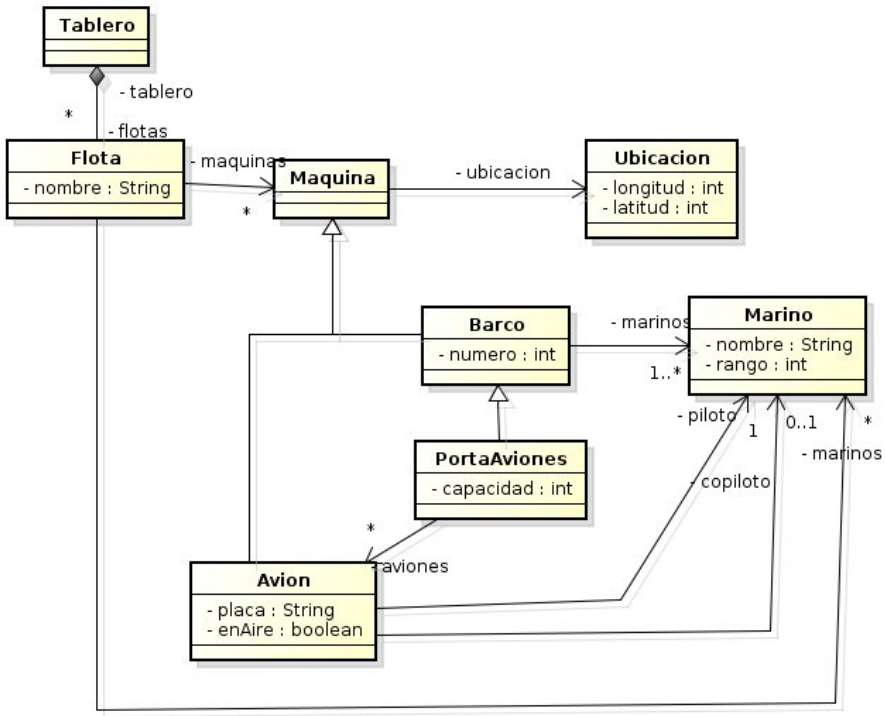
## Mejor estructura

- Aprovechando la herencia



# Batalla naval

## Batalla naval

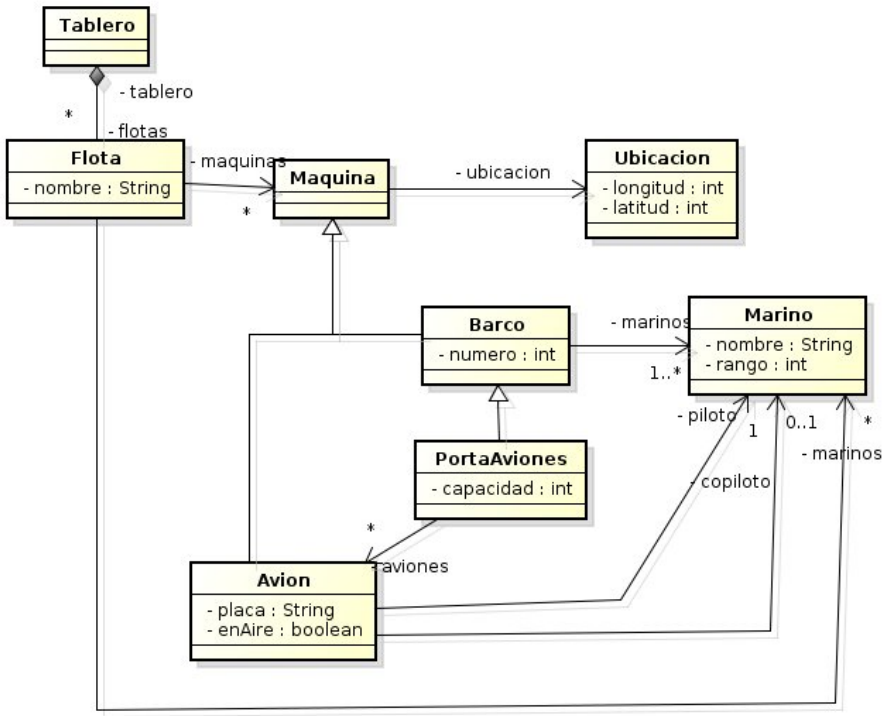



## Refactorizando

- ¿Quién debe ser abstracta?

# Batalla naval

## Batalla naval



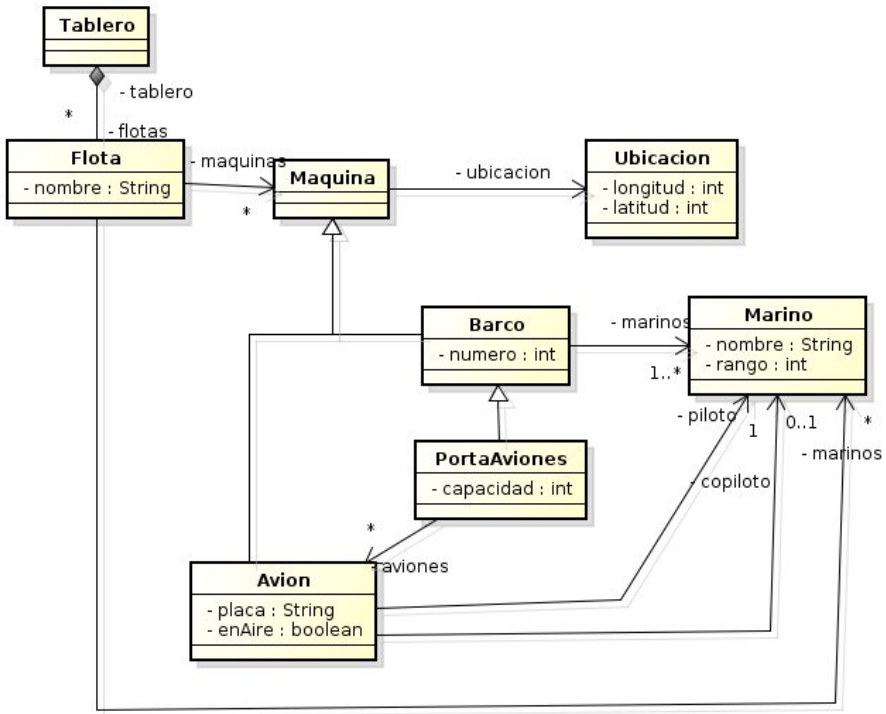

## Refactorizando

- ¿Quién debe ser abstracta?
- ¿Quién podría ser final?

# Agenda

# Batalla naval

## Batalla naval

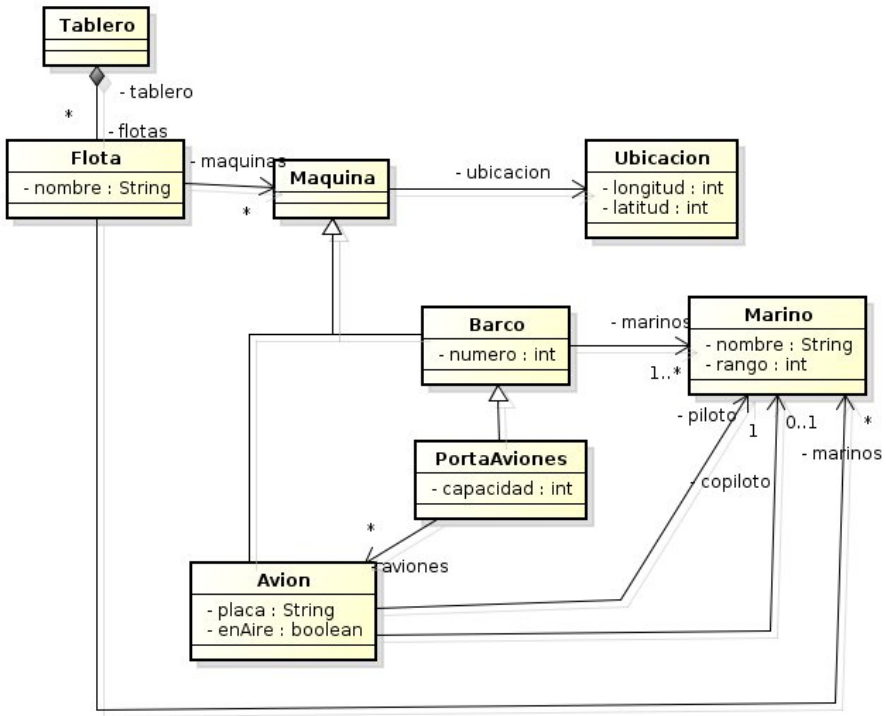



## Dos métodos

- Incorporar avance

# Batalla naval

## Batalla naval




## Dos métodos

- Incorporar avance
- Incorporar maquinasDebiles