Prácticas DS resueltas

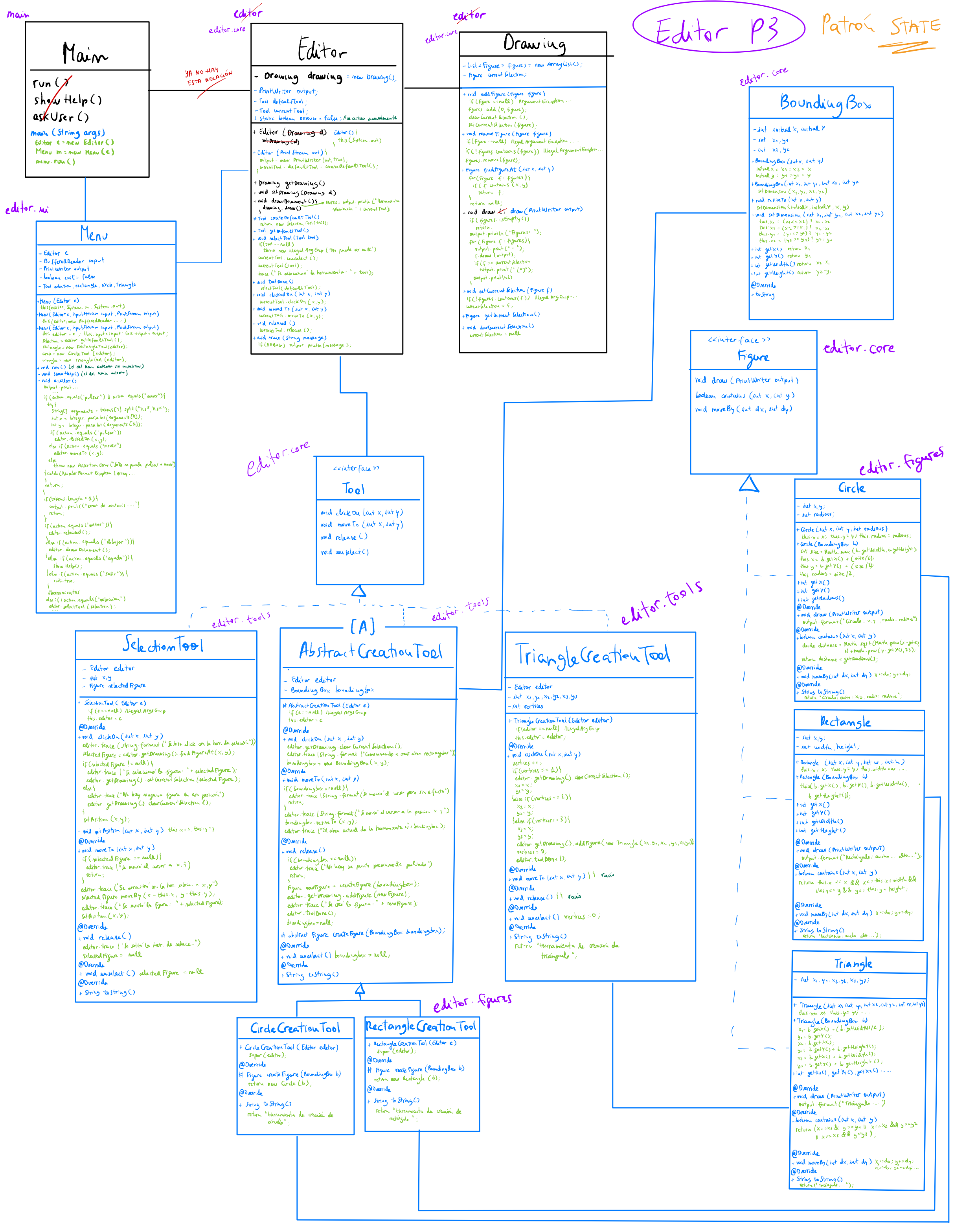
Eduardo Blanco Bielsa

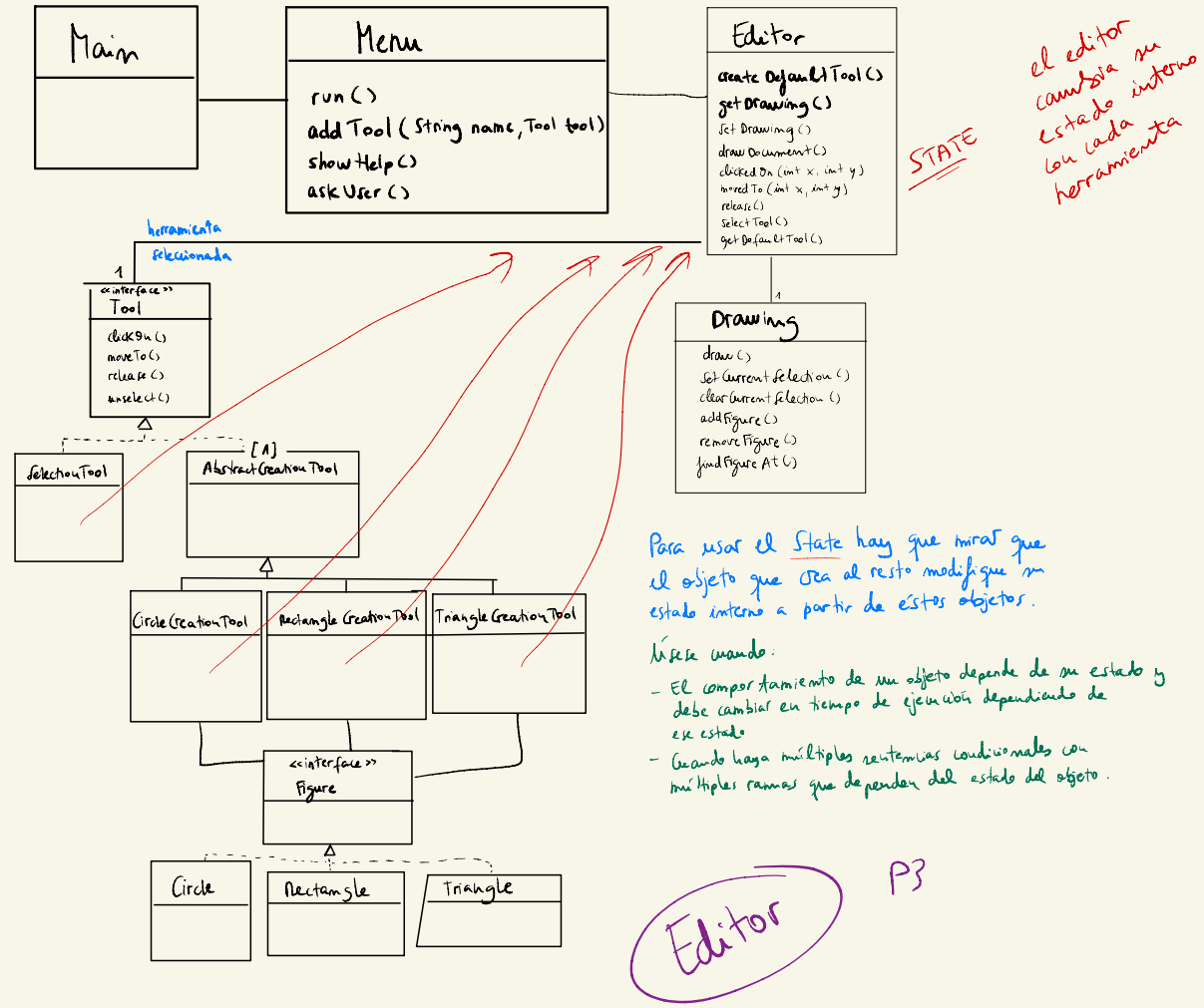
U0285176

Leyenda prácticas

Programa original
Código que ya no existe
Nuevo código
Implementación del nuevo código
Paquete contenedor
Patrón/es usado/s
Comentarios sobre el código

Nota: los que están mejor son los que siguen este código de colores. Los demás son más generales


Editor P3
Patrón STATE
main
Main
run ( )
show Help ( )
askUser ( )
main (String args)
Editor e = new Editor ( )
Menu m = new Menu (e)
menu.run ( )
YA NO HAY ESTA RELACIÓN
editor.ui
Menu
editor
editor.core
Editor
- Drawing drawing = new Drawing();
Drawing
editor.core
BoundingBox
editor.core
<<interface>>
Figure
void draw (PrintWriter output)
boolean contains (int x, int y)
void moveBy (int dx, int dy)
editor.core
<<interface>>
Tool
void clickOn (int x, int y)
void moveTo (int x, int y)
void release ( )
void unselect ( )
editor.figures
Circle
Rectangle
Triangle
editor.tools
SelectionTool
[A]
AbstractCreationTool
editor.tools
TriangleCreationTool
editor.figures
CircleCreationTool
RectangleCreationTool



Para usar el State hay que mirar que el objeto que crea al resto modifique su estado interno a partir de éstos objetos.

Úsese cuando:

- El comportamiento de un objeto depende de su estado y debe cambiar en tiempo de ejecución dependiendo de ese estado
- Cuando haya múltiples sentencias condicionales con múltiples ramas que dependan del estado del objeto.

Editor

P3

P1 videoclub

Strategy

todo en el mismo paquete

Customer

- String name
- List<Rental> rentals = new ArrayList<>();

+ Customer (String name)
+ void addRental (Rental rental)
+ String getName()
+ String statement()
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  String result = "Rental Record for " + getName() + "\n";
  for (Rental each : rentals){
    double thisAmount = 0;
    switch (each.getMovie().getPriceCode()){
      case Movie.REGULAR:
      ...
      case Movie.NEW_RELEASE:
      ...
      case Movie.CHILDRENS:
    }
    frequentRenterPoints++;
    if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
      each.getDaysRented() > 1)
      frequentRenterPoints ++;
    result += "\t" + each.getMovie().getTitle() + "\t" + String.valueOf(each.getCharge()) + "\n";
    thisAmount += this.Amount
    totalAmount += thisAmount
  }
  result += "Amount owed " + totalAmount + getTotalAmount()
  result += "You earned " + frequentRenterPoints + ... getTotalFrequentRenterPoints()
  return result;
- double getTotalAmount()
  double result = 0;
  for (Rental each : rentals)
    result += each.getCharge();
  return result;
- int getTotalFrequentRenterPoints()
  int result = 0;
  for (Rental each : rentals)
    result += each.getFrequentRenterPoints();
  return result;

esto está bien
NUEVO
result += "\t" + each.getMovie().getTitle() + "\t" + String.valueOf(each.getCharge()) + "\n";

0...*

Rental

- Movie movie;
- int daysRented;

+ Rental (Movie m, int days)
+ int getDaysRented()
+ Movie getMovie()
+ double getCharge() return movie.getCharge(daysRented);
+ int getFrequentRenterPoints() return movie.getFrequentRenterPoints(daysRented);

Movie

+ static final ~~int~~ MovieType CHILDRENS = 2; new ChildrensMovie()
+ static final ~~int~~ MovieType NEW_RELEASE = 1; new NewReleaseMovie()
+ static final ~~int~~ MovieType REGULAR = 0; new RegularMovie()
- String title;
- ~~int priceCode;~~ - MovieType movieType;

+ Movie (String title, ~~int priceCode~~ MovieType type)
+ ~~int getPriceCode()~~ + double getCharge(int daysRented) return movieType.getCharge(daysRented)
+ ~~void setPriceCode()~~ + int getFrequentRenterPoints(int daysRented) return movieType.getFrequentRenterPoints(daysRented)
+ String getTitle()

<<interface>> MovieType

double getCharge(int daysRented);
int getFrequentRenterPoints(int daysRented);

ChildrenMovie

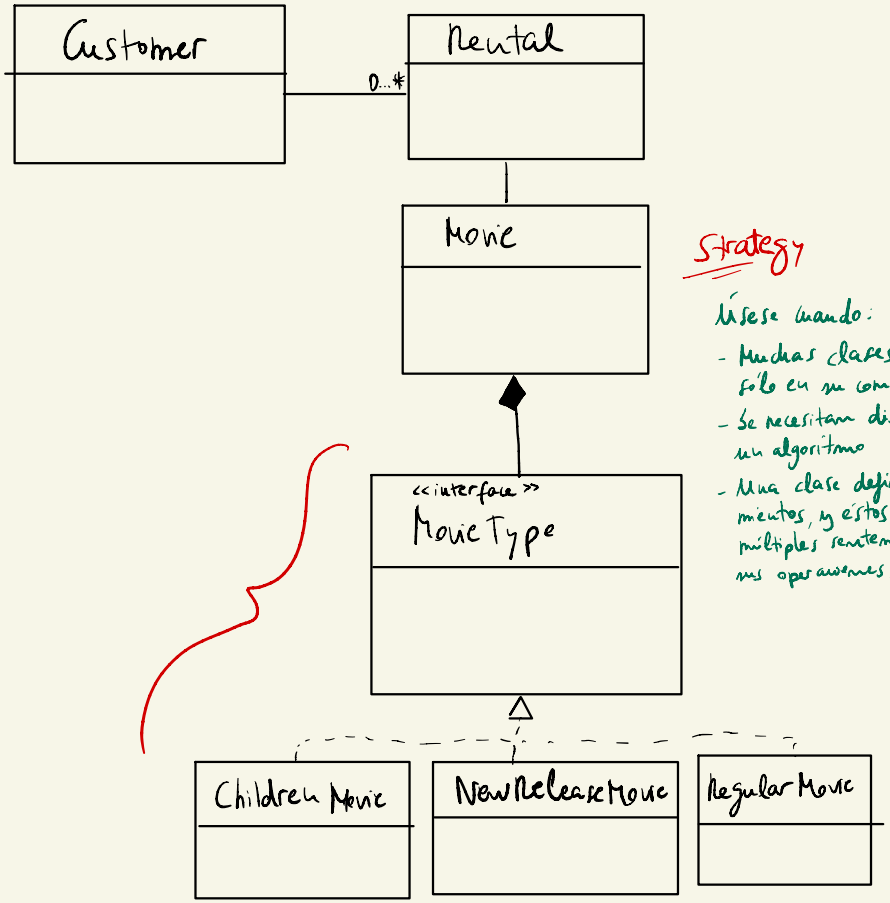
@Override
+ double getCharge(int daysRented)
  double result = 1,5;
  if(daysRented > 3)
    result += (daysRented - 3) * 1.5;
  return result;
@Override
+ int getFrequentRenterPoints(int days)
  return 1;

RegularMovie

@Override
+ double getCharge(int daysRented)
  double result = 2;
  if(daysRented > 2)
    result += (daysRented - 2) * 1.5;
  return result;
@Override
+ int getFrequentRenterPoints(int days)
  return 1;

NewReleaseMovie

@Override
+ double getCharge(int daysRented)
  return daysRented * 3;
@Override
+ int getFrequentRenterPoints(int days)
  if (daysRented > 1)
    return 2;
  return 1



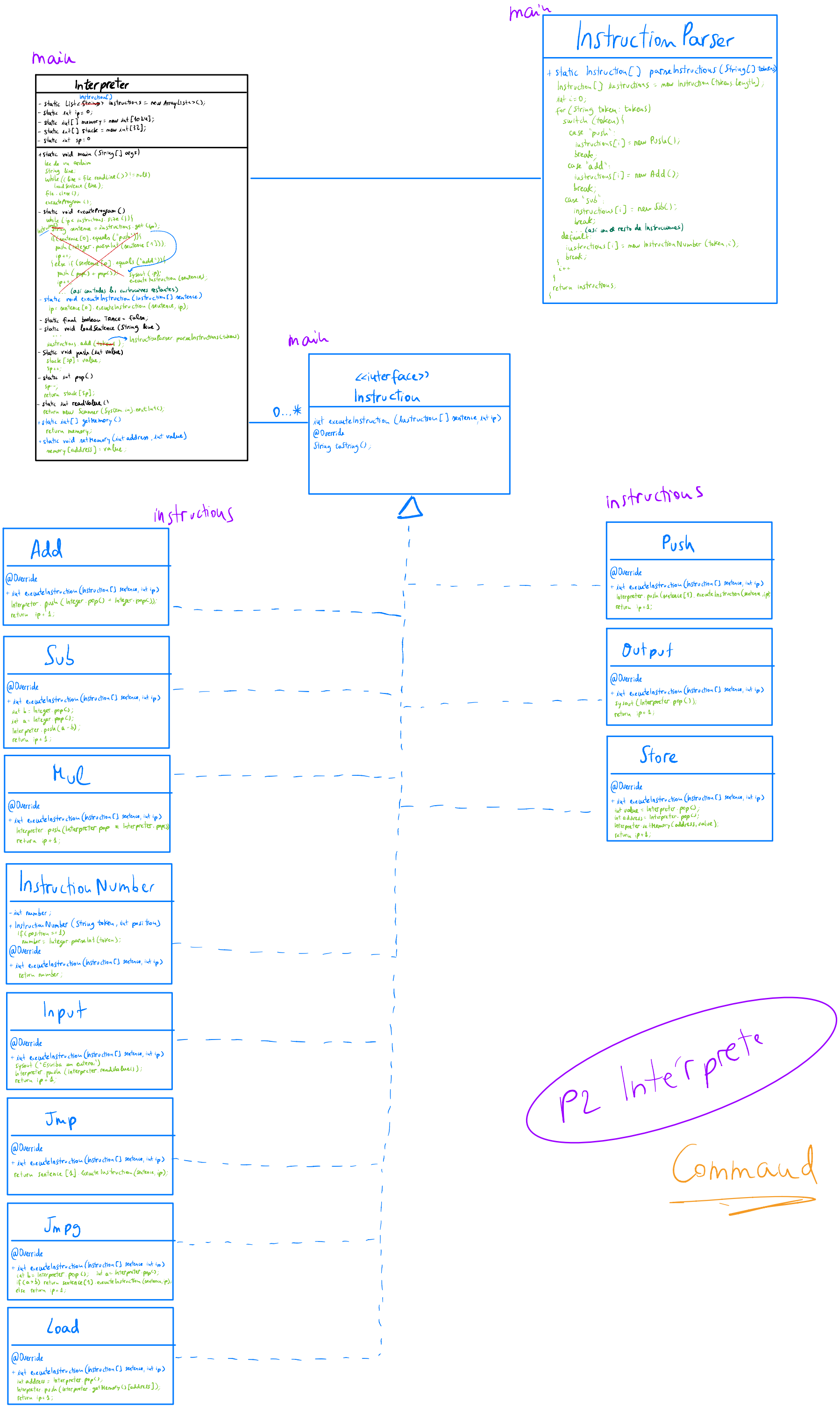
Strategy

Úsese cuando:

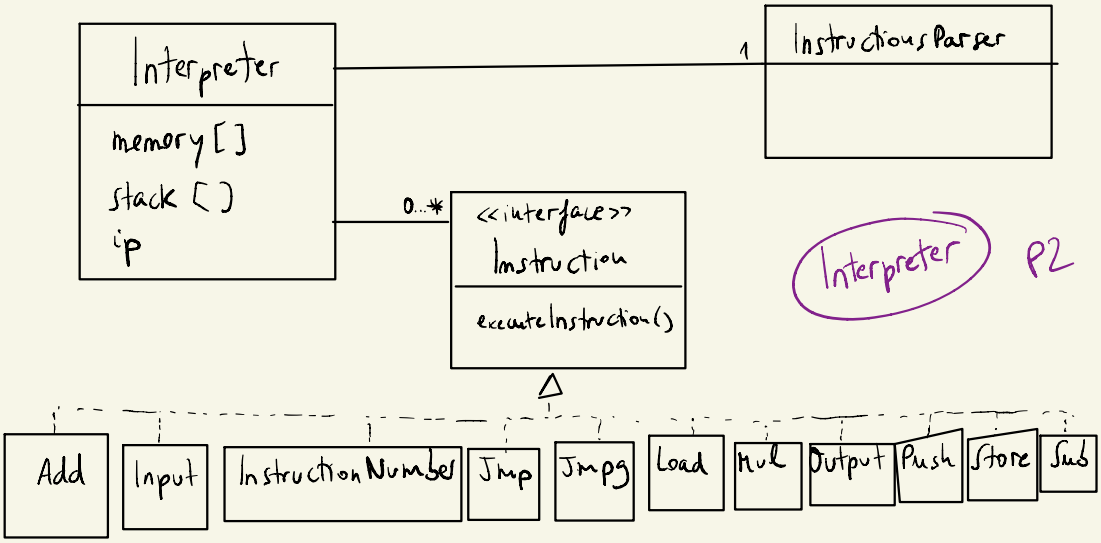
- Muchas clases relacionadas difieren sólo en su comportamiento
- Se necesitan distintas variantes de un algoritmo
- Una clase define muchos comportamientos, y éstos se representan con múltiples sentencias condicionales en sus operaciones.

Video club

P1



main
Interpreter
main
Instruction Parser
+ static Instruction[] parseInstructions (String[] tokens)
main
<<interface>>
Instruction
int executeInstruction (Instruction[] sentence, int ip)
@Override
String toString();
0...*
instructions
Add
Sub
Mul
Instruction Number
Input
Jmp
Jmpg
Load
instructions
Push
Output
Store
@Override
P2 Interprete
Command

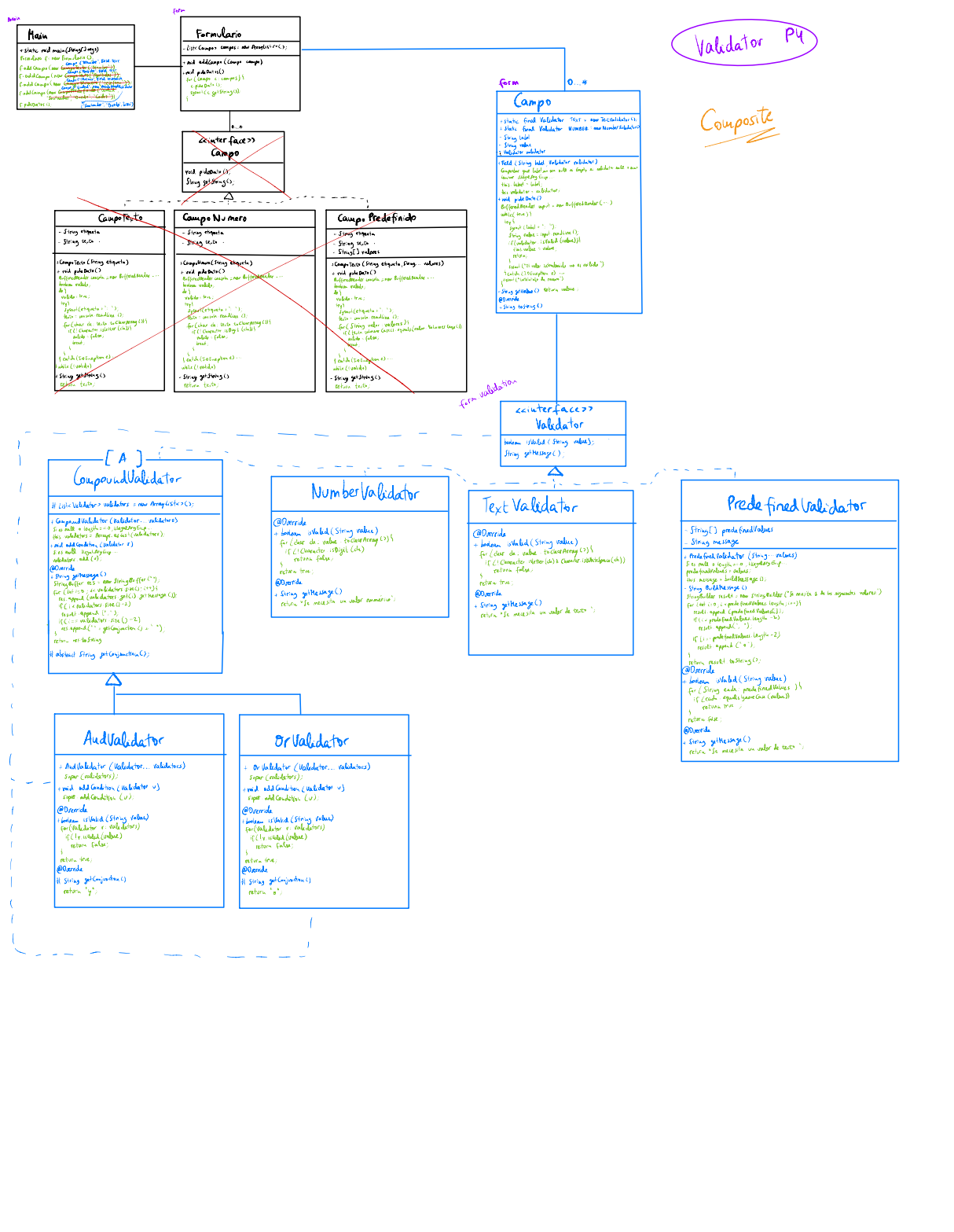


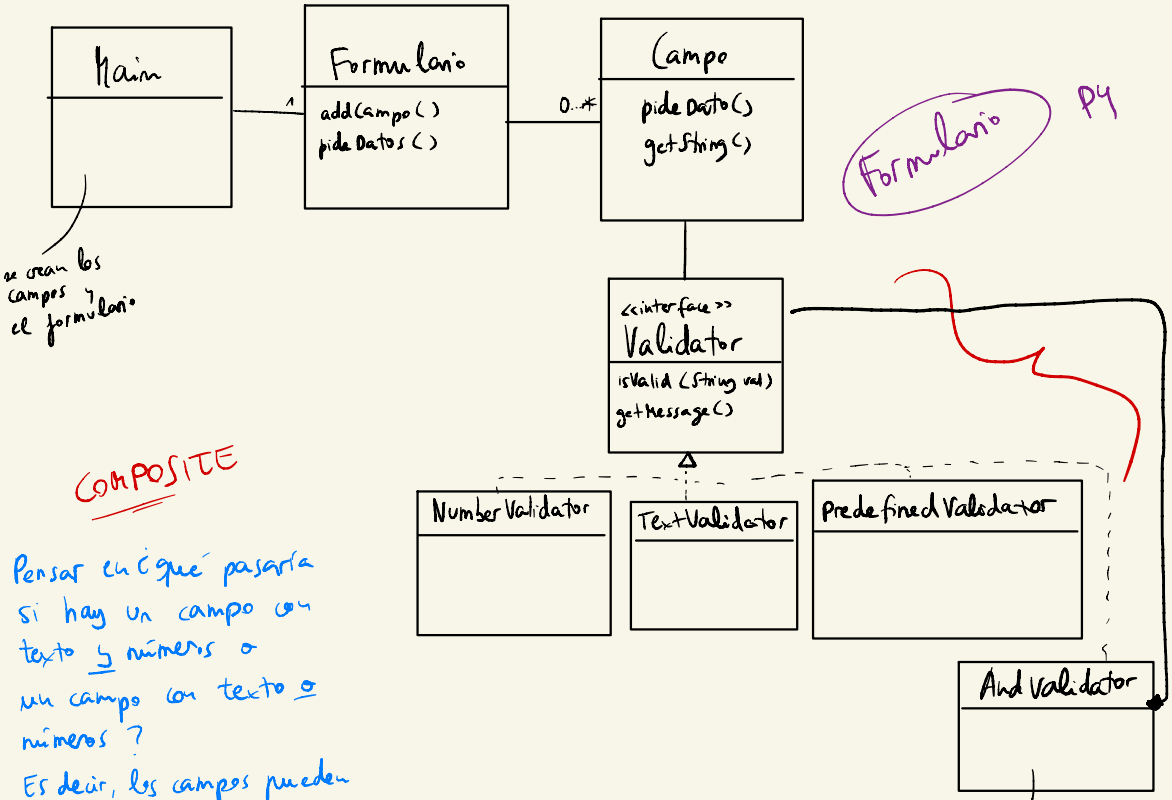
Interpreter P2

Command

Úsese cuando se quiera:

- Parametrizar objetos con una acción a realizar.
- Especificar, poner en cola y ejecutar peticiones en diferentes instantes de tiempo
- Permitir deshacer
- Permitir registrar los cambios para posibles caídas del sistema.

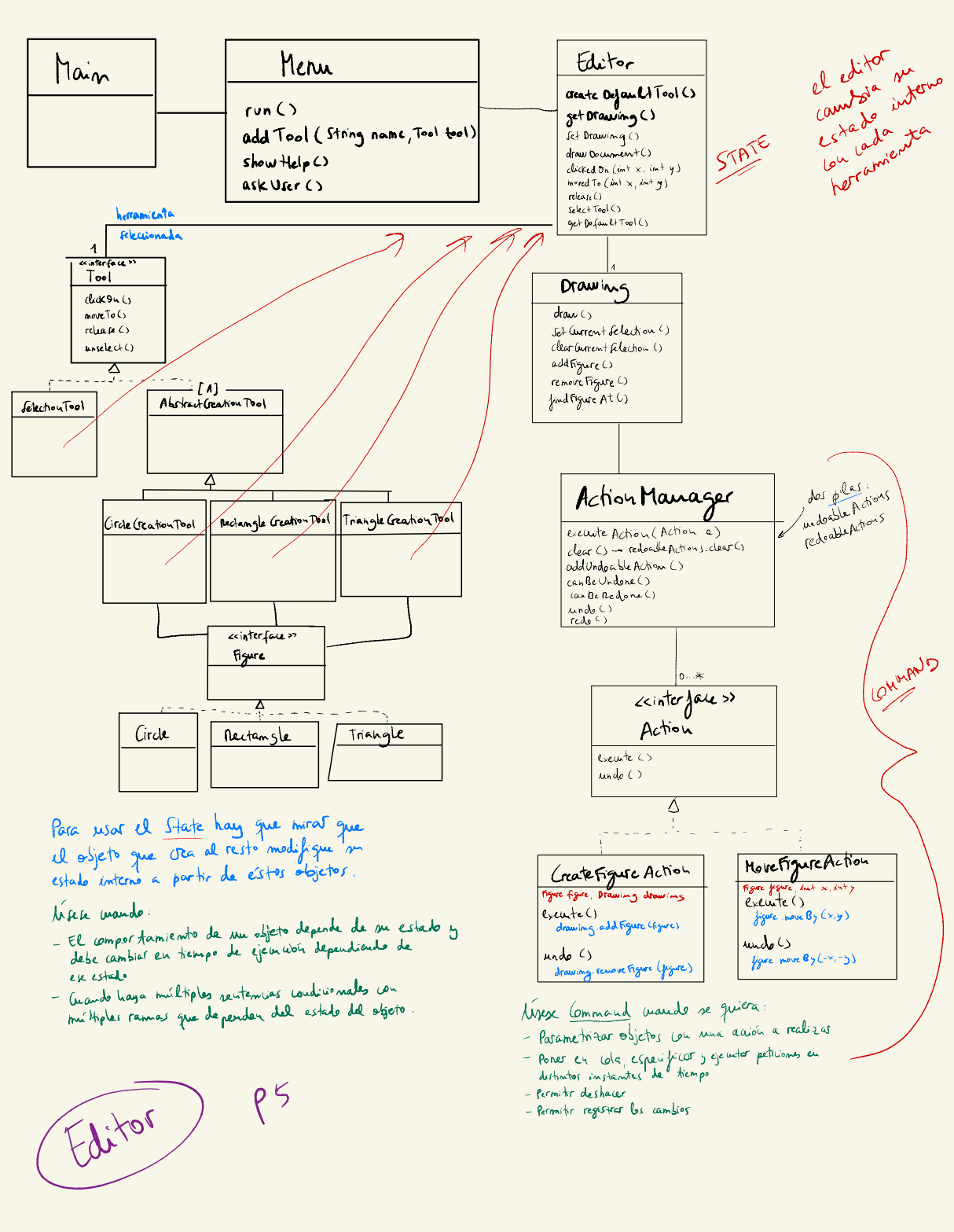

Validator PY
Composite
Main
Formulario
Campo
0...*
<<interface>>
Campo
CampoTexto
CampoNumero
CampoPredefinido
form validation
<<interface>>
Validator
[A]
CompoundValidator
NumberValidator
TextValidator
PredefinedValidator
AndValidator
OrValidator



COMPOSITE

Pensar en qué pasaría si hay un campo con texto y números o un campo con texto o números?
Es decir, los campos pueden componerse de varias validaciones.

Úsese cuando:
- Se quiera representar jerarquías de parte-todo
- Se quiera que los clientes puedan diferenciar entre composiciones de objetos y objetos individuales.



Para usar el State hay que mirar que el objeto que crea al resto modifique su estado interno a partir de estos objetos.

Usaría cuando:

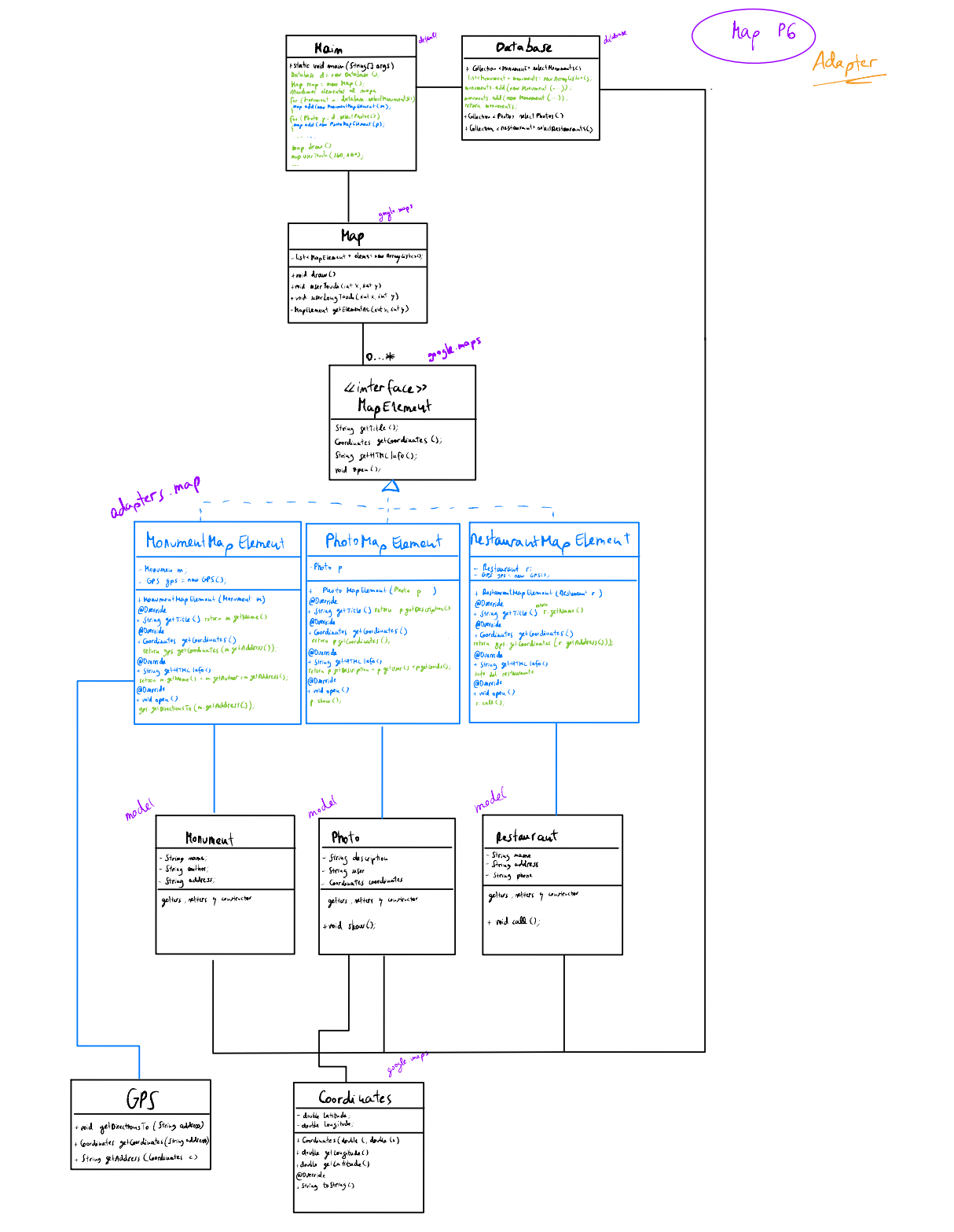
- El comportamiento de un objeto depende de su estado y debe cambiar en tiempo de ejecución dependiendo de ese estado
- Cuando haya múltiples sentencias condicionales con múltiples ramas que dependan del estado del objeto.

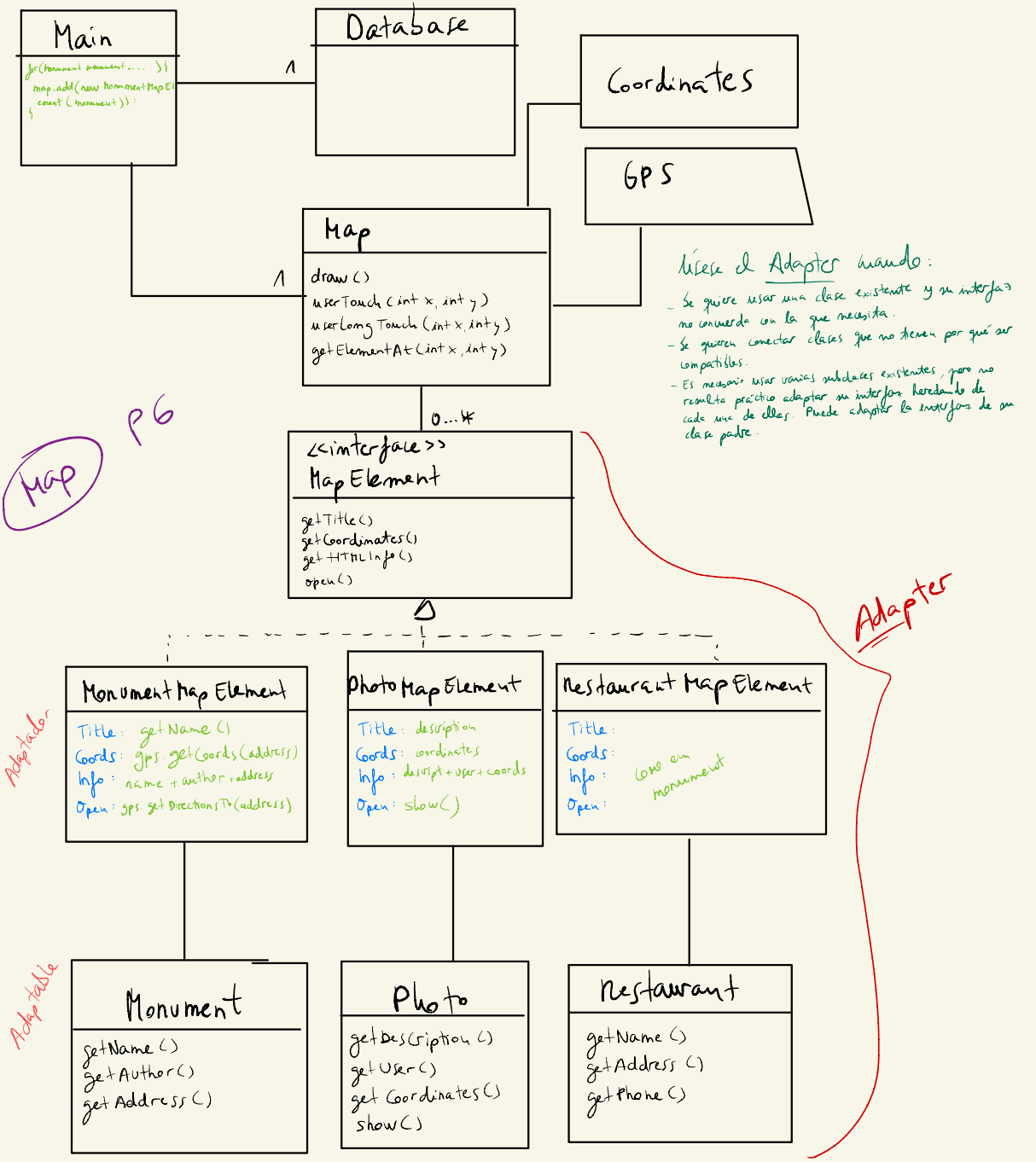
Usaría Command cuando se quiera:

- Parametrizar objetos con una acción a realizar
- Poner en cola, especificar y ejecutar peticiones en distintos instantes de tiempo
- Permitir deshacer
- Permitir registrar los cambios

Editor

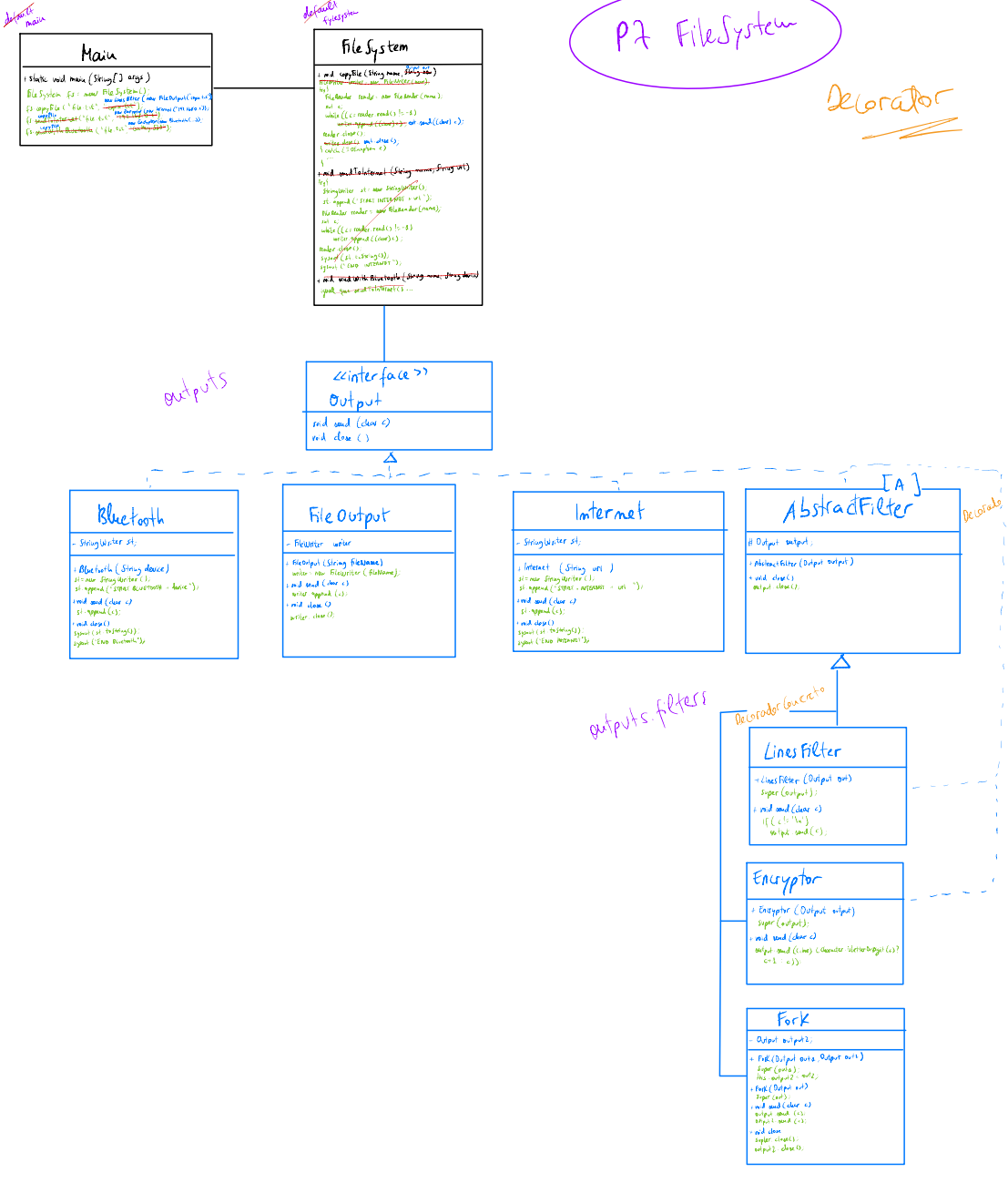
P5

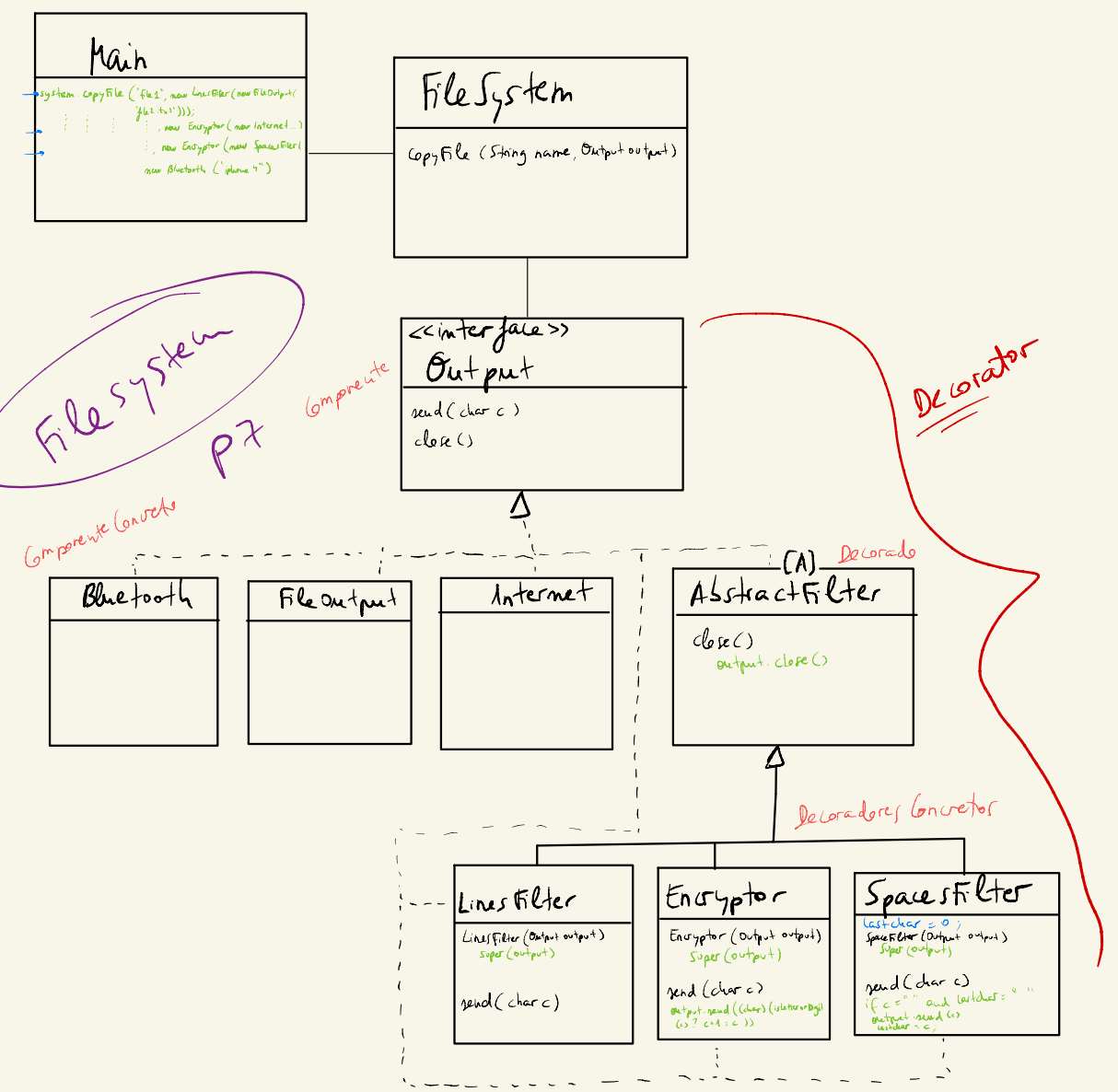

Map P6
Adapter
Main
+ static void main (String[] args)
Database db = new Database ();
Map map = new Map ();
Monument element; or maps
for (Monument m : database.selectMonuments())
map.add(new MonumentMapElement(m));
for (Photo p : db.selectPhotos())
map.add(new PhotoMapElement(p));
map.draw();
map.userInside(150, 400);
default
Database
+ Collection <Monument> selectMonuments();
ListMonument> monuments = new ArrayList<>();
monuments.add(new Monument(...));
monuments.add(new Monument(...));
return monuments;
+ Collection < Photo> selectPhotos()
+ Collection <Restaurant> selectRestaurants()
database
google maps
Map
- List< MapElement > elems = new ArrayList<>();
+ void draw()
+ void userInside(int x, int y)
+ void userLongTouch(int x, int y)
- MapElement getElementAt(int x, int y)
0...*
google maps
«interface»
MapElement
String getTitle();
Coordinates getCoordinates();
String getHTMLInfo();
void open();
adapters . map
MonumentMap Element
- Monument m;
- GPS gps = new GPS();
+ MonumentMapElement (Monument m)
@Override
+ String getTitle() return m.getName()
@Override
+ Coordinates getCoordinates()
return gps.getCoordinates(m.getAddress());
@Override
+ String getHTMLInfo()
return m.getName() + m.getAuthor() + m.getAddress();
@Override
+ void open()
gps.getDirectionsTo(m.getAddress());
PhotoMap Element
- Photo p
+ PhotoMapElement (Photo p)
@Override
+ String getTitle() return p.getDescription()
@Override
+ Coordinates getCoordinates()
return p.getCoordinates();
@Override
+ String getHTMLInfo()
return p.getDescription() + p.getUser() + p.getCoords();
@Override
+ void open()
p.show();
RestaurantMap Element
- Restaurant r;
- GPS gps = new GPS();
+ RestaurantMapElement (Restaurant r)
@Override
+ String getTitle() return r.getName()
@Override
+ Coordinates getCoordinates()
return gps.getCoordinates (r.getAddress());
@Override
+ String getHTMLInfo()
info del restaurante
@Override
+ void open()
r.call();
model
Monument
- String name;
- String author;
- String address;
getters , setters y constructor
model
Photo
- String description
- String user
- Coordinates coordinates
getters , setters y constructor
+ void show();
model
Restaurant
- String name
- String address
- String phone
getters , setters y constructor
+ void call();
google maps
GPS
+ void getDirectionsTo (String address)
+ Coordinates getCoordinates (String address)
- String getAddress (Coordinates c)
Coordinates
- double latitude
- double longitude
+ Coordinates (double l, double l)
+ double getLongitude()
+ double getLatitude()
@Override
+ String toString()


Main
for(Monument monument....){
map.add(new MonumentMapEl
ement(monument));
}
1
Database
Coordinates
GPS
Map
1
draw()
userTouch(int x, int y)
userLongTouch(int x, int y)
getElementAt(int x, int y)
Usese el Adapter cuando:
- Se quiere usar una clase existente y su interfaz no concuerda con la que necesita.
- Se quieren conectar clases que no tienen por qué ser compatibles.
- Es necesario usar varias subclases existentes, pero no resulta práctico adaptar su interfaz heredando de cada una de ellas. Puede adaptar la interfaz de su clase padre.
P6
Map
0...*
<<interface>>
MapElement
getTitle()
getCoordinates()
getHTMLInfo()
open()
Adapter
Adaptador
MonumentMapElement
Title: getName()
Coords: gps.getCoords(address)
Info: name + author + address
Open: gps.getDirectionsTo(address)
PhotoMapElement
Title: description
Coords: coordinates
Info: description + user + coords
Open: show()
RestaurantMapElement
Title:
Coords:
Info:
Open:
como en monument
Adaptado
Monument
getName()
getAuthor()
getAddress()
Photo
getDescription()
getUser()
getCoordinates()
show()
Restaurant
getName()
getAddress()
getPhone()

P7 FileSystem

Decorator

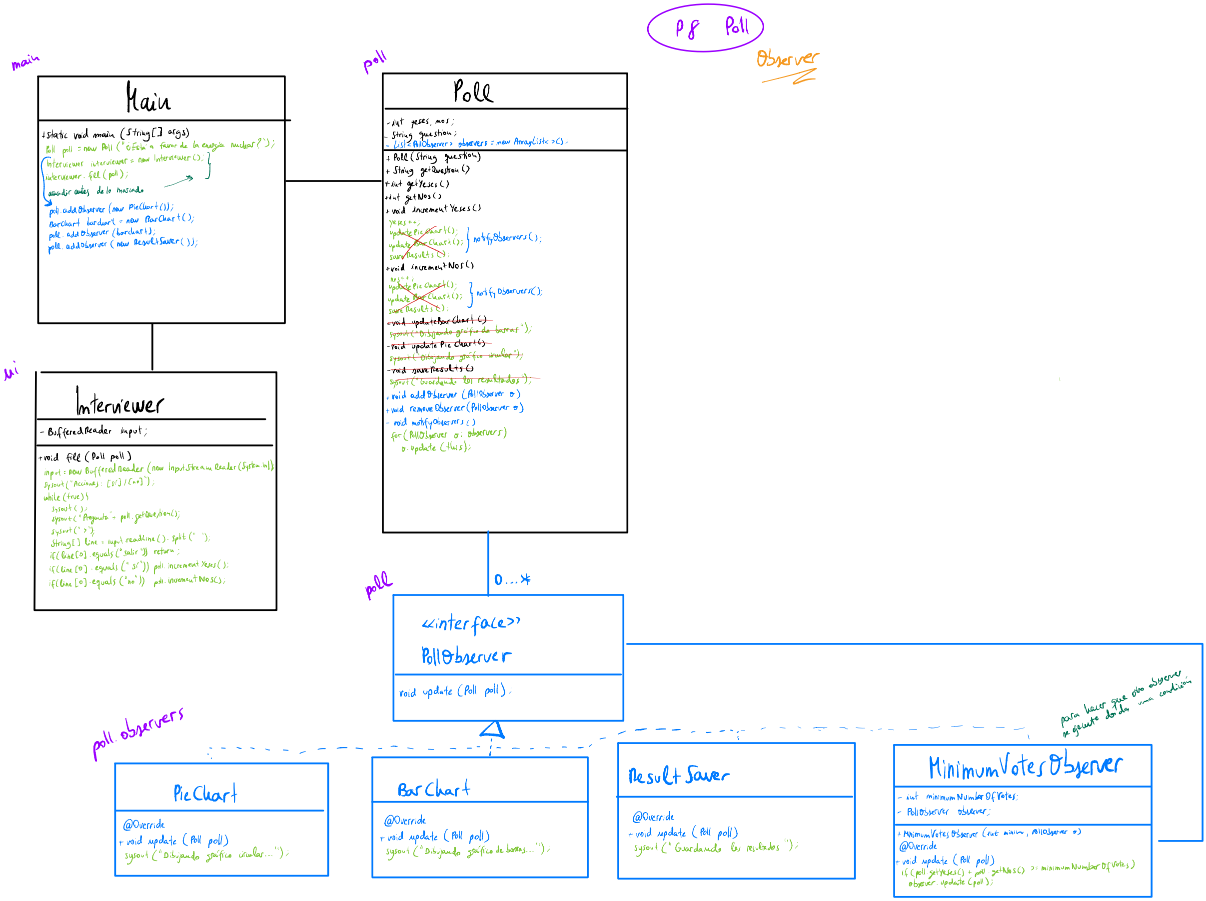




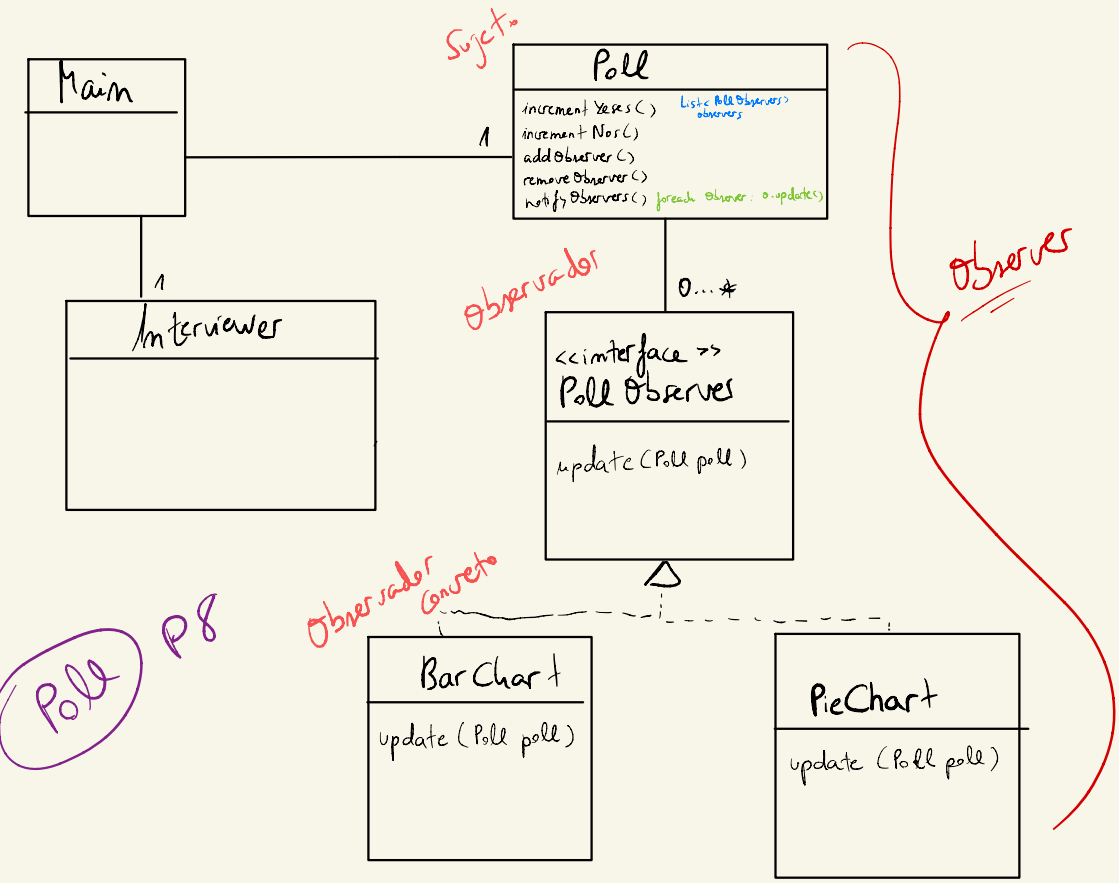
Usar el Decorator:
- Para añadir objetos individuales de forma dinámica y transparente, sin afectar a otros objetos.
- Para responsabilidades que pueden ser retiradas
- Cuando la extensión mediante la herencia no es viable

Funcionamiento (como en Main) → un Decorador → un Decorador → un Componente

new Encryptor (new SpacesFilter (new Bluetooth ("x")))

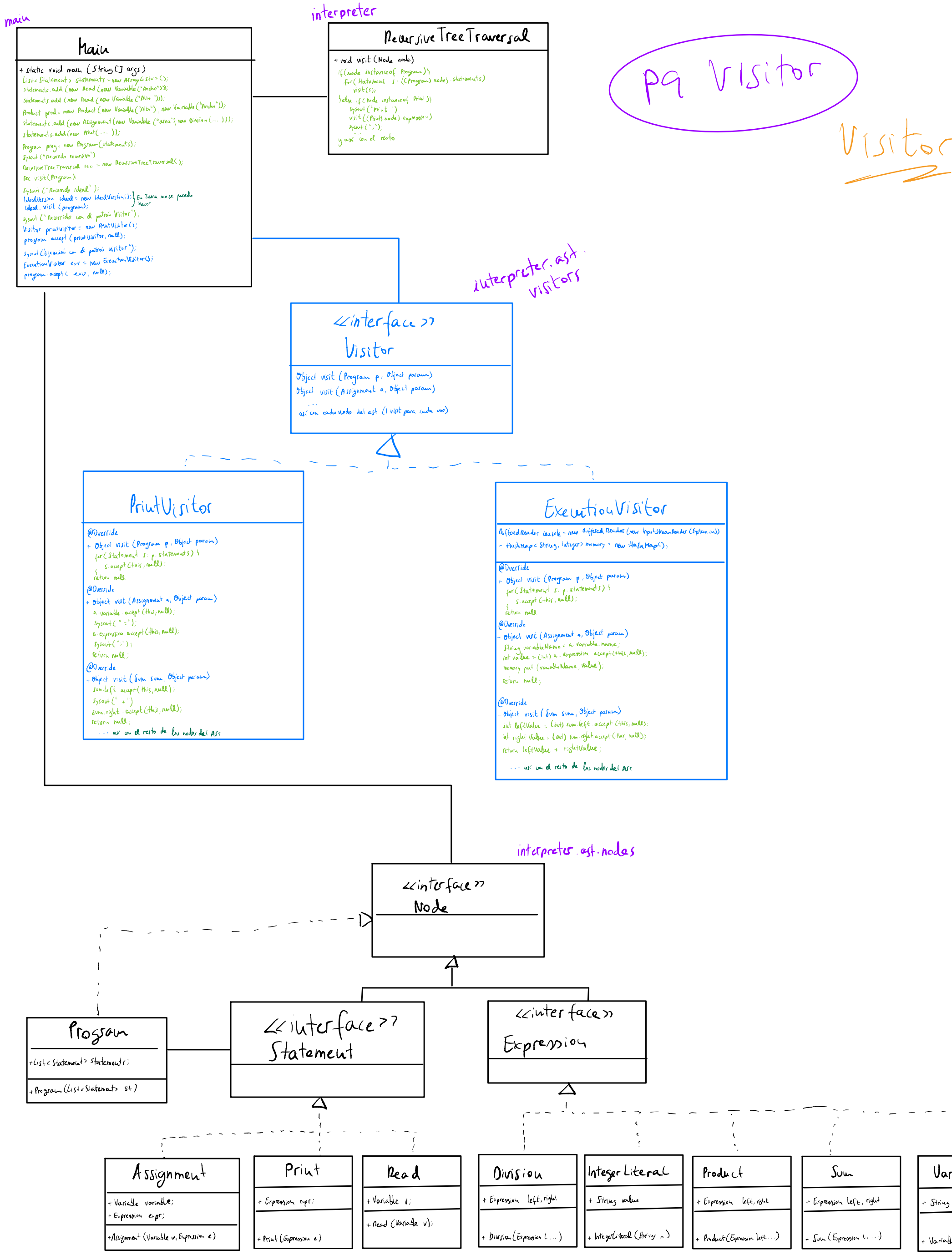


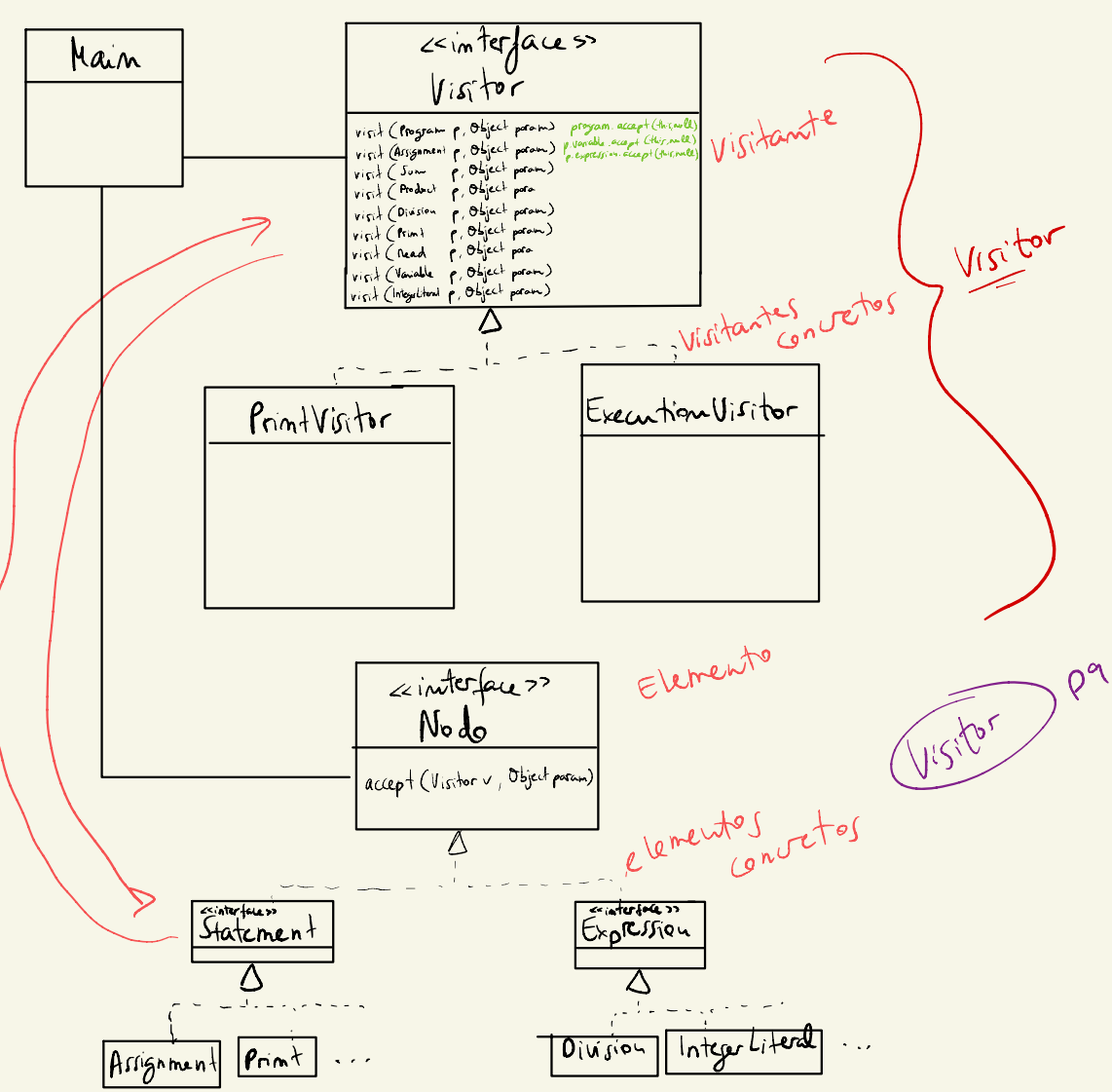
P8 Poll
Observer
main
Main
+static void main (String[] args)
Poll poll = new Poll ("¿Está a favor de la energía nuclear?");
Interviewer interviewer = new Interviewer();
interviewer.fill(poll);
añadir antes de lo marcado
poll.addObserver (new PieChart());
BarChart barchart = new BarChart();
poll.addObserver (barchart);
poll.addObserver (new ResultSaver());
poll
Poll
-int yeses, nos;
- String question;
- List<PollObserver> observers = new ArrayList<>();
+ Poll (String question)
+ String getQuestion ()
+int getYeses ()
+int getNos ()
+void incrementYeses ()
yeses++;
updatePieChart();
updateBarChart();
saveResults();
notifyObservers();
+void incrementNos ()
nos++;
updatePieChart();
updateBarChart();
saveResults();
notifyObservers();
-void updateBarChart ()
sysout("Dibujando gráfico de barras");
-void updatePieChart()
sysout("Dibujando gráfico circular");
-void saveResults ()
sysout("Guardando los resultados");
+void addObserver (PollObserver o)
+void removeObserver (PollObserver o)
- void notifyObservers ()
for (PollObserver o: observers)
o.update (this);
ui
Interviewer
- BufferedReader input;
+void fill (Poll poll)
input = new BufferedReader (new InputStreamReader (System.in));
sysout("Acciones: [sí] / [no]");
while (true){
sysout();
sysout("Pregunta" + poll.getQuestion();
sysout(">");
String[] line = input.readLine().split(" ");
if(line[0].equals("salir")) return;
if(line[0].equals("sí")) poll.incrementYeses();
if(line[0].equals("no")) poll.incrementNos();
0...*
poll
«interface»
PollObserver
void update (Poll poll);
poll.observers
PieChart
@Override
+ void update (Poll poll)
sysout ("Dibujando gráfico circular...");
BarChart
@Override
+ void update (Poll poll)
sysout ("Dibujando gráfico de barras...");
ResultSaver
@Override
+ void update (Poll poll)
sysout ("Guardando los resultados");
para hacer que otro observer se ejecute dada una condición
MinimumVotesObserver
- int minimumNumberOfVotes;
- PollObserver observer;
+ MinimumVotesObserver (int minim, PollObserver o)
@Override
+ void update (Poll poll)
if (poll.getYeses() + poll.getNos() >= minimumNumberOfVotes)
observer.update(poll);



Úsese el observer cuando:

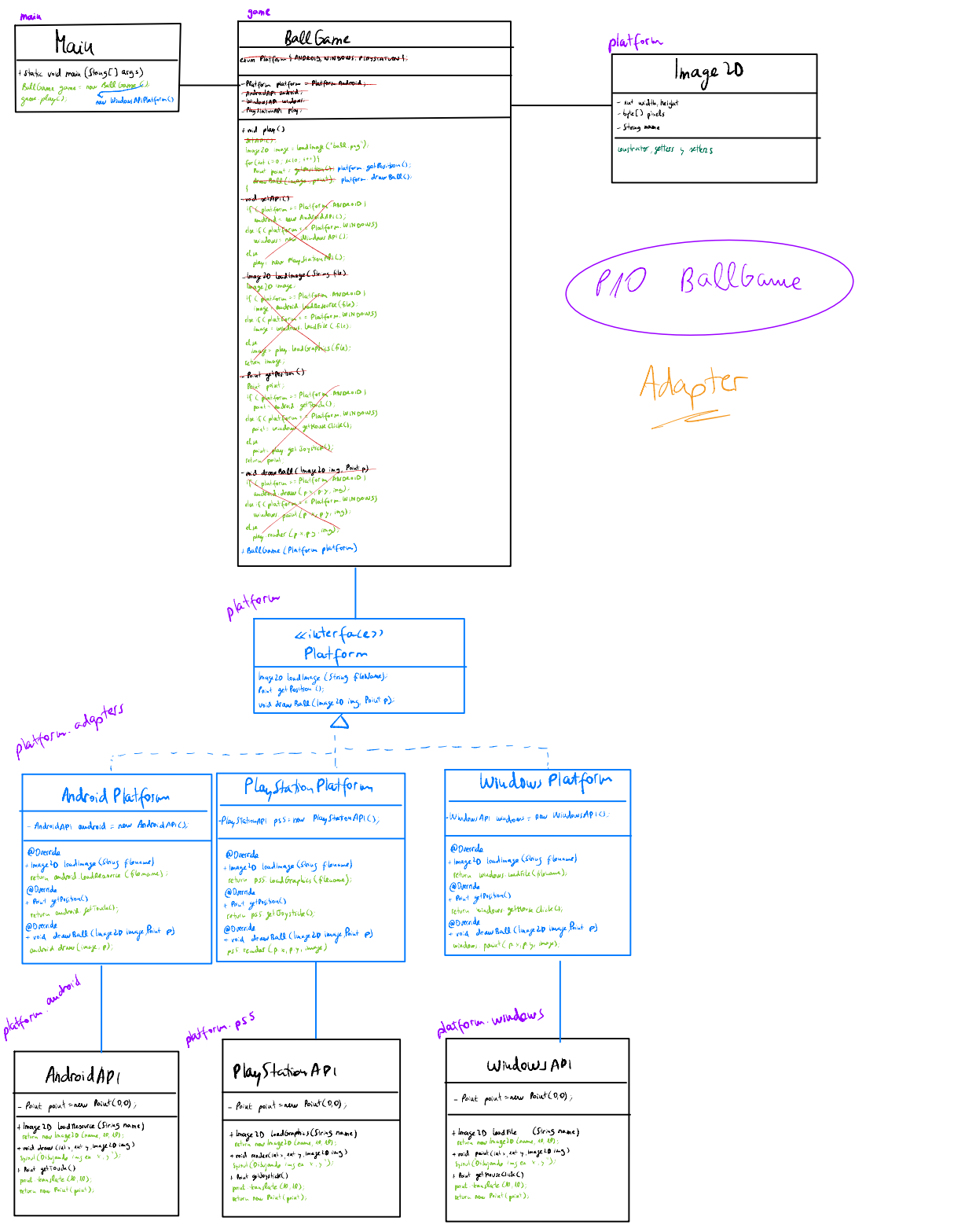
- Cuando una abstracción tiene dos aspectos y uno depende del otro.
- Cuando un cambio en un objeto requiere cambiar otros, y no sabe más cuántos objetos necesitan cambiarse.
- Cuando un objeto debería ser capaz de notificar a otros sin hacer suposiciones sobre quiénes son dichos objetos.

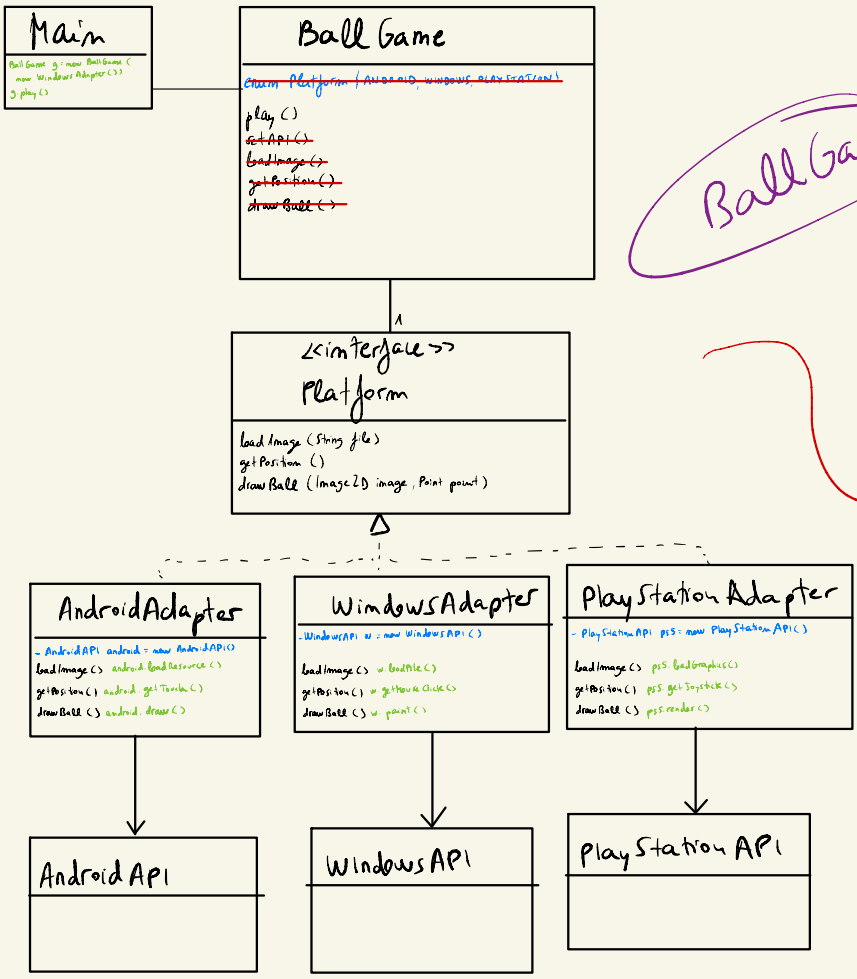

main
Main
+ static void main (String[] args)
List<Statement> statements = new ArrayList<>();
statements.add (new Read (new Variable ("Ancho")));
statements.add (new Read (new Variable ("Alto")));
Product prod = new Product (new Variable ("Alto"), new Variable ("Ancho"));
statements.add (new Assignment (new Variable ("area"), new Division (...)));
statements.add (new Print(...));
Program prog = new Program (statements);
Sysout ("Recorrido recursivo")
RecursiveTreeTraversal rec = new RecursiveTreeTraversal();
rec.visit (Program);
Sysout ("Recorrido ideal");
IdealVersion ideal = new IdealVersion();
ideal.visit (program);
En Java no se puede hacer
Sysout ("Recorrido con el patrón Visitor");
Visitor printvisitor = new PrintVisitor();
program.accept (printVisitor, null);
Sysout ("Ejecución con el patrón visitor");
ExecutionVisitor exv = new ExecutionVisitor();
program.accept (exv, null);
interpreter
RecursiveTreeTraversal
+ void visit (Node node)
if (node instanceof Program) {
for (Statement s : ((Program) node).statements)
visit(s);
} else if (node instanceof Print) {
Sysout ("print ")
visit ((Print) node).expression)
Sysout (";");
y así con el resto
pq Visitor
Visitor
interpreter.ast.visitors
<<interface>>
Visitor
Object visit (Program p, Object param)
Object visit (Assignment a, Object param)
...
así con cada nodo del ast (1 visit para cada uno)
PrintVisitor
@Override
+ Object visit (Program p, Object param)
for (Statement s : p.statements) {
s.accept (this, null);
}
return null
@Override
+ Object visit (Assignment a, Object param)
a.variable.accept (this, null);
Sysout (" = ");
a.expression.accept (this, null);
Sysout (";");
return null;
@Override
+ Object visit (Sum sum, Object param)
sum.left.accept (this, null);
Sysout (" + ")
sum.right.accept (this, null);
return null;
... así con el resto de los nodos del AST
ExecutionVisitor
BufferedReader console = new BufferedReader (new InputStreamReader (System.in))
- HashMap<String, Integer> memory = new HashMap();
@Override
+ Object visit (Program p, Object param)
for (Statement s : p.statements) {
s.accept (this, null);
}
return null
@Override
- Object visit (Assignment a, Object param)
String variableName = a.variable.name;
int value = (int) a.expression.accept (this, null);
memory.put (variableName, value);
return null;
@Override
- Object visit (Sum sum, Object param)
int leftValue = (int) sum.left.accept (this, null);
int rightValue = (int) sum.right.accept (this, null);
return leftValue + rightValue;
... así con el resto de los nodos del AST
interpreter.ast.nodes
<<interface>>
Node
Program
+ List<Statement> statements;
+ Program (List<Statement> st)
<<interface>>
Statement
<<interface>>
Expression
Assignment
+ Variable variable;
+ Expression expr;
+ Assignment (Variable v, Expression e)
Print
+ Expression expr;
+ Print (Expression e)
Read
+ Variable v;
+ Read (Variable v);
Division
+ Expression left, right
+ Division (Expression l ...)
IntegerLiteral
+ String value
+ IntegerLiteral (String x)
Product
+ Expression left, right
+ Product (Expression left ...)
Sum
+ Expression left, right
+ Sum (Expression l, ...)
Variable
+ String name
+ Variable (String name)



Úsese el patrón Visitor cuando:

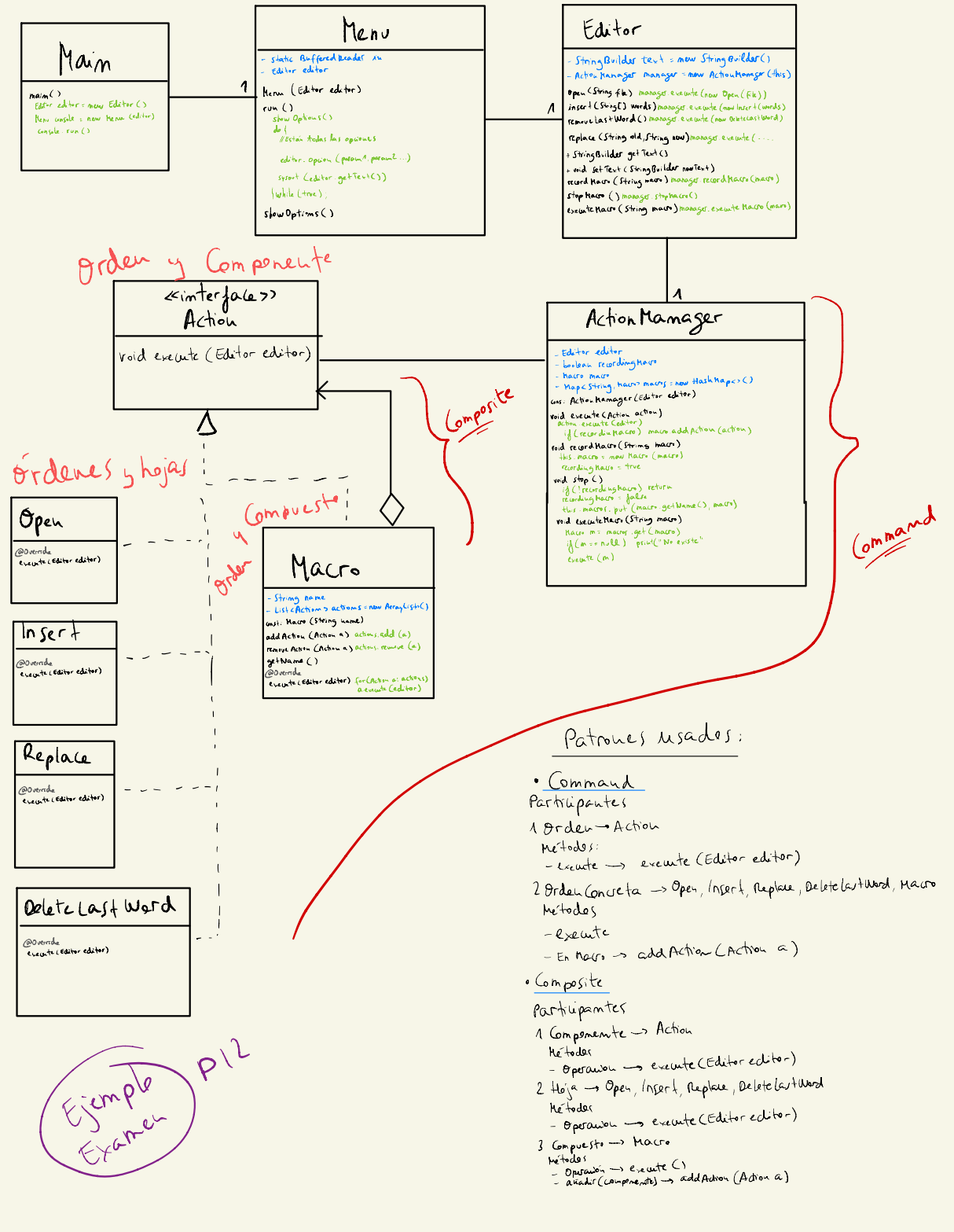
- Una estructura de objetos contiene muchas clases de objetos con diferentes interfaces y queremos realizar operaciones sobre esos elementos.
- Se necesitan realizar muchas operaciones distintas y no relacionadas sobre objetos de una estructura de objetos. Visitor permite juntar operaciones relacionadas definiéndolas en una clase.


main
Main
+ static void main (String[] args)
BallGame game = new BallGame();
game.play();
new WindowsAPIPlatform()
game
BallGame
enum Platform { ANDROID, WINDOWS, PLAYSTATION }
- Platform platform = Platform.Android;
- AndroidAPI android;
- WindowsAPI windows;
- PlayStationAPI PS4;
+ void play()
INITIALIZE
Image2D image = loadImage("ball.png");
for(int i=0; i<10; i++){
Point point = getPosition(i); platform.getPosition();
drawBall(image, point); platform.drawBall();
}
- void setAPI()
if(platform == Platform.ANDROID)
android = new AndroidAPI();
else if(platform == Platform.WINDOWS)
windows = new WindowsAPI();
else
ps4 = new PlayStationAPI();
- Image2D loadImage(String file)
Image2D image;
if(platform == Platform.ANDROID)
image = android.loadResource(file);
else if(platform == Platform.WINDOWS)
image = windows.loadFile(file);
else
image = ps4.loadGraphics(file);
return image;
- Point getPoint()
Point point;
if(platform == Platform.ANDROID)
point = android.getTouch();
else if(platform == Platform.WINDOWS)
point = windows.getMouseClick();
else
point = ps4.getJoystick();
return point;
- void drawBall(Image2D img, Point p)
if(platform == Platform.ANDROID)
android.draw(p.x, p.y, img);
else if(platform == Platform.WINDOWS)
windows.paint(p.x, p.y, img);
else
ps4.render(p.x, p.y, img);
+ BallGame(Platform platform)
platform
Image2D
- int width, height
- byte[] pixels
- String name
constructor, getters, setters
P10 BallGame
Adapter
platform
«interface»
Platform
Image2D loadImage(String fileName);
Point getPoint();
void drawBall(Image2D img, Point p);
platform: adapters
Android Platform
- AndroidAPI android = new AndroidAPI();
@Override
+ Image2D loadImage(String filename)
return android.loadResource(filename);
@Override
+ Point getPosition()
return android.getTouch();
@Override
+ void drawBall(Image2D image, Point p)
android.draw(image, p);
PlayStation Platform
-PlayStationAPI ps5 = new PlayStationAPI();
@Override
+ Image2D loadImage(String filename)
return ps5.loadGraphics(filename);
@Override
+ Point getPosition()
return ps5.getJoystick();
@Override
+ void drawBall(Image2D image, Point p)
ps5.render(p.x, p.y, image);
Window Platform
-WindowsAPI windows = new WindowsAPI();
@Override
+ Image2D loadImage(String filename)
return windows.loadFile(filename);
@Override
+ Point getPosition()
return windows.getMouseClick();
@Override
+ void drawBall(Image2D image, Point p)
windows.paint(p.x, p.y, image);
platform: android
platform: ps5
platform: windows
AndroidAPI
- Point point = new Point(0,0);
+ Image2D loadResource(String name)
return new Image2D(name, 40, 40);
+ void draw(int x, int y, Image2D img)
System.Out.println(img.name + x + y);
+ Point getTouch()
point.translate(10, 10);
return new Point(point);
PlayStationAPI
- Point point = new Point(0,0);
+ Image2D loadGraphics(String name)
return new Image2D(name, 40, 40);
+ void render(int x, int y, Image2D img)
System.Out.println(img.name + x + y);
+ Point getJoystick()
point.translate(10, 10);
return new Point(point);
WindowsAPI
- Point point = new Point(0,0);
+ Image2D loadFile(String name)
return new Image2D(name, 40, 40);
+ void paint(int x, int y, Image2D img)
System.Out.println(img.name + x + y);
+ Point getMouseClick()
point.translate(10, 10);
return new Point(point);



Úsese el Adapter cuando:

- Se quiere usar una clase existente y su interfaz no concuerda con la que necesita
- Se quiere crear una clase no reutilizable que coopere con clases no relacionadas o que no han sido previstas, es decir, que no tienen por qué tener interfaces compatibles.
- Si es necesario usar varias subclases existentes pero no es práctico adaptar su interfaz heredando de cada una de ellas.



Patrones usados:

- **Command**

Participantes

1 Orden → Action

Métodos:

- execute → execute(Editor editor)

2 Orden Concreta → Open, Insert, Replace, DeleteLastWord, Macro

Métodos

- execute
- En Macro → addAction(Action a)

- **Composite**

Participantes

1 Componente → Action

Métodos

- Operación → execute(Editor editor)

2 Hoja → Open, Insert, Replace, DeleteLastWord

Métodos

- Operación → execute(Editor editor)

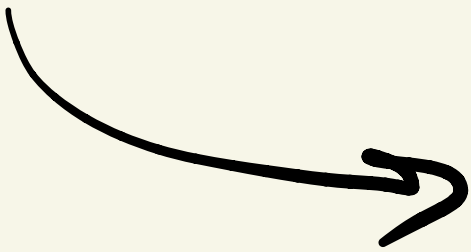
3 Compuesto → Macro

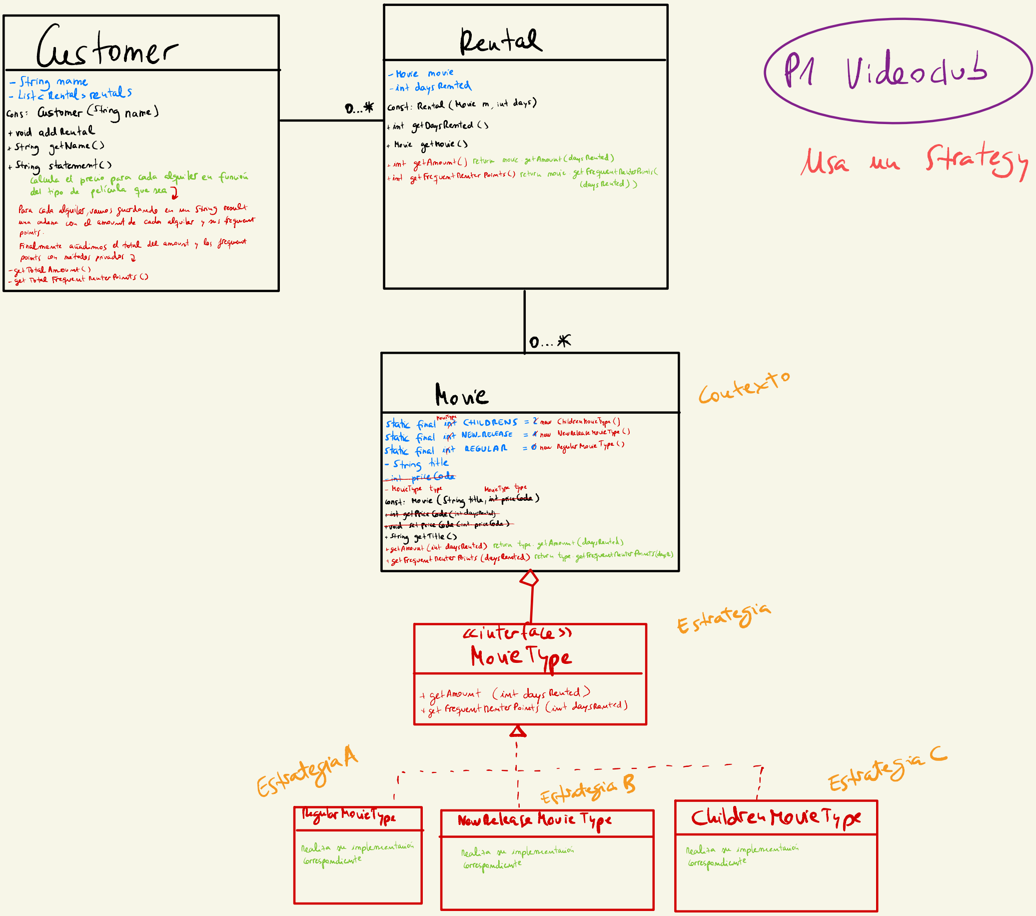
Métodos

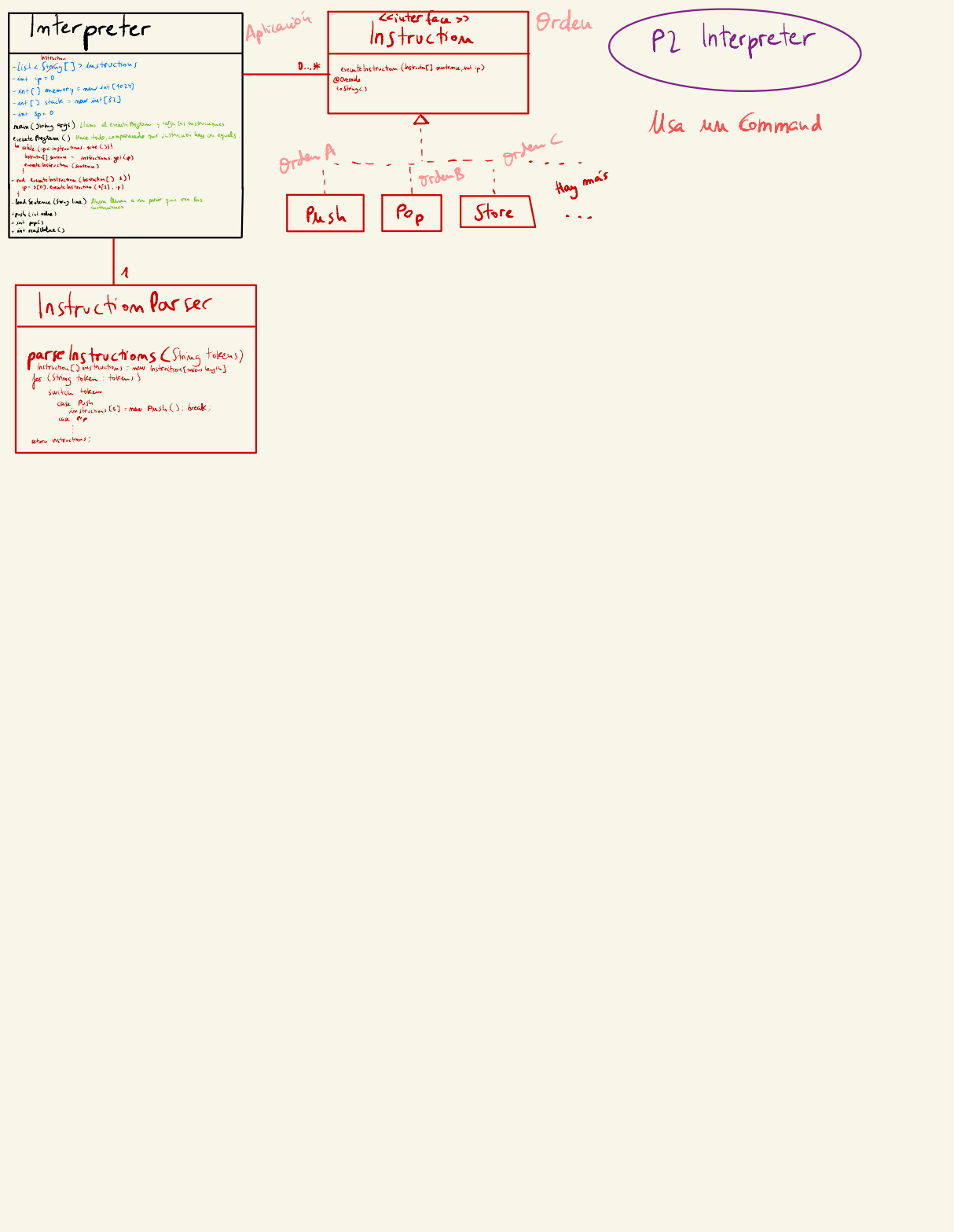
- Operación → execute()
- añadir(componente) → addAction(Action a)

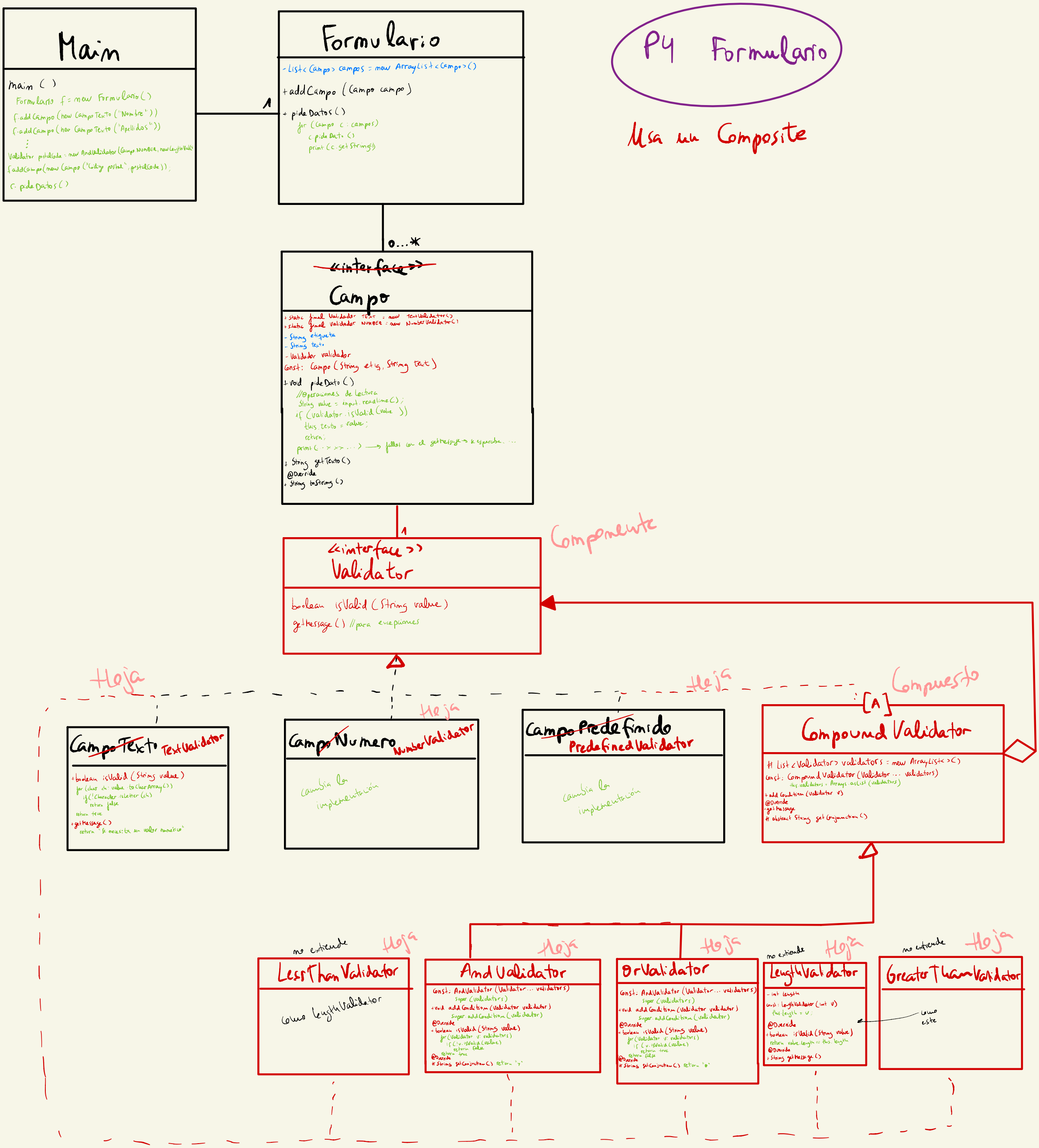
Ejemplo Examen P12

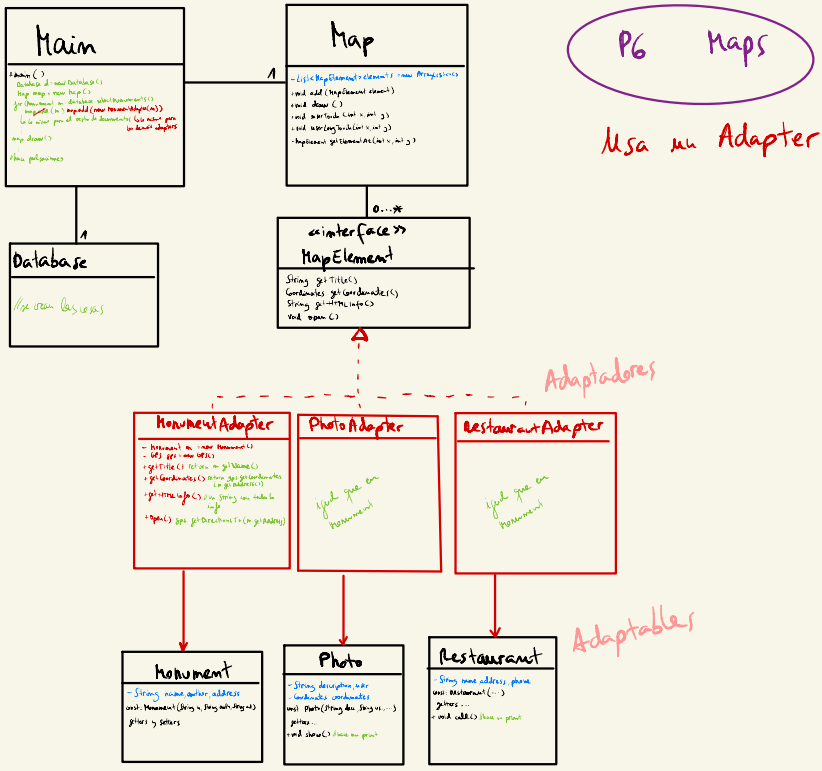
Revisita de las prácticas

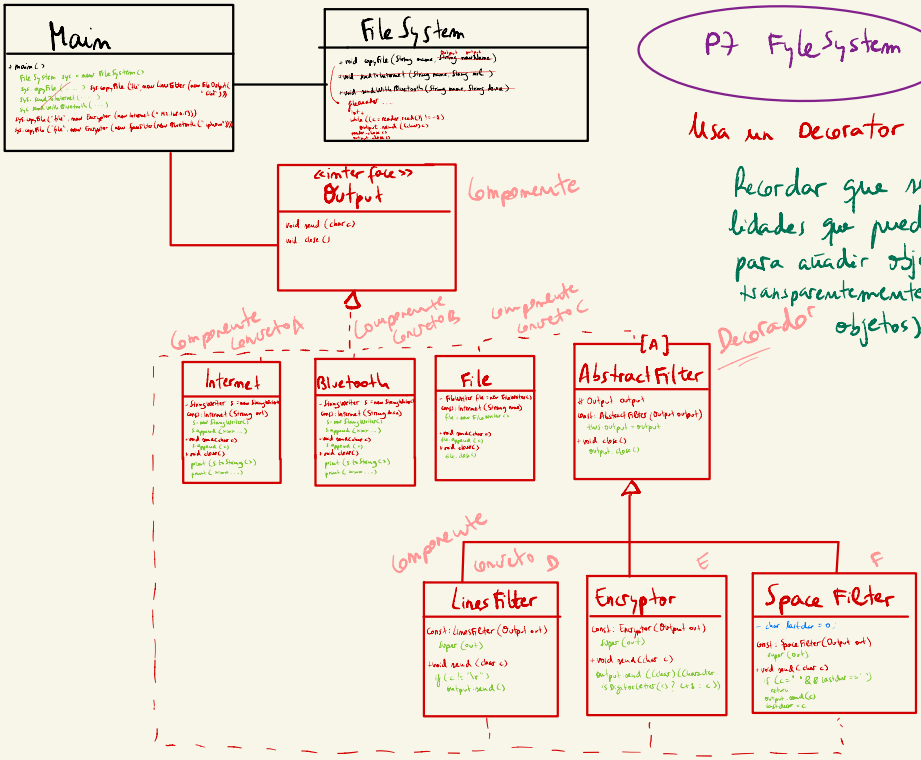


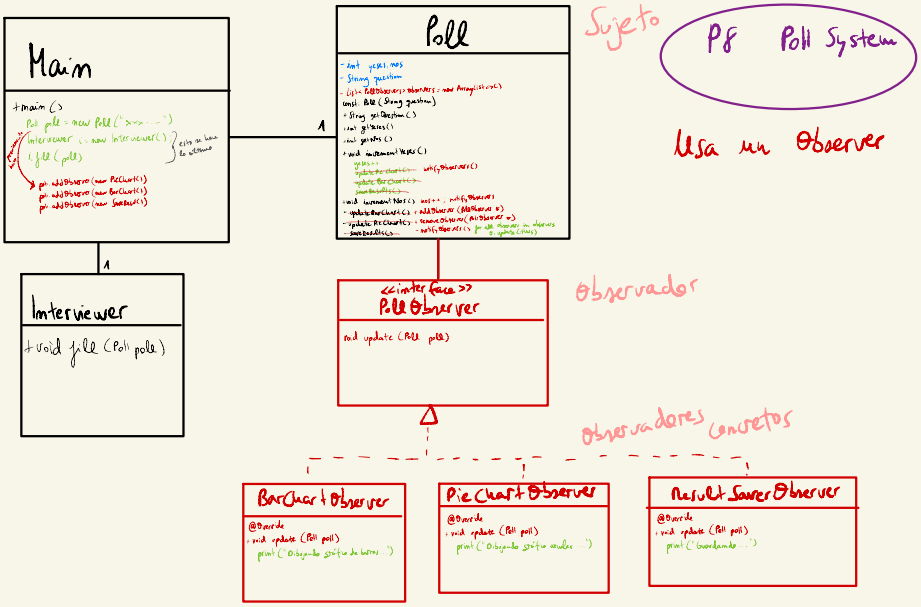

Customer
- String name
- List<Rental> rentals
Cons: Customer (String name)
+ void addRental
+ String getName()
+ String statement()
Calcula el precio para cada alquiler en función del tipo de película que sea
Para cada alquiler, vamos guardando en un String result una cadena con el amount de cada alquiler y sus frequent points.
Finalmente añadimos el total del amount y los frequent points con métodos privados
- getTotalAmount()
- get Total Frequent Renter Points ()
0...*
Rental
- Movie movie
- int daysRented
Const: Rental (Movie m, int days)
+ int getDaysRented ()
+ Movie getMovie ()
+ int getAmount() return movie.getAmount(daysRented)
+ int getFrequentRenterPoints() return movie.getFrequentRenterPoints((daysRented))
0...*
Movie
static final MovieType CHILDRENS = new ChildrenMovieType()
static final int NEW_RELEASE = new NewReleaseMovieType()
static final int REGULAR = new RegularMovieType()
- String title
- int priceCode
- MovieType type
const: Movie (String title, MovieType type)
+ int getPriceCode (int daysRented)
+ void setPriceCode (int priceCode)
+ String getTitle ()
+ getAmount (int daysRented) return type.getAmount(daysRented)
+ getFrequentRenterPoints (daysRented) return type.getFrequentRenterPoints(daysR)
Contexto
«interface»
MovieType
+ getAmount (int daysRented)
+ getFrequentRenterPoints (int daysRented)
Estrategia
Estrategia A
Estrategia B
Estrategia C
RegularMovieType
Realiza su implementación correspondiente
NewReleaseMovieType
Realiza su implementación correspondiente
ChildrenMovieType
Realiza su implementación correspondiente
P1 Videoclub
Usa un Strategy

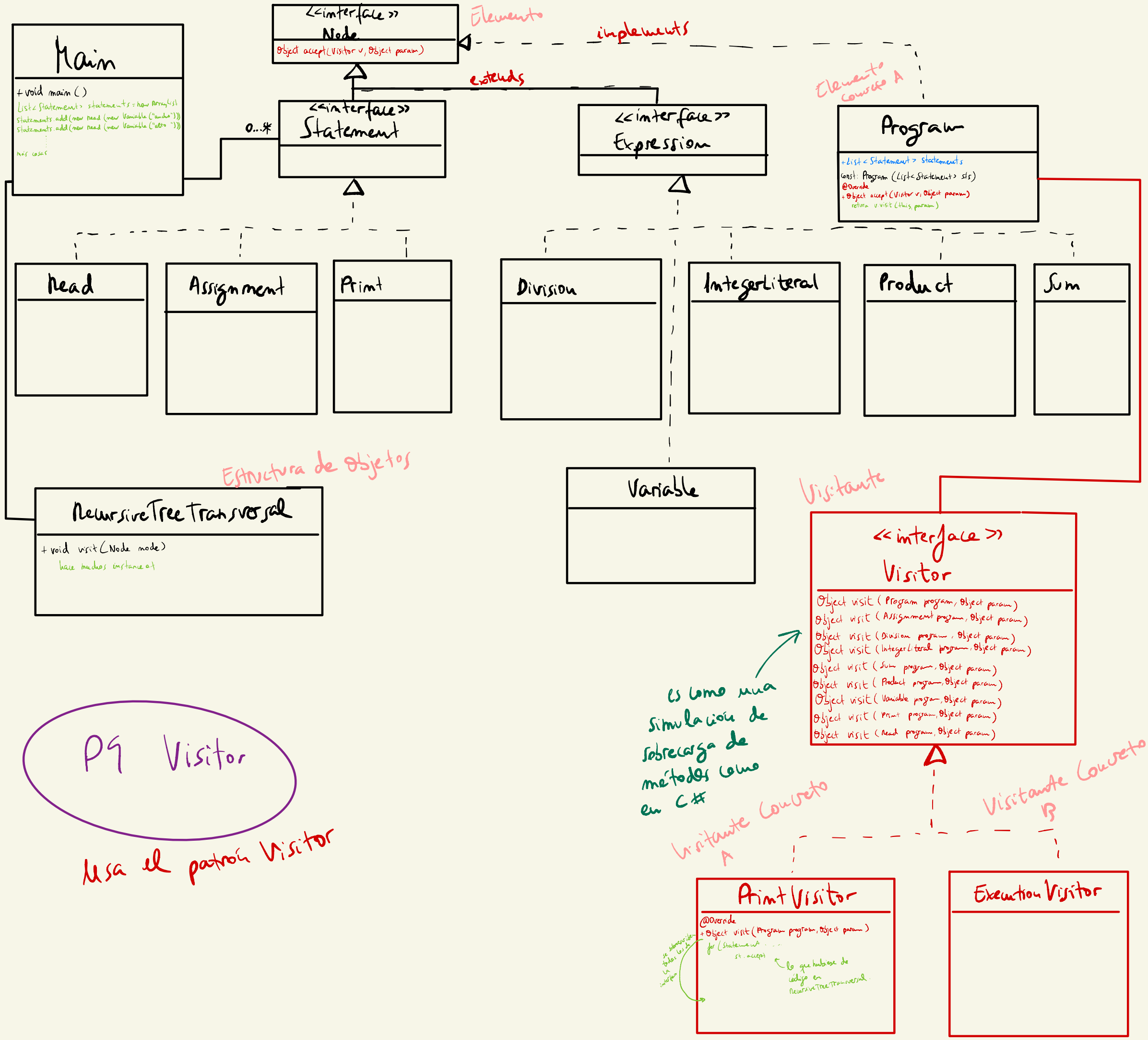

Interpreter
-List<String[]> instructions
- int ip = 0
- int[] memory = new int[1024]
- int[] stack = new int[32]
- int sp = 0
main(String args) Llama al execute Program y coloca las instrucciones
executeProgram() Hace todo, comparando que instruction hay en equals
while (ip < instructions.size())
Instruction[] instruction = instructions.get(ip)
execute Instruction (instruction)
- void executeInstruction(Instruction[] i)
ip = i[0].executeInstruction(i[1], ip)
loadSentence(String line) Mismo llamas a un parser que crea las instrucciones
+ push(int value)
+ int pop()
+ int readAddress()
Aplicación
0...*
<<interface>>
Instruction
executeInstruction(Instruction[] sentence, int ip)
@Override
toString()
Orden
Orden A
Orden B
Orden C
hay más
...
Push
Pop
Store
1
InstructionParser
parseInstructions(String tokens)
Instruction[] instructions = new Instruction[tokens.length]
for (String token : tokens)
switch token
case Push
instructions[i] = new Push(); break;
case Pop
:
return instructions;
P2 Interpreter
Usa un Command

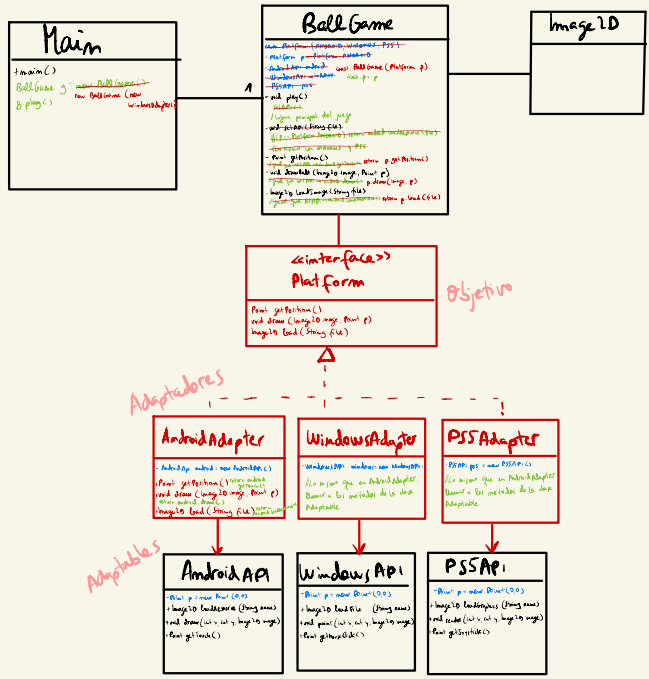

P4 Formulario
Usa un Composite
Main
main ( )
Formulario f = new Formulario ( )
f.addCampo (new CampoTexto ("Nombre"))
f.addCampo (new CampoTexto ("Apellidos"))
c.pideDatos ( )
Formulario
- List<Campo> campos = new ArrayList<Campo>()
+ addCampo (Campo campo)
+ pideDatos ( )
for (Campo c : campos)
c.pideDatos ()
1
0...*
«interface»
Campo
- String etiqueta
- String texto
- Validador validador
Const: Campo (String etiq, String text)
+ void pideDato ( )
+ String getTexto ( )
@Override
+ String toString ( )
1
Componente
«interface»
Validator
boolean isValid (String value)
getMessage ( ) //para excepciones
Hoja
Hoja
Hoja
Compuesto
CampoTexto TextValidator
CampoNumero NumberValidator
cambia la implementación
CampoPredefinido PredefinedValidator
cambia la implementación
[A]
CompoundValidator
no extiende
Hoja
LessThanValidator
como LengthValidator
Hoja
AndValidator
Hoja
OrValidator
no extiende
Hoja
LengthValidator
no extiende
Hoja
GreaterThanValidator
como este


Main
Map
P6 Maps
Usa un Adapter
1
1
0...*
Database
<<interface>>
MapElement
Adaptadores
MonumentAdapter
PhotoAdapter
RestaurantAdapter
Adaptables
Monument
Photo
Restaurant


Main
FileSystem
P7 FyleSystem
Usa un Decorator
Recordar que se usa para responsabilidades que pueden ser retiradas y para añadir objetos dinámica y transparentemente (sin afectar a otros objetos)
<<interface>>
Output
Componente
Componente concreto A
Componente concreto B
Componente concreto C
Internet
Bluetooth
File
[A]
AbstractFilter
Decorador
Componente
concreto D
E
F
LinesFilter
Encryptor
SpaceFilter

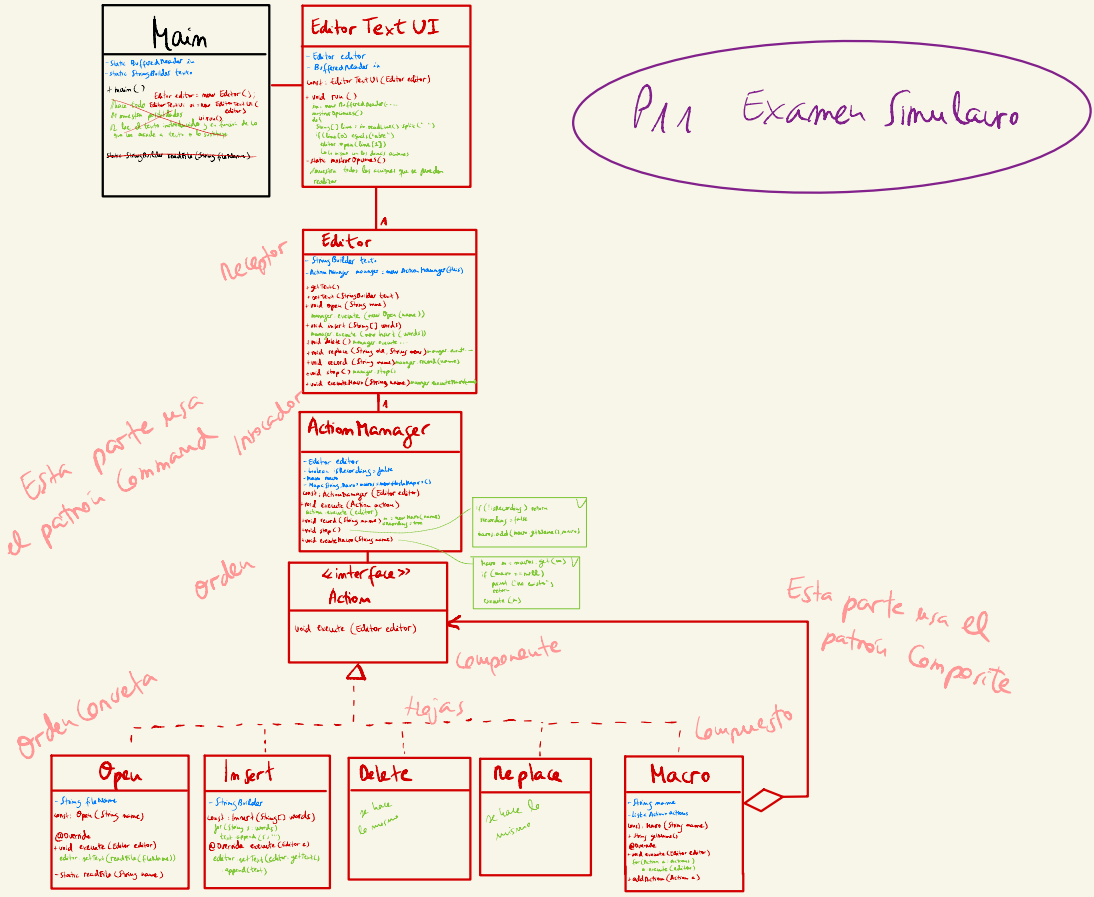

Main
+main ( )
Poll pole = new Poll ("...")
Interviewer i = new Interviewer ( )
i.fill (pole)
pole.addObserver (new PieChart())
pole.addObserver (new BarChart())
pole.addObserver (new SaveResult())
Poll
- int votes
- String question
- List<PollObserver> observers = new ArrayList<>()
+ Poll (String question)
+ String getQuestion ( )
+ int getVotes()
+ int getId ( )
+ void incrementVotes ( )
+ void addObserver (PollObserver o)
+ void removeObserver (PollObserver o)
- void notifyObservers ( )
Sujeto
P8 Poll System
Usa un Observer
1
Interviewer
+ void fill (Poll pole)
<<interface>>
PollObserver
void update (Poll pole)
Observador
Observadores concretos
BarChartObserver
@Override
+ void update (Poll poll)
PieChartObserver
@Override
+ void update (Poll poll)
ResultSaverObserver
@Override
+ void update (Poll poll)

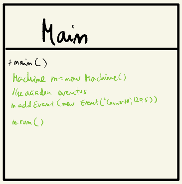

Main
+ void main ( )
List<Statement> statements = new ArrayList
statements.add(new Read (new Variable ("ancho")))
statements.add(new Read (new Variable ("alto ")))
más cosas
0...*
<<interface>>
Node
Object accept(Visitor v, Object param)
Elemento
implements
extends
<<interface>>
Statement
<<interface>>
Expression
Elemento concreto A
Program
+ List < Statement > statements
const: Program (List<Statement> sts)
@Override
+ Object accept (Visitor v, Object param)
return v.visit (this, param)
Read
Assignment
Print
Division
IntegerLiteral
Product
Sum
Variable
Estructura de objetos
RecursiveTreeTransversal
+ void visit (Node node)
hace muchos instance of
Visitante
<< interface >>
Visitor
Object visit (Program program, Object param)
Object visit (Assignment program, Object param)
Object visit (Division program , Object param)
Object visit (IntegerLiteral program, Object param)
Object visit (Sum program, Object param)
Object visit (Product program, Object param)
Object visit (Variable progran, Object param)
Object visit (Print program, Object param)
Object visit (Read program, Object param)
es como una simulación de sobrecarga de métodos como en C#
Visitante Concreto A
Visitante Concreto B
PrintVisitor
@Override
+ Object visit (Program program, Object param)
for (Statement . . .
st . accept
lo que hubiese de código en RecursiveTreeTransversal.
ExecutionVisitor
P9 Visitor
Usa el patrón Visitor

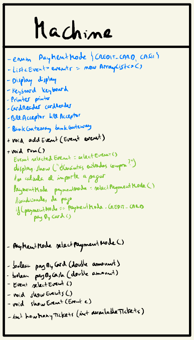


P10 Ball Game

Usa el patrón Adapter


Main
Editor Text UI
P11 Examen Simulacro
Editor
Receptor
Esta parte usa el patrón Command
Invocador
ActionManager
Orden
<<interface>>
Action
Componente
Esta parte usa el patrón Composite
Orden Concreta
Hojas
Compuesto
Open
Insert
Delete
Replace
Macro


Main
+ main ( )
Machine m = new Machine()
//se añaden eventos
m.addEvent (new Event ("Concierto", 120, 5))
m.run ( )


Machine
- enum PaymentMode {CREDIT_CARD, CASH}
- List<Event> events = new ArrayList<>()
- Display display
- Keyboard keyboard
- Printer printer
- CardReader cardReader
- BillAcceptor billAcceptor
- BankGateway bankGateway
+ void addEvent (Event event)
+ void run()
Event selectedEvent = selectEvent()
display.show ("¿Cuántas entradas compra?")
//se calcula el importe a pagar
PaymentMode paymentMode = selectPaymentMode ( )
//condiciones de pago
if (paymentMode == PaymentMode.CREDIT_CARD
payByCard()
- PaymentMode selectPaymentMode ( )
- boolean payByCard (double amount)
- boolean payByCash (double amount)
- Event selectEvent ()
- void showEvents ()
- void showEvent (Event e)
- int howManyTickets (int availableTickets)