Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

Trabajo Práctico 3: Programación en Objetos

Paradigmas de Lenguajes de Programación

Segundo Cuatrimestre de 2014

Apellido y Nombre	LU	E-mail
Aisemberg, Dan	242/12	dea4493@hotmail.com
Almansi, Emilio	674/12	ealmansi@gmail.com
Levy Alfie, Jonás	081/12	jonaslevy5@gmail.com

Índice

1. Observaciones **3**

2. Código fuente **3**

2.1. Eleccion . . . 3

2.1.1. Lagarto . . . 4

2.1.2. Papel . . . 5

2.1.3. Piedra . . . 5

2.1.4. Spock . . . 6

2.1.5. Tijera . . . 6

2.2. GeneradorRandomParaTest . . . 7

2.3. Juego . . . 7

2.4. Jugador . . . 8

2.4.1. JugadorAdaptativo . . . 8

2.4.2. JugadorAleatorio . . . 9

2.4.3. JugadorHumano . . . 9

2.4.4. JugadorSiempre . . . 10

2.5. Resultado . . . 10

2.5.1. Empate . . . 10

2.5.2. Victoria . . . 11

2.6. TP3Tests . . . 11

1. Observaciones

En la siguiente sección, incluimos el código fuente desarrollado. Para obtener una versión en texto plano del código exportamos el mismo mediante el comando **File Out** de *Pharo*, y posteriormente lo editamos manualmente para mejorar su legibilidad.

Por cada clase que forma parte de la solución, presentamos los métodos de instancia y los métodos de clase asociados. Los métodos de instancia se preceden por un encabezado con el siguiente formato:

```
A subclass: #B
```

indicando la definición de la clase B como subclase de la clase A. Después de los métodos de instancia, sigue un encabezado con este otro formato:

```
B class
```

precediendo a los métodos de clase de la clase B.

2. Código fuente

2.1. Eleccion

```
Object subclass: #Eleccion

ganadorContraLagarto
    ^ self subclassResponsibility.

ganadorContraTijera
    ^ self subclassResponsibility.

ganadorContraPapel
    ^ self subclassResponsibility.

ganadorContraPiedra
    ^ self subclassResponsibility.

ganadorContraSpock
    ^ self subclassResponsibility.

ganadorContra: otraEleccion
    ^ self subclassResponsibility.

pierdeContra: otraEleccion
    ^ (self noEmpataContra: otraEleccion)
        and:
        [ (self ganaContra: otraEleccion) not ]

empataContra: otraEleccion
    ^ (self ganadorContra: otraEleccion) isNil.

ganaContra: otraEleccion
    ^ (self noEmpataContra: otraEleccion)
        and:
        [ (self ganadorContra: otraEleccion) = self ]

noEmpataContra: otraEleccion
```

```
    ^ (self empataContra: otraEleccion) not.

asString
    ^ self class name.

eleccionSuperadora
    "devuelve una elección cualquiera que le gane a self, o nil si no existe ninguna"
    | eleccion |
    eleccion := nil.
    Eleccion todas do: [ :item |  (item ganaContra: self) ifTrue: [ eleccion := item ] ].
    ^ eleccion

Eleccion class

papel
    ^ Papel singleton.

piedra
    ^ Piedra singleton.

todas
    ^ self allSubclasses collect: [ :item | item singleton ].

lagarto
    ^ Lagarto singleton.

spock
    ^ Spock singleton.

tijera
    ^ Tijera singleton.
```

2.1.1. Lagarto

```
Eleccion subclass: #Lagarto
    classVariableNames: 'UniqueInstance'

ganadorContraLagarto
    ^ nil.

ganadorContraTijera
    ^ Eleccion tijera.

ganadorContraPapel
    ^ self.

ganadorContraPiedra
    ^ Eleccion piedra.

ganadorContraSpock
    ^ self.

ganadorContra: otraEleccion
    ^ otraEleccion ganadorContraLagarto.

Lagarto class

singleton
```

```
    "devuelve la unica instancia de la clase"
    UniqueInstance ifNil: [ UniqueInstance := self new ].
    ^UniqueInstance
```

2.1.2. Papel

```
Eleccion subclass: #Papel
    classVariableNames: 'UniqueInstance'

ganadorContraLagarto
    ^ Eleccion lagarto.

ganadorContraTijera
    ^ Eleccion tijera.

ganadorContraPapel
    ^ nil.

ganadorContraPiedra
    ^ self.

ganadorContraSpock
    ^ self.

ganadorContra: otraEleccion
    ^ otraEleccion ganadorContraPapel.

Papel class

singleton
    "devuelve la unica instancia de la clase"
    UniqueInstance ifNil: [ UniqueInstance := self new ].
    ^UniqueInstance
```

2.1.3. Piedra

```
Eleccion subclass: #Piedra
    classVariableNames: 'UniqueInstance'

ganadorContraLagarto
    ^ self.

ganadorContraTijera
    ^ self.

ganadorContraPapel
    ^ Eleccion papel.

ganadorContraPiedra
    ^ nil.

ganadorContraSpock
    ^ Eleccion spock.

ganadorContra: otraEleccion
    ^ otraEleccion ganadorContraPiedra.
```

```
Piedra class

singleton
    "devuelve la unica instancia de la clase"
    UniqueInstance ifNil: [ UniqueInstance := self new ].
    ^UniqueInstance
```

2.1.4. Spock

```
Eleccion subclass: #Spock
    classVariableNames: 'UniqueInstance'

ganadorContraLagarto
    ^ Eleccion lagarto.

ganadorContraTijera
    ^ self.

ganadorContraPapel
    ^ Eleccion papel.

ganadorContraPiedra
    ^ self.

ganadorContraSpock
    ^ nil.

ganadorContra: otraEleccion
    ^ otraEleccion ganadorContraSpock.

Spock class

singleton
    "devuelve la unica instancia de la clase"
    UniqueInstance ifNil: [ UniqueInstance := self new ].
    ^UniqueInstance
```

2.1.5. Tijera

```
Eleccion subclass: #Tijera
    classVariableNames: 'UniqueInstance'

ganadorContraLagarto
    ^ self.

ganadorContraTijera
    ^ nil.

ganadorContraPapel
    ^ self.

ganadorContraPiedra
    ^ Eleccion piedra.

ganadorContraSpock
    ^ Eleccion spock.
```

```
ganadorContra: otraEleccion
    ^ otraEleccion ganadorContraTijera.

Tijera class

singleton
    "devuelve la unica instancia de la clase"
    UniqueInstance ifNil: [ UniqueInstance := self new ].
    ^UniqueInstance
```

2.2. GeneradorRandomParaTest

```
Object subclass: #GeneradorRandomParaTest
    instanceVariableNames: 'siguiente'

siguiente: unNumeroAleatorio
    siguiente := unNumeroAleatorio.

nextInt: anInteger
    ^ siguiente.
```

2.3. Juego

```
Object subclass: #Juego
    instanceVariableNames: 'jugador1 jugador2 resultado eleccion1 eleccion2'

resultado
    resultado isNil ifTrue: [ self iniciar ].
    ^ resultado.

revancha
    ^ self iniciar.

iniciar
    eleccion1 := jugador1 eleccion.
    eleccion2 := jugador2 eleccion.
    resultado := Resultado jugador1: jugador1 conEleccion: eleccion1
                           jugador2: jugador2 conEleccion: eleccion2.
    resultado notificarAJugadores.

jugador1: unJugador jugador2: otroJugador
    jugador1 := unJugador.
    jugador2 := otroJugador.

Juego class

mejorDe: cantidadDeJuegos contra: unJugador
    Juego mejorDe: cantidadDeJuegos entre: Jugador humano y: unJugador.

mejorDe: cantidadDeJuegos entre: unJugador y: otroJugador
    | j |
    j := Juego entre: unJugador y: otroJugador.
    cantidadDeJuegos timesRepeat: [ j iniciar ].

entre: unJugador y: otroJugador
    ^ self new jugador1: unJugador jugador2: otroJugador;
           yourself.
```

2.4. Jugador

```
Object subclass: #Jugador

perdioEligiendo: miEleccion contra: otroJugador eligiendo: eleccionContraria
    "Implementar si hace falta actuar en base a este evento."

ganoEligiendo: miEleccion contra: otroJugador eligiendo: eleccionContraria
    "Implementar si hace falta actuar en base a este evento."

eleccion
    self subclassResponsibility.

empatoEligiendo: miEleccion contra: otroJugador eligiendo: eleccionContraria
    "Implementar si hace falta actuar en base a este evento."

Jugador class

aleatorioConGenerador: unGenerador
    ^ JugadorAleatorio new
        generador: unGenerador;
        yourself.

conHistoriaHasta: cantidadDeResultadosAGuardar
    ^ self shouldBeImplemented.

siempre: unaEleccion
    ^ JugadorSiempre new
        juegaCon: unaEleccion;
        yourself.

humano
    ^ JugadorHumano new.

aleatorio
    ^ JugadorAleatorio new
        generador: (Random new);
        yourself.

adaptativoIniciandoCon: unaEleccion
    ^ JugadorAdaptativo new
        iniciaCon: unaEleccion;
        yourself.
```

2.4.1. JugadorAdaptativo

```
Jugador subclass: #JugadorAdaptativo
    instanceVariableNames: 'eleccion'

perdioEligiendo: miEleccion contra: otroJugador eligiendo: eleccionContraria
    "actualiza la elección del jugador adaptativo a la elección superadora de aquella que le ganó"
    eleccion := eleccionContraria eleccionSuperadora.

eleccion
    "getter de la eleccion"
    ^ eleccion.

iniciaCon: unaEleccion
```

```
    "setea la elección inicial del jugador adapatativo"
    eleccion := unaEleccion.

empatoEligiendo: miEleccion contra: otroJugador eligiendo: eleccionContraria
    "actualiza la elección del jugador adaptativo a la elección superadora de aquella que le empató"
    eleccion := eleccionContraria eleccionSuperadora.
```

2.4.2. JugadorAleatorio

```
Jugador subclass: #JugadorAleatorio
    instanceVariableNames: 'generador'

generador: unGenerador
    "setter del generador del jugador aleatorio"
    generador := unGenerador.

eleccion
    | todas |
    todas := Eleccion todas.
    ^ todas at: (generador nextInt: (todas size)).

JugadorAleatorio class
```

2.4.3. JugadorHumano

```
Jugador subclass: #JugadorHumano

perdioEligiendo: miEleccion contra: otroJugador eligiendo: eleccionContraria
    "Informa al jugador humano que ha ganado la partida"
    UITheme current
        messageIn: Morph new
        text: ('Perdiste contra ', (eleccionContraria className))
        title: 'Resultado'.

ganoEligiendo: miEleccion contra: otroJugador eligiendo: eleccionContraria
    "Informa al jugador humano que ha ganado la partida"
    UITheme current
        messageIn: Morph new
        text: ('Le ganaste a ', (eleccionContraria className))
        title: 'Resultado'.

eleccion
    ^ UITheme current
        chooseDropListIn: Morph new
        text: 'Qué jugamos?'
        title: 'Nueva elección'
        list: Eleccion todas.

empatoEligiendo: miEleccion contra: otroJugador eligiendo: eleccionContraria
    "Informa al jugador humano que ha ganado la partida"
    UITheme current
        messageIn: Morph new
        text: ('Empataste contra ', (eleccionContraria className))
        title: 'Resultado'.
```

2.4.4. JugadorSiempre

```
Jugador subclass: #JugadorSiempre
    instanceVariableNames: 'eleccion'

juegaCon: unaEleccion
    "setea la elección fija del jugador siempre"
    eleccion := unaEleccion.

eleccion
    "getter de la eleccion"
    ^ eleccion.
```

2.5. Resultado

```
Object subclass: #Resultado

Resultado class

jugador1: unJugador conEleccion: unaEleccion jugador2: otroJugador conEleccion: otraEleccion
    (unaEleccion empataContra: otraEleccion)
    ifTrue: [
        ^ Empate entre: unJugador eligiendo: unaEleccion y: otroJugador eligiendo: otraEleccion.
    ].
    ^ (unaEleccion ganaContra: otraEleccion)
        ifTrue: [ Victoria de: unJugador eligiendo: unaEleccion contra: otroJugador
            eligiendo: otraEleccion
        ]
        ifFalse: [ Victoria de: otroJugador eligiendo: otraEleccion contra: unJugador
            eligiendo: unaEleccion
        ].
```

2.5.1. Empate

```
Resultado subclass: #Empate
    instanceVariableNames: 'jugador1 eleccion1 jugador2 eleccion2'

notificarAJugadores
    jugador1 empatoEligiendo: eleccion1 contra: jugador2 eligiendo: eleccion2.
    jugador2 empatoEligiendo: eleccion2 contra: jugador1 eligiendo: eleccion1.

ganador
    ^ nil.

jugador1: unJugador eleccion1: unaEleccion jugador2: otroJugador eleccion2: otraEleccion
    jugador1 := unJugador.
    eleccion1 := unaEleccion.
    jugador2 := otroJugador.
    eleccion2 := otraEleccion.

esEmpate
    ^ true.

Empate class

entre: unJugador eligiendo: unaEleccion y: otroJugador eligiendo: otraEleccion
    ^ self new
```

```
        jugador1: unJugador
        eleccion1: unaEleccion
        jugador2: otroJugador
        eleccion2: otraEleccion;
        yourself.
```

2.5.2. Victoria

```
Resultado subclass: #Victoria
    instanceVariableNames: 'ganador eleccionGanadora perdedor eleccionPerdedora'

notificarAJugadores
    ganador ganoEligiendo: eleccionGanadora contra: perdedor eligiendo: eleccionPerdedora.
    perdedor perdioEligiendo: eleccionPerdedora contra: ganador eligiendo: eleccionGanadora.

ganador
    ^ ganador

ganador: unJugador eleccionGanadora: unaEleccion perdedor: otroJugador eligiendo: otraEleccion
    ganador := unJugador.
    eleccionGanadora := unaEleccion.
    perdedor := otroJugador.
    eleccionPerdedora := otraEleccion.

esEmpate
    ^ false.

Victoria class

de: unJugador eligiendo: unaEleccion contra: otroJugador eligiendo: otraEleccion
    ^ self new
        ganador: unJugador
        eleccionGanadora: unaEleccion
        perdedor: otroJugador
        eligiendo: otraEleccion;
        yourself.
```

2.6. TP3Tests

```
TestCase subclass: #TP3Tests

test01LaIgualdadYElHashDeEleccionNoDistingueInstanciasDelMismoTipo
    self assert: Eleccion piedra = Eleccion piedra.
    self assert: Eleccion papel = Eleccion papel.
    self assert: Eleccion tijera = Eleccion tijera.
    self assert: Eleccion piedra hash = Eleccion piedra hash.
    self assert: Eleccion papel hash= Eleccion papel hash.
    self assert: Eleccion tijera hash= Eleccion tijera hash.


test02SeCumplenLasReglasDelJuego
    "Implementar los distintos ganaContraX para lograr la funcionalidad pedida"
    self assert: (Eleccion piedra ganaContra: Eleccion tijera).
    self assert: (Eleccion piedra pierdeContra: Eleccion papel).
    self assert: (Eleccion piedra empataContra: Eleccion piedra).
    self assert: (Eleccion papel ganaContra: Eleccion piedra).
    self assert: (Eleccion papel pierdeContra: Eleccion tijera).
```

```
    self assert: (Eleccion papel empataContra: Eleccion papel).
    self assert: (Eleccion tijera ganaContra: Eleccion papel).
    self assert: (Eleccion tijera pierdeContra: Eleccion piedra).
    self assert: (Eleccion tijera empataContra: Eleccion tijera).

test03SePuedeObtenerUnaOpcionSuperadoraParaCualquierEleccion
    Eleccion todas allSatisfy: [ :eleccion |
        eleccion eleccionSuperadora ganaContra: eleccion.
    ]

test04UnJuegoConsisteEnHacerCompetirLaEleccionDeDosJugadores
    | jugadorPiedra jugadorPapel |
    "https://www.youtube.com/watch?feature=player_detailpage&v=dwj254ofJbk#t=14"
    jugadorPiedra := Jugador siempre: Eleccion piedra.
    jugadorPapel := Jugador siempre: Eleccion papel.
    self assert: (Juego entre: jugadorPiedra y: jugadorPapel) resultado ganador = jugadorPapel.
    self assert: (Juego entre: jugadorPiedra y: jugadorPiedra) resultado esEmpate.

test05UnJugadorAdaptativoCambiaSuEleccionCuandoPierde
    | jugador contrincante |
    jugador := Jugador adaptativoIniciandoCon: Eleccion piedra.
    contrincante := Jugador siempre: Eleccion papel.
    self assert: jugador eleccion = Eleccion piedra.
    (Juego entre: jugador y: contrincante) iniciar.
    self assert: (jugador eleccion ganaContra: contrincante eleccion).

test06UnJugadorAdaptativoCambiaSuEleccionCuandoEmpata
    | jugador contrincante |
    jugador := Jugador adaptativoIniciandoCon: Eleccion piedra.
    contrincante := Jugador siempre: Eleccion piedra.
    self assert: jugador eleccion = Eleccion piedra.
    (Juego entre: jugador y: contrincante) iniciar.
    self assert: (jugador eleccion ganaContra: contrincante eleccion).

test07ElJuegoSeAdaptaANuevasReglas
    "https://www.youtube.com/watch?v=_PUEoDYpUyQ"
    self assert: (Eleccion piedra pierdeContra: Eleccion spock).
    self assert: (Eleccion papel ganaContra: Eleccion spock).
    self assert: (Eleccion lagarto empataContra: Eleccion lagarto).
    self assert: (Eleccion spock pierdeContra: Eleccion lagarto).

test08UnJugadorAleatorioUtilizaElGeneradorParaDecidirSuEleccion
    | jugador generador |
    generador := GeneradorRandomParaTest new.
    jugador := Jugador aleatorioConGenerador: generador.
    generador siguiente: (Eleccion todas indexOf: Eleccion piedra).
    self assert: jugador eleccion = Eleccion piedra.
    generador siguiente: (Eleccion todas indexOf: Eleccion spock).
    self assert: jugador eleccion = Eleccion spock.
```