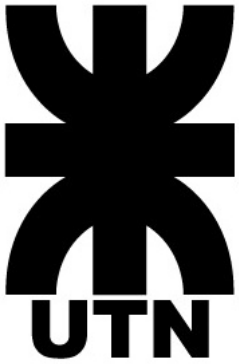

Resumen Paradigma Orientado a Objetos
Facultad Regional Buenos Aires
Universidad Tecnologica Nacional

Paradigmas de Programación – $K2032$

Trabajos Prácticos

PAZ PORTILLA, José Miguel 2028244
jpazportilla@frba.utn.edu.ar

27 de noviembre de 2023

Índice

1. Introducción a Objetos ->Video 17 Youtube

En este paradigma de objetos se vuelve a tener efecto, pero es menos declarativo que funcional y lógico. Se utiliza el lenguaje de programación Wollok. La idea es combinar estructuras de datos y operaciones.

Las caracteristicas de un objeto son:

1. Exponen una interfaz, es un conjunto de operaciones con las que se pueden interactuar con el objetos. Solo se puede interactuar con objetos mediantes mensajes. Los mensajes que un objeto entiende va a ser el resultado de poseer metodos.
2. Pueden llegar a tener estado interno, que son atributos, es decir referencias a otros objetos. Estos atributos pueden cambiar de referencia y apuntar a otros objetos.
3. Tienen una identidad, cada objeto es diferente a cualquier otro, aunque hayan otros que respondan a los mismos mensajes y estado interno.

1.1. Problema: La Golondrina Pepita

- Un ornitólogo nos pide ayuda para estudiar el Consumo de Energía de la golondrina Pepita
- El Volar consume energia de Pepita.
- El Comer recupera la energía de Pepita.

1.1.1. Resolución: La Golondrina Pepita

- La palabra object define un objeto nuevo.
- La palabra var define un atributo que podrá ser cambiado
- La palabra method permite crear métodos.
- El metodo volar() y comer() causan efecto.
- El metodo energia() son solo de consulta.

1.1.2. Wollok: La Golondrina Pepita

```
1  object pepita
2  {
3      var energia = 100
4
5      method vola (kilometros)
6      {
7          energia = energia - kilometros * 2
8      }
9
10     method come (gramos)
11     {
12         energia = energia + gramos * 10
13     }
14
15     //Es un getter, obtiene el valor del atributo energia y lo retorna
```

```
16      //Cuando retorna una expresion se utiliza esa forma de definir metodos
17      method energia () = energia
18      //Se podria haber definido asi
19      /*
20       method energia ()
21       {
22         return energia
23       }
24      */
25  }
```

1.2. Problema: La entrenadora de aves Emilia

- Emilia solo sabe entrenar aves
- La aves deben comer 5g, volar 10km y volver a comer 5g.

1.2.1. Resolución: La entrenadora de aves Emilia

- Emilia no conoce a Pepita, pero como Pepita entiende los mensajes come y vola, entonces podra entrenarla.

1.2.2. Wollok: La entrenadora de aves Emilia

```
1   import pepita.*
2   import pepote.*
3
4   object emilia
5   {
6       method entrena (ave)
7       {
8           ave.come(5)
9           ave.vola(10)
10          ave.come(5)
11      }
12  }
```

1.3. Emilia entrena a Pepita

Emilia puede entrenar a cualquier objeto que entienda los mensajes come y vola. En la Figura 1 se observa que pepita sufre el efecto luego que emilia la entrena, aumentando su energia de $100J$ a $180J$.

1.3.1. Wollok: Emilia entrena a Pepita

```
1   import pepita.*
2   import emilia.*
3
4   describe "Emilia conoce a su golondrina Pepita"
5   {
6       test "Pepita inicia con 100J de ejergia"
```

```
7       {
8           assert.equals(100,pepita.energia())
9       }
10      test "Emilia entrena a Pepita y finalmente tiene 180J de energia"
11      {
12          emilia.entrena(pepita)
13          assert.equals(180,pepita.energia())
14          /*Ya que luego de 5g su energia aumento en 5 * 10= 50J
15           * Luego volar 10km y su energia disminuyo en 10 * 2 = 20J
16           * Finalmente volvio a comer 5g y su energia aumento en 5*10 = 50J
17           * por lo tando la energia final resulta 100+50-20+50=180J
18           */
19      }
20  }
```

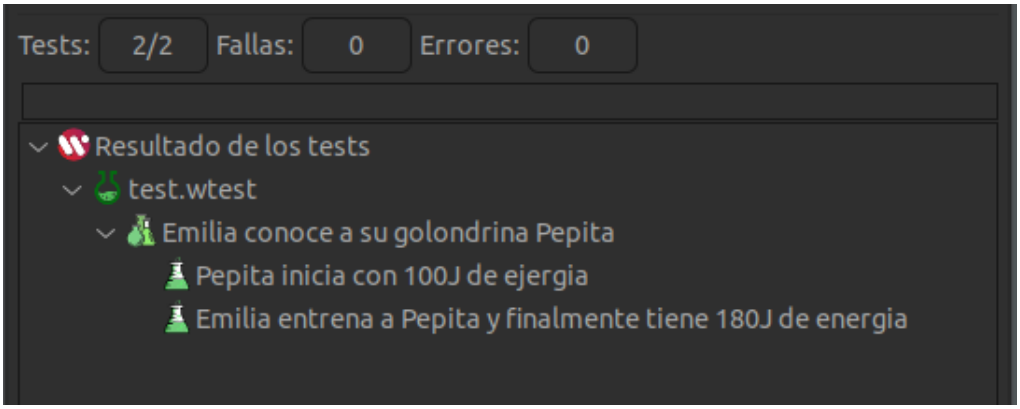

Figura 1. Test unitario de pepita siendo entrenada por emilia.

1.4. Conceptos clave de la programación orientada a objetos

1. Los objetos están encapsulados ya que oculta y protege de los detalles de su implementación. Solo veo la interfaz de un objeto, o sea que mensajes entiende. Solo nos intereza saber que puede hacer. No puedo ver, ni usar los atributos de un objeto, sólo un objeto puede manipular sus atributos.

2. Cuando se delega en un objeto alguna actividad, se le da la responsabilidad al objeto de saber como hacerlo mientras que lo termine haciendo.

3. El polimorfismo implica que un objeto que envia mensajes pueda manipular al menos a dos objetos, siempre y cuando ellos entiendan los mensajes que envia el objeto.

1.5. Otra ave: El hancón pepote

Pepote es otra ave ya que entiende los mismos mensajes que Pepita, es decir come y vola. Para emilia que sabe entrenar aves le es indiferente cual de ellos debe entrenar. Pepote causa efecto sobre su energia al volar y comer, pero lo hace de forma distinta a Pepita. Se observa que su tiene como atributos a comido y volado, ademas su metodo energia no es un getter, sino que devuelve el resultado una operación.

1.6. Wollok: El hancón pepote

```
object pepote
{
    var volado = 0
    var comido = 0

    method vola (kilometros)
    {
        volado = volado + kilometros
    }
    method come(gramos)
    {
        comido = comido + gramos
    }
    method energia () = 255 + comido **2 - volado / 5
}
```

1.7. Wollok-Test: Emilia entrena a Pepote

```
import pepote.*
import emilia.*

describe "Emilia conoce a su Halcón Pepote"
{
    test "Pepote inicia con 255J de ejergia"
    {
        assert.equals(255,pepote.energia())
    }
    test "Emilia entrena a Pepote y finalmente tiene 353J de energia"
    {
        emilia.entrena(pepote)
        assert.equals(353,pepote.energia())
        /*Ya que luego de comer 5g
         * luego de volar 10km
         * y volver a comer 5g, sus atributos quedan
         * volado 10km y comido 10g, por lo tanto
         * su energia es 255 + 10**2 - 10/5 =
         * = 255 + 100 -2 = 353
         */
    }
}
```

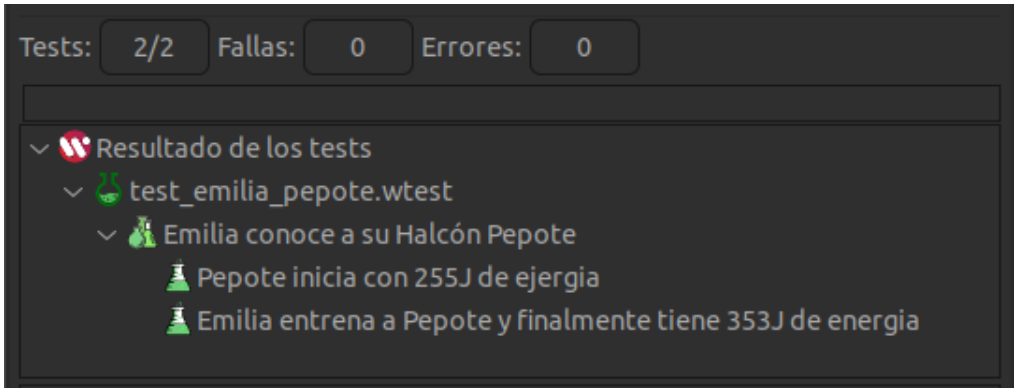

Figura 2. Test unitario de pepote siendo entrenado por emilia.

1.8. Diseño de métodos para favorecer el polimorfismo entre objetos

Si se tiene un objeto Pepaza que entiende los mensajes:

- come()
- volar(kilometros)
- nadar()

Emilia no la podrá entrenar ya que Pepaza no entiene el mensaje vola, sino volar. Además su metodo come no espera un parametro que indique la cantidad que debe comer. Solución cambiar volar por vola y agregar parametro gramos en el metodo come aunque no se use.

Si se tiene otro objeto Pepudo que entiende los mensajes:

- come()
- nada()

Emilia tampoco lo podrá entrenar a Pepudo, ya que no entiende el mensaje vola.

1.9. Otro entrenado Ramiro

Ramiro en otro entrenado de aves. Si esta de buen humor hace volar a las aves 15km, sino 30km. Se sabe que si duerme por lo menos 8 horas entonces esta de buen humor, sino no.

1.10. Resolución: Otro entrenado Ramiro

Por ser entrenador, debe enterder el mensaje entrena que tiene un ave como parametro. Ademas tiene un atributo variable que indica las horas que durmio. Lo inicio en 0 horas dormida y entonces esta de mal humor. Si entreda a Pepita o Pepote los hara volar 30km, pero si en cambien durmio al menos 8 horas, entonces solo hará volar al ave que entrene 15km

1.11. Wollok: Otro entrenador Ramiro

```
1 import pepita.*
2 import pepote.*
3
```

```
4  object ramiro
5  {
6      var horasDormidas = 0
7
8      //Getter->Permite obtener el valor del atributo horasDormidas
9      method horasDormidas () = horasDormidas
10     //Setter->Permite establecer un valor _horasDormidas al atributo
           horasDormidas
11     method horasDormidas (_horasDormidas)
12     {
13         horasDormidas = _horasDormidas
14     }
15
16     //Retorna true si horasDormidas es mayor o igual a 8, sino retorna
           false
17     method estaDeBuenHumor () = horasDormidas >= 8
18
19     method entrena (ave)
20     {
21         //Si estaDeBuenHumor es true, asigna 15 a la constante distancia,
               sino asigna 30
22         const distancia = if ( self.estaDeBuenHumor() ) 15 else 30
23         //Hace volar a ave la distancia que establecio previamente
24         ave.vola(distancia)
25     }
26 }
```

1.12. Wollok-Test: Ramiro entrena a Pepita y Pepote, con buen y mal humor

```
1  import pepita.*
2  import pepote.*
3  import ramiro.*
4
5  describe "Ramiro entrena aves con mal humor"
6  {
7      test "Ramiro no durmio y tiene mal humor"
8      {
9          assert.notThat(ramiro.estaDeBuenHumor())
10     }
11     test "Pepita inicialmente tiene 100J de energia, si es entrenada por
           ramiro un dia que tiene mal humor termina con 40J de energia"
12     {
13         assert.equals(100,pepita.energia())
14         ramiro.entrena(pepita)
15         /*
16          * Ramiro hace volar a pepita 30km, su energia disminuye en 30*2=60J
                .
17          * Por lo tanto luego de ser entrenada por ramiro termina con
                100-60=40J
18         */
19         assert.equals(40,pepita.energia())
20     }
21     test "Pepote inicialmente tiene 255J de energia, si es entrenada por
           ramiro un dia que tiene mal humor termina con 249J de energia"
```

```
22      {
23          assert.equals(255,pepote.energia())
24          ramiro.entrena(pepote)
25          /*
26           * Ramiro hace volar a pepote 30km, entonces ha volado 30km y
27           * por ende su energia es 255 - 30/5 = 249J
28          */
29          assert.equals(249,pepote.energia())
30      }
31  }
32
33  describe "Ramiro entrena aves con buen humor"
34  {
35      test "Ramiro durmio 8 horas y tiene buen humor"
36      {
37          ramiro.horasDormidas(8)
38          assert.that(ramiro.estaDeBuenHumor())
39      }
40      test "Pepita inicialmente tiene 100J de energia, si es entrenada por
            ramiro un dia que tiene buen humor termina con 70J de energia"
41      {
42          ramiro.horasDormidas(8)
43          assert.equals(100,pepita.energia())
44          ramiro.entrena(pepita)
45          /*
46           * Ramiro hace volar a pepita 15km, su energia disminuye en 15*2=30J
                .
47           * Por lo tanto luego de ser entrenada por ramiro termina con
                100-30=70J
48          */
49          assert.equals(70,pepita.energia())
50      }
51      test "Pepote inicialmente tiene 255J de energia, si es entrenada por
            ramiro un dia que tiene buen humor termina con 252J de energia"
52      {
53          ramiro.horasDormidas(8)
54          assert.equals(255,pepote.energia())
55          ramiro.entrena(pepote)
56          /*
57           * Ramiro hace volar a pepote 15km, entonces ha volado 15km y
58           * por ende su energia es 255 - 15/5 = 252J
59          */
60          assert.equals(252,pepote.energia())
61      }
62  }
```

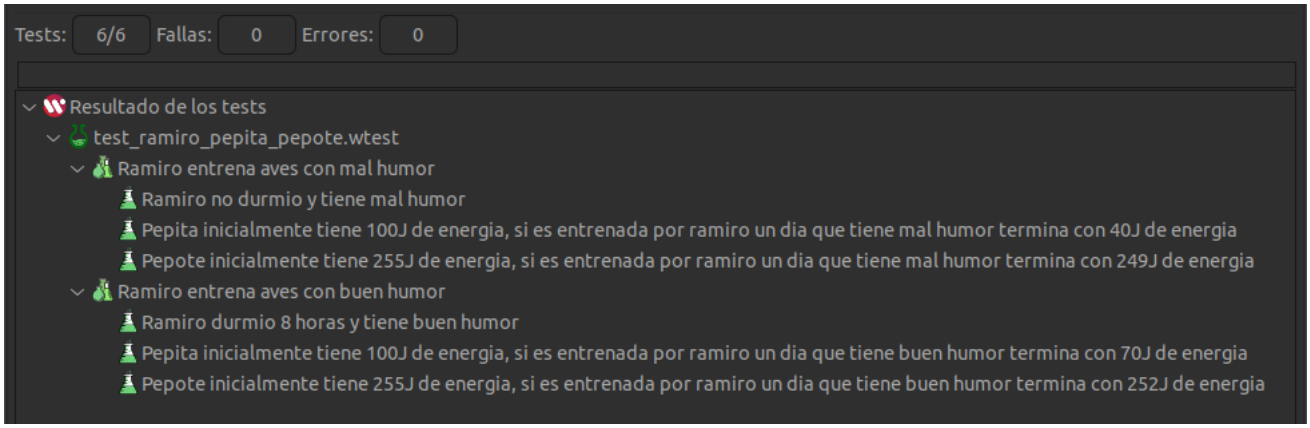

Figura 3. Test unitario de ramiro entrenado con buen y mal humor a pepita y pepote.

2. Práctica: La Feria ->Video 18 Youtube

Julieta cuenta con tickets y ademas no está cansada. Va a jugar los distintos juegos de la feria y al finalizar termina con más tickets y cansada dependiendo de cada juego.

2.1. Wollok: Jugadores de la feria

```
1  import juegos.*
2  import premios.*
3
4  object julieta
5  {
6      //Cantidad de Tickets del año anterior, puede ser modificado
7      var property tickets = 15
8      //Esta completamente descansada
9      var cansancio = 0
10
11     method punteria() = 20
12     method fuerza() = 80 - cansancio
13
14     //Cada vez que juega algun juego, cambian el valor de sus atriburos
           tickets y cansancio
15     method jugar ( juego )
16     {
17         tickets = tickets + juego.ticketsGanados (self)
18         cansancio = cansancio + juego.cansancioQueProduce ()
19     }
20     method puedeCanjear ( premio ) = tickets >= premio.costo()
21 }
22 object gerundio
23 {
24     method jugar() { }
25     method puedeCanjear ( premio ) = true
26 }
```

2.2. Wollok: Juegos de la feria

```
object tiroAlBlanco
{
   method ticketsGanados (jugador) = ( jugador.punteria() / 10 ).roundUp()
   method cansancioQueProduce () = 3
}

object pruebaDeFuerza
{
   method ticketsGanados (jugador) = if (jugador.fuerza() > 75) 20 else 0
   method cansancioQueProduce () = 8

}

object ruedaDeLaFortuna
{
   var property aceitada = true

   //0.randomUpTo(20) Genera un numero real aleatorio entre 0 y 20
   //roundUp() Redonde un numero real a uno entero, redondeando para
       arriba
   method ticketsGanados (jugador) = 0.randomUpTo(20).roundUp()
   method cansancioQueProduce() = if (aceitada) 0 else 1
}
```

2.3. Wollok: Premio de la feria

```
object osito
{
   method costo () =    45
}

object taladro
{
   var property costo = 200
}
```

2.4. Wollok-Test: Julieta juega en la feria

```
import jugadores.*
import juegos.*

describe "Julieta juega en la feria que llego al pueblo"
{
   test "Julieta tiene 15 tickets del año anterior"
   {
      assert.equals (15,julieta.tickets())
   }
   test "Julieta como esta descansada(cansancio 0 puntos) tiene 80N de
       fuerza"
```

```
11      {
12          assert.equals (80,julieta.fuerza())
13      }
14      test "Julieta tiene 20 puntos de punteria"
15      {
16          assert.equals(20,julieta.punteria())
17      }
18      test "Julieta juega tiro al blanco y termina con 17 tickets y como le
            produce 3 puntos de cansancio, tiene 77N de fuerza despues de jugar
            "
19      {
20          julieta.jugar(tiroAlBlanco)
21          assert.equals (17,julieta.tickets())
22          assert.equals (77,julieta.fuerza())
23      }
24      test "Julieta juega prueba de fueza y termina con 35 tickets ya que
            tenia 80 puntos de fuerza y como le produce 8 puntos de cansancio,
            tiene 72N de fuerza despues de jugar"
25      {
26          julieta.jugar(pruebaDeFuerza)
27          assert.equals (35,julieta.tickets())
28          assert.equals (72,julieta.fuerza())
29      }
30      test "Si Julieta vuelve a jugar prueba de fuerza seguira teniendo 35
            tickets ya que no supera los 75 N de fuerza"
31      {
32          julieta.jugar(pruebaDeFuerza)
33          julieta.jugar(pruebaDeFuerza)
34          assert.equals (35,julieta.tickets())
35      }
36      test "Si Julieta juega la rueda de la fortuna gana entre 0 y 20 tickets
             más"
37      {
38          julieta.jugar(ruedaDeLaFortuna)
39          assert.that(julieta.tickets()>=15 and julieta.tickets()<=35)
40      }
41      test "Si la rueda de la fortuna esta aceitada no produce cansancio y
            sigue teniendo 80N de fuerza"
42      {
43          julieta.jugar(ruedaDeLaFortuna)
44          assert.equals (80,julieta.fuerza())
45      }
46      test "Si la rueda de la fortuna  no esta aceitada no produce 1 punto de
             cansancio y termina con 79N de fuerza"
47      {
48          ruedaDeLaFortuna.aceitada(false)
49          julieta.jugar(ruedaDeLaFortuna)
50          assert.equals (79,julieta.fuerza())
51      }
52  }
```

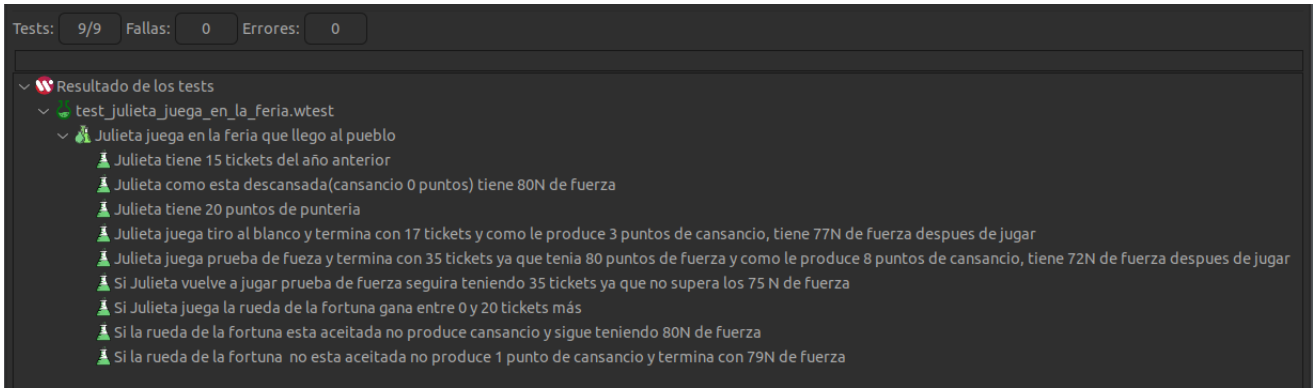

Figura 4. Test unitario julieta juega en la feria.

Práctica 1: Julieta en la Feria

PdeP JN - 2020 - Objetos

Queremos hacer un sistema para registrar la visita de Julieta a la feria que llegó al pueblo.

En la feria hay varios juegos, todos distintos y, cada vez que Julieta juega a uno, gana algún número de tickets, que luego puede cambiar por fabulosos premios. Algunos de los juegos que hay en la feria son:

Tiro al Blanco: En este juego los chicos le disparan a patitos de madera usando un rifle de aire comprimido. La cantidad de tickets que otorga depende de cuántos patos tire (para ser más exactos, se entrega un ticket por cada 10 puntos de puntería del participante, redondeando para arriba). Jugar este juego tensa un poco a los chicos, causandoles 3 puntos de cansancio.

Prueba de Fuerza: Este juego está equipado con un balancín y una plomada instalada en una corredera vertical, que termina en una campana. Los chicos golpean el balancín con una maza de madera y, si hacen sonar la campana, reciben 20 tickets (y si no, nada). Para poder hacer sonar la campana hace falta una fuerza de, al menos, 75 puntos. De más está decir que este juego es agotador (produce 8 puntos de cansancio cada vez que se juega).

Rueda de la Fortuna: Este es un juego puramente de azar. Los participantes hacen girar una rueda gigante que, al detenerse, indica cuántos tickets ganó. La rueda premia un número aleatorio de tickets entre 1 y 20. Cuando está bien aceitada la rueda gira suavemente y sin esfuerzo, pero cuando pasa mucho tiempo sin mantenimiento empieza a hacer fricción y causa un 1 punto de cansancio al girarla.

De Julieta sabemos que es una chica muy fuerte (tiene 80 puntos de fuerza, menos el cansancio que acumule), pero su puntería no es muy buena (apenas 20 puntos). También sabemos que llega a la feria completamente descansada y con 15 tickets que guardó del año anterior.

Se pide:

a. Modelar a Julieta y los juegos mencionados, de forma tal que podamos hacer que Julieta juegue y nos informe cuántos tickets tiene.
b. Extender el programa para saber si Julieta puede canjear alguno de los premios disponibles. La mano viene dura y en la feria quedan solamente dos premios disponibles: Un osito de peluche que cuesta 45 tickets y un taladro rotopercutor Bosch de 750w y mandril de 13mm, cuyo costo el dueño de la feria ajusta todos los días en base al precio del dólar.
c. Agregar al sistema a Gerundio, otro chico más que visita la feria. Geru es el hijo del dueño así que no necesita juntar tickets y puede canjear el premio que quiera cuando quiera. Para pensar: ¿Qué mensajes necesita entender Gerundio?