Guía 8 - Ejercicios adicionales para el primer parcial
Objetos 1 – UNQ
15/05/2017

1. Method Lookup

Ejercicio 1: Virus

A partir de las siguientes definiciones de clase, y considerando las variables definidas en el test

```
class Virus {
  var potencia = 1000
  var cantidadDeInfectados = 0

  method infectar(cuerpo) {
    potencia -= cuerpo.peso()
    cantidadDeInfectados += 1
  }

  method potencia() {
    return potencia
  }

  method tieneExperiencia() {
    return cantidadDeInfectados > 3
  }

  method quiereInfectar() {
    return potencia > 500
  }
}


class VirusGordito inherits Virus {
  var peso = 20
  override method infectar(cuerpo) {
    peso += cuerpo.peso() / 100.0
  }

  method peso()  {
    return peso
  }
}


class VirusRechonchito inherits VirusGordito {
  override method quiereInfectar()  {
    return super() and self.peso() > 100
  }
}

class VirusConLiquido inherits Virus {
  var dosisLiquido = 20
  override method infectar(cuerpo) {
    super(cuerpo)
```

```
    dosisLiquido -= 1
  }

  method comerMiel() {
    dosisLiquido += 15
  }
}

class VirusConLiquidoDenso inherits VirusConLiquido {
  var densidad = 10
  override method infectar(cuerpo) {
    super(cuerpo)
    densidad -= 2
  }

  override method comerMiel() {
    super()
    densidad += 5
  }

  method estaRobusto() {
    return densidad > 10 and self.potencia() > 2000
  }

  override method quiereInfectar() {
    return super() and densidad > 4
  }
}


class VirusMusculoso inherits VirusConLiquidoDenso {
  override method potencia() {
    return super() + 500
  }
}

class Ornitorrinco {
  method peso() {
    return 40
  }
}
```

Listing 1: methodLookupVirus.wlk

```
test "ejemplos de virus" {
  var virus1 = new VirusGordito()
  var virus2 = new VirusRechonchito()
  var virus3 = new VirusConLiquido()
  var virus4 = new VirusConLiquidoDenso()
  var pepe = new Ornitorrinco()
}
```

Listing 2: methodLookupVirusTest.wtest

Se pide:

1. Indicar qué atributos tiene cada uno de los cuatro virus creados en el test.
2. Indicar cuáles de ellos cambian su valor si a cada uno le envío el mensaje `infectar(pepe)`.
3. Supongamos que `virus4` tiene 8 de densidad, 10 dosis de líquido y 300 de potencia. ¿Quiere infectar? Explicar por qué.

4. Agregamos este método en `VirusConLiquido`
   ```
   method quiereInfectar() = dosisLiquido > 5
   ```
   ¿cómo cambia la respuesta del punto anterior?

5. Para cada uno de estos mensajes, indicar cuáles de los cinco objetos los entienden

   a) `comerMiel`

   b) `quiereInfectar`

   c) `peso`

   d) `tieneExperiencia`

   e) `estaRobusto`

6. Con el código original, supongamos que la potencia de virus3 es 150 y de los otros tres es 3000, el peso de virus1 y virus2 es 40, y la densidad de virus4 es 35. Si le pregunto a cada uno si quiereInfectar, ¿qué me responde?

7. Con el código original, agreguemos esta variable al test
   ```
   var virus5 = new VirusMusculoso()
   ```
   y supongamos que tanto virus4 como virus5 tienen: 50 como valor del atributo densidad, y 1800 como valor del atributo potencia. Si les pregunto a virus4 y a virus5 si están robustos, ¿qué me responde en cada caso? Explicar.

8. Agregar las siguientes variantes de virus

   a) `VirusConLiquidoSabio`, es como los virus líquidos, con la única diferencia que si tiene más de 50 dosis de líquido entonces tiene experiencia, independientemente de la cantidad de infectados. Si no llega a 50 dosis, entonces sí corre la condición que trae de la clase Virus.

   b) `VirusFiaca`: nunca quiere infectar.

Ejercicio 2: Civilizaciones

El código que sigue define las clases `Civilizacion`, `CivilizacionComercial`, `CivilizacionComercialSuper`, `CivilizacionCultural`, `CivilizacionMilitar` y `CivilizacionPoetica`, además del objeto `juego`.

```
1  class Civilizacion {
2      var _puntajeCientifico
3      var _puntajeMilitar
4      var _puntajeEconomico
5
6      constructor(cien,mil,eco) {
7          _puntajeCientifico = cien
8          _puntajeMilitar = mil
9          _puntajeEconomico = eco
10     }
11
12     method puntaje(){
13       return self.puntajeCientifico()
14       + self.puntajeMilitar() + self.puntajeEconomico()
```

```
15      }
16
17      method puntajeCientifico() {
18        return _puntajeCientifico
19      }
20      method puntajeMilitar() {
21        return _puntajeMilitar
22      }
23      method puntajeEconomico() {
24        return  _puntajeEconomico
25      }
26  }
27
28  class CivilizacionComercial inherits Civilizacion {
29      var _cantTratadosComerciales
30      var _cantEscuelasContables
31
32      constructor(cien,mil,eco,tratadosCom,escuelasContables) =
33      super(cien,mil,eco) {
34          _cantTratadosComerciales = tratadosCom
35          _cantEscuelasContables = escuelasContables
36      }
37
38      override method puntajeEconomico() {
39        return super()
40          + self.cantTratadosComerciales() * self.baseComercio()
41        }
42
43      override method puntajeCientifico() {
44      return super()
45          + self.cantEscuelasContables() * self.baseComercio()
46      }
47      method baseComercio() {
48        return juego.baseComercio()
49      }
50
51      method cantTratadosComerciales() {
52        return _cantTratadosComerciales
53      }
54      method cantEscuelasContables() {
55        return _cantEscuelasContables
56      }
57  }
58
59
60  class CivilizacionComercialSuper inherits CivilizacionComercial {
61      constructor(cien,mil,eco,tratadosCom,escuelasContables) =
62          super(cien,mil,eco,tratadosCom,escuelasContables)
63
64      override method baseComercio() {
65        return super() * 2
66      }
67  }
68
69  class CivilizacionCultural inherits Civilizacion {
70      var _cantMuseos
71      var _cantUniversidades
72
73      constructor(cien,mil,eco,museos,univs) =
74      super(cien,mil,eco) {
75          _cantMuseos = museos
76          _cantUniversidades = univs
77      }
78
79      override method puntaje() {
80        return super() + self.puntajeCultural()
81      }
82
83      method puntajeCultural() {
84        return self.cantMuseos() * juego.baseCultural()
85      }
```

```
86
87
88      override method puntajeCientifico() {
89          return super()
90           + self.cantUniversidades() * juego.baseCultural()
91      }
92
93      method cantMuseos() {
94        return _cantMuseos
95      }
96
97      method cantUniversidades() {
98        return  _cantUniversidades
99      }
100 }
101
102 class CivilizacionMilitar inherits Civilizacion {
103     constructor(cien,mil,eco) = super(cien,mil,eco)
104
105     override method puntajeCientifico() {
106       return  0
107     }
108     override method puntajeEconomico() {
109       return 0
110     }
111 }
112
113 class CivilizacionPoetica inherits CivilizacionCultural {
114     var _nombreLiterario
115
116     constructor(cien,mil,eco,museos,univs,nombre) =
117     super(cien,mil,eco,museos,univs) {
118         _nombreLiterario = nombre
119     }
120
121     override method puntaje() {
122       return self.puntajeCultural()
123         + self.nombreLiterario().size() * 5
124     }
125
126     method nombreLiterario() {
127       return _nombreLiterario
128     }
129 }
130
131 object juego {
132     var _baseComercio = 0
133     var _baseCultural = 0
134
135
136     method baseComercio(unValor) {
137         _baseComercio = unValor
138     }
139
140     method baseComercio() {
141       return _baseComercio
142     }
143
144     method baseCultural(unValor) {
145         _baseCultural = unValor
146     }
147
148     method baseCultural() {
149       return  _baseCultural
150     }
151 }
```

Listing 3: methodLookupCiv.wlk

Se pide:

a)
Armar el diagrama de clases que incluya las 6 clases definidas, indicando la herencia y qué métodos se definen en cada una.

b)
Indicar qué valores hay que poner en reemplazo de
`fenT, fenC, carT, carC, milT, milC, espT, espC, delT, delC`
para que el siguiente test dé verde.

```
test "puntajes" {
    juego.baseComercio(5)
    juego.baseCultural(8)

    var fenicia = new CivilizacionComercial(
      20, 15, 100,     // cientifico, militar, economico
      5, 6             // tratadosComerciales, escuelasContables
    )

    var cartago = new CivilizacionComercialSuper(
      20, 35, 100,     // cientifico, militar, economico
      10, 4            // tratadosComerciales, escuelasContables
    )

  var mileto = new CivilizacionCultural (
      40, 50, 20,      // cientifico, militar, economico
      6, 9             // museos, universidades
    )


  var esparta = new CivilizacionMilitar (
      5, 200, 45       // cientifico, militar, economico
    )

  var delfos = new CivilizacionPoetica (
      60, 2, 20,       // cientifico, militar, economico
      10, 2,           // museos, universidades
      "Delfos la bella"    // nombre, tiene 15 letras
    )

    assert.equals(/* fenT */, fenicia.puntaje())
    assert.equals(/* fenC */, fenicia.puntajeCientifico())
    assert.equals(/* carT */, cartago.puntaje())
    assert.equals(/* carC */, cartago.puntajeCientifico())
    assert.equals(/* milT */, mileto.puntaje())
    assert.equals(/* milC */, mileto.puntajeCientifico())
    assert.equals(/* espT */, esparta.puntaje())
    assert.equals(/* espC */, esparta.puntajeCientifico())
    assert.equals(/* delT */, delfos.puntaje())
    assert.equals(/* delC */, delfos.puntajeCientifico())
}
```

Listing 4: methodLookupCivTest.wtest

c)
Indicar cómo implementar la clase `CivilizacionMilitar` si se quiere preservar los puntajes científico y económico, pero que no influyan en el puntaje total. P.ej. si queremos que en el test anterior, el puntaje total de Esparta sea el mismo que se calculó, pero el puntaje científico sea 5 y el económico 45.

Respuestas del punto b)

Fenicia	total	190
Fenicia	científico	50
Cartago	total	295
Cartago	científico	60
Mileto	total	230
Mileto	científico	112
Esparta	total	200
Esparta	científico	0
Delfos	total	155
Delfos	científico	76

Ejercicio 3: El escritor automático

El código que sigue define las clases `YoAprendi`, `YoAprendiConfigurable`, `YoFui`, `YoFuiDeportivo`, `YoAprendiEnEscuela`, `AprendiBaile`, `AprendiBienBaile`, `AprendiSalsa`, `AprendiSalsaLunes` y `Escuela`.

```
class YoAprendi {
    method accion()
    method objeto()
    method final()
    method encabezado()  {
      return "yo aprendi a"
    }

    method frase() {
      return self.encabezado() + " "
        + self.accion() + " "
        + self.objeto() + " "
        + self.final()
    }
}

class YoAprendiConfigurable inherits YoAprendi {
    var _accion
    var _objeto
    var _final

    constructor(accion,objeto,final) {
        _accion = accion
        _objeto = objeto
        _final = final
    }

    override method accion()  { return _accion }
    override method objeto() { return _objeto }
    override method final() { return _final }
}

class YoFui inherits YoAprendiConfigurable {
    constructor(accion,objeto,final) =
        super(accion,objeto,final)

    override method encabezado() {  return "yo fui a" }
}

class YoFuiDeportivo inherits YoFui {
    constructor(accion,objeto,final) =
        super(accion,objeto,final)

```

```
44      override method accion() { return  super() + " con energia" }
45      override method final() { return super() + ", y me gusto" }
46  }
47
48  class YoAprendiEnEscuela inherits YoAprendi {
49      var _escuela
50      constructor(escuela) { _escuela = escuela }
51
52      override method accion() {
53        return self.escuela().queEnseniaAHacer()
54      }
55      override method objeto() { return self.escuela().objeto() }
56      override method encabezado() { return "les cuento que" }
57      override method final() {
58        return "en la escuela " + self.escuela().nombre()
59      }
60      method escuela() = { return _escuela }
61  }
62  class AprendiBaile inherits YoAprendi {
63      override method accion() { return "bailar" }
64  }
65
66  class AprendiBienBaile inherits AprendiBaile {
67      override method accion() {
68        return super() + " con precision"
69      }
70  }
71
72  class AprendiSalsa inherits AprendiBienBaile {
73      var _cuando
74      constructor(cuando) { _cuando = cuando }
75
76      override method objeto() { return "salsa" }
77      override method final() {
78        return self.cuando() + ", y me fue bien"
79      }
80      method cuando() { return _cuando }
81  }
82
83  class AprendiSalsaLunes inherits AprendiSalsa {
84      constructor(cuando) = super(cuando)
85
86      override method cuando() { return super() + " los lunes" }
87  }
88
89  class Escuela {
90      var _nombre
91      var _queEnseniaAHacer
92      var _objeto
93
94      constructor(nombre,queEnsenia,objeto) {
95          _nombre = nombre
96          _queEnseniaAHacer = queEnsenia
97          _objeto = objeto
98      }
99
100     method nombre() { return _nombre }
101     method queEnseniaAHacer() { return _queEnseniaAHacer }
102     method objeto() { return _objeto }
103 }
```

Listing 5: methodLookupFrases.wlk

Se pide:

a)
Armar el diagrama de clases que incluya las 10 clases definidas, indicando la herencia y qué métodos se definen en cada una, y además, cuáles de estas clases son abstractas.

b)
Indicar qué valores hay que poner en reemplazo de
`sui, nadar, nadarC, ing, salsi, salsiL`
para que el siguiente test dé verde.

```
test "frases" {
    var icana = new Escuela(
      "ICANA", "hablar", "ingles"      // nombre, accion, objeto
    )

    var sui = new YoAprendiConfigurable(
      "ser", "formal y cortes",               // accion, objeto
      "cortandome el pelo una vez por mes"    // final
    )

    var nadar = new YoFui(
      "nadar", "crawl", "al rio"     // accion, objeto, final
    )

    var nadarCopado = new YoFuiDeportivo(
      "nadar", "crawl", "al rio"     // accion, objeto, final
    )

    var inglish = new YoAprendiEnEscuela(icana)     // escuela

    var salsita = new AprendiSalsa(
      "el mes pasado"     // cuando
    )

    var salsitaLunes = new AprendiSalsaLunes (
      "el mes pasado"     // cuando
    )

    assert.equals(/* sui */, sui.frase())
    assert.equals(/* nadar */, nadar.frase())
    assert.equals(/* nadarC */, nadarCopado.frase())
    assert.equals(/* ing */, inglish.frase())
    assert.equals(/* salsi */, salsita.frase())
    assert.equals(/* salsiL */, salsitaLunes.frase())
}
```

Listing 6: methodLookupFrasesTest.wtest

c)
Definir la clase `AprendiTango`, que a partir de una acción complementaria y un lugar, arme la frase
"Yo aprendi a bailar y ⟨⟨acción complementaria⟩⟩ tango en ⟨⟨lugar⟩⟩."

P.ej. si definimos como acción complementaria "disfrutar" y como lugar "El Beso", la frase queda:
"Yo aprendi a bailar y disfrutar tango en El Beso."

Armar `AprendiTango` a partir de las clases que están implementadas, separando en acción (que incluye siempre "bailar y" más la complementaria), objeto (que es tango), y final (que es el lugar con "en" adelante).

Ejercicio 4: Trace

A partir de las siguientes clases:

```
class A {

  method m1() { return "A.m1" }

  method m2() { return "A.m2" }

  method m3() { return "A.m3" }
}

class B inherits A {

  override method m1() { return "B.m1 " + super() }

  override method m2() { return "B.m2 " + self.m3() }

  method m4() { return "B.m4" }
}

class C inherits B {

  override method m1() { return "C.m1" }

  override method m2() {
    return "C.m2 " + super() + " " + self.m4()
  }

  override method m3() { return "C.m3" }
}
```

Listing 7: methodLookupTrace.wlk

Indicar que devuelve cada línea de código:

```
  var o = new A()
  o.m1()
  o.m2()
  o.m3()
  o.m4()

  o = new B()
  o.m1()
  o.m2()
  o.m3()
  o.m4()

  o = new C()
  o.m1()
  o.m2()
  o.m3()
  o.m4()
```

Listing 8: methodLookupTrace.wtest

Ejercicio 5: Trace 2

Dadas las siguientes clases

```
class Abuela {

  method m1() {
    return "A.m1 " + self.m2() + " " + self.m3()
  }

  method m2() {
    return "A.m2"
  }
  //abstracto
  method m3()
}

class Madre inherits Abuela {
  override method m2() {
    return "M.m2 " + super() + " " + self.m3()
  }

  override method m3() {
    return "M.m3"
  }
}

class Hija inherits Madre {

  override method m1() {
    return "H.m1 " + self.m3() + " " + super()
  }

}
```

Listing 9: methodLookupTrace2.wlk

Indique que devuelve cada línea de código:

```
    var p

  p = new Madre()
  p.m1()
  p.m2()
  p.m3()

  p = new Hija()
  p.m1()
  p.m2()
  p.m3()
```

Listing 10: methodLookupTrace2.wpgm

Ejercicio 6: Gritos

A partir de las siguientes clases:

```
class Gritador {
  method gritar() {
    return self.gritito() + self.granGrito() + self.final()
  }

  method gritito() { return "uy" }

  method granGrito() { return "UY" }

  method final() { return "!!" }
}

class Gritador1 inherits Gritador{
  override method gritito() { return "ay" }

  override method granGrito() {
    return self.gritito() + "AAH"
  }
}

class Gritador2 inherits Gritador1 {

    override method gritito() { return "aa" + super() }

    override method final() {
      return self.granGrito() + super()
    }

    override method gritar() {
      return self.final() + self.granGrito() + self.gritito()
    }
}

class Gritador3 inherits Gritador2 {

  override method granGrito() { return "EEE" }

  override method final() { return self.gritito() }
}
```

Listing 11: methodLookupGritos.wlk

Indicar que devuelve cada línea de código:

```
  new Gritador().gritar()
  new Gritador1().gritar()
  new Gritador2().gritar()
  new Gritador3().gritar()
```

Listing 12: methodLookupGritos.wpgm

De este te damos las **respuestas**

Gritador	uyUY!!
Gritador1	ayayAAH!!
Gritador2	aaayAAH!!aaayAAHaaay
Gritador3	aaayEEEaaay

Ejercicio 7: Vikingos

Dadas las siguientes clases

```
class Vikingo {

  method accionesDeSaqueo() {
    return self.accionesTesoros() + self.accionesEnemigos()
  }
  method accionesTesoros() {
    return ["saquear tesoros", "fundir oro"]
  }
  method accionesEnemigos() { return ["matar guerreros"] }
}
class Conde inherits Vikingo {

  override method accionesEnemigos () {
    return super() + ["matar familia"]
  }
  override method accionesDeSaqueo() {
    return super() + self.accionesLugar()
  }
  method accionesLugar() { return ["incendiar lugar"] }
}
class Rey inherits Conde{

  override method accionesEnemigos() {
    return ["reclutar jefes vencidos"] + super()
  }
  override method accionesLugar() {
    return super() + ["echar sal", "arar"]
  }
  override method accionesTesoros() { return ["dividir tesoros"]}
}
class Profeta inherits Vikingo {

  override method accionesTesoros() { return [] }

  override method accionesEnemigos() {
    return ["predecir futuro de enemigos"]
  }
}
```

Listing 13: methodLookupVikingos.wlk

Indique que devuelve cada línea de código:

```
  new Vikingo().accionesDeSaqueo()
  new Conde().accionesDeSaqueo()
  new Rey().accionesDeSaqueo()
  new Profeta().accionesDeSaqueo()
```

Listing 14: methodLookupVikingos.wpgm

Nota: Recordar que las listas entienden el mensaje + para concatenar. El resultado es una nueva lista. P.ej. el resultado de
`["alfa","beta"] + ["gamma","iota","epsilon"] + [] + ["omega"]` es
`["alfa","beta","gamma","iota","epsilon","omega"]`

Ejercicio 8: Skills

Dadas las siguientes clases

```
class Junior {
  method actividades() {
    return self.burocracia() + self.programacion()
  }
  method burocracia() {
    return ["cargar horas"]
  }
  method programacion() {
    return ["resolver issues"]
  }
}
class SemiSenior inherits Junior {
  override method programacion () {
    return super() + ["detectar issues"]
  }
  override method actividades() {
    return super() + self.planificacion()
  }
  method planificacion() {
    return ["estimar issues"]
  }
}
class Senior inherits SemiSenior {
  override method programacion() {
    return ["refactorizar arquitectura"] + super()
  }
  override method planificacion() {
    return ["coordinar reunion"] + self.demos()
  }
  method demos() {
    return ["coordinar demo"]
  }
  override method burocracia() {
    return []
  }
}
class Ninja inherits Senior {
  override method programacion() {
    return ["resolver todo"]
  }
  override method demos() {
    return ["salvar demo"]
  }
}
class Acomodado inherits Junior { //OJO!! Mira bien de quien hereda!!
  override method programacion() {
    return []
  }
  override method burocracia() {
    return ["tomar cafe"] + super()
  }
}
```

Listing 15: methodLookupSkills.wlk

Indique que devuelve cada línea de código:

```
  new Junior().actividades()
  new SemiSenior().actividades()
  new Senior().actividades()
  new Ninja().actividades()
  new Acomodado().actividades()
```

Listing 16: methodLookupSkills.wpgm

Ejercicio 9: Jedi

Dadas las siguientes clases

```
class Personaje {

  method habilidades() {
    return self.batalla() + self.extras()
  }
  method batalla()  {
    return ["disparar laser"]
  }
  method extras() {
    return ["programar droide"]
  }
}
class Padawan inherits Personaje {

  override method batalla() {
    return ["usar sable laser"]
  }
  override method extras() {
    return ["sentir la fuerza"] + super()
  }
}
class Jedi inherits Padawan {

  override method habilidades() {
    return super() + self.estrategia()
  }
  method estrategia() {
    return ["comandar clones"]
  }
  override method extras() {
    return ["construir sable", "mover objetos"] + super()
  }
}
class MaestroJedi inherits Jedi {

  override method estrategia() {
    return self.jedis() + super()
  }
  method jedis() {
    return ["comandar jedis"]
  }
}
class LordSith inherits MaestroJedi {

  override method habilidades() {
    return self.estrategia() + ["usar lado oscuro"]
  }
  override method jedis() {
    return ["matar jedis"]
  }
}
```

Listing 17: methodLookupJedi.wlk

Indique que devuelve cada línea de código:

```
  new Personaje().habilidades()
  new Padawan().habilidades()
  new Jedi().habilidades()
  new MaestroJedi().habilidades()
  new LordSith().habilidades()
```

Listing 18: methodLookupJedi.wpgm

Ejercicio 10: Soldados

Dadas las siguientes clases

```
class Soldado {

  method habilidades() {
    return self.batalla() + self.extras()
  }
  method batalla()  {
    return ["fusil"]
  }
  method extras() {
    return ["limpiar"]
  }
}
class Cabo inherits Soldado {

  override method batalla() {
    return ["artilleria"] + super()
  }
  override method extras() {
    return ["fumar"]
  }
}
class Sargento inherits Cabo {

  override method habilidades() {
    return super() + self.estrategia()
  }
  method estrategia() {
    return ["comandar soldados"]
  }
  override method extras() {
    return ["poker"]  + super()
  }
}
class Teniente inherits Sargento {

  override method estrategia() {
    return self.politica() + super()
  }
  method politica() {
    return ["contentar al superior"]
  }
}
class Coronel inherits Teniente {

  override method habilidades() {
    return self.estrategia() + ["planificar"]
  }
  override method politica() {
    return ["negociar"]
  }
}
```

Listing 19: methodLookupSoldados.wlk

Indique que devuelve cada línea de código:

```
  new Soldado().habilidades()
  new Cabo().habilidades()
  new Sargento().habilidades()
  new Teniente().habilidades()
  new Coronel().habilidades()
```

Listing 20: methodLookupSoldados.wpgm

2. Polimorfismo

Ejercicio 11: Exámenes

Reorganice y modifique el siguiente código utilizando polimorfismo para mejorar la distribución de responsabilidades. Se permite (de hecho, se alienta a) agregar más clases a las planteadas, pasando parte del código a las nuevas clases.

```
class Alumno {
  method puntaje(examen) {
    var puntos = 0
    examen.preguntas().forEach({ preg =>
      if (preg.tipo() == "Exacta") {
        if (preg.lePegoJusto()) {
          puntos += preg.puntajeQueOtorga()
        }
      }
      if (preg.tipo() == "Aproximada") {
        if (preg.lePegoJusto()) {
          puntos += preg.puntajeQueOtorga()
        }
        if (preg.respuestaCorrecta() == (preg.respuesta() - 1)) {
          puntos += preg.puntajeQueOtorga() - 2
        }
        if (preg.respuestaCorrecta() == (preg.respuesta() + 1)) {
          puntos += preg.puntajeQueOtorga() - 2
        }
      }
      // en las preguntas de este tipo, la respuesta
      // es una lista ordenada
      if (preg.tipo() == "Listas") {
        if (preg.lePegoJusto()) {
          puntos += preg.puntajeQueOtorga()
        } else {
          if (preg.respuestaCorrecta().asSet() ==
              preg.respuesta().asSet()) {
            puntos += preg.puntajeQueOtorga() - 5
          }
        }
      }
    })
    if (examen.preguntas().all({preg => preg.contesto()})) {
      puntos += 10
    }
    return puntos
  }
}

class Examen {
  var preguntas

  constructor(_preguntas) {preguntas = _preguntas}

  method preguntas() = preguntas
}

class Pregunta {
  var respuestaCorrecta
  var respuesta
  var puntajeQueOtorga
  var tipo

  constructor(_respuestaCorrecta, _respuesta, _puntajeQueOtorga, _tipo) {
    respuestaCorrecta = _respuestaCorrecta
    respuesta = _respuesta
    puntajeQueOtorga = _puntajeQueOtorga
    tipo = _tipo
```

```
60    }
61
62    method respuestaCorrecta()   {
63      return respuestaCorrecta
64    }
65    method respuesta() {
66      return respuesta
67    }
68    method puntajeQueOtorga()  {
69      return puntajeQueOtorga
70    }
71
72    method lePegoJusto() {
73      return self.respuestaCorrecta() == self.respuesta()
74    }
75
76    method tipo() {
77      return tipo
78    }
79    method contesto() {
80      return respuesta != null
81    }
82
83  }
```

Listing 21: polimorfismoExamen.wlk

Ejercicio 12: Renderización de una pantalla

Reorganice y modifique el siguiente código utilizando polimorfismo para mejorar la distribución de responsabilidades. Se permite (de hecho, se alienta a) agregar más clases a las planteadas, pasando parte del código a las nuevas clases.

Tener en cuenta las siguientes aclaraciones:

- Los métodos no desarrollados en la clase `Pantalla` indican que una pantalla sabe trazar una línea en un color a partir de la posición de un cursor, y también mover el cursor. La implementación de estas operaciones no está entre el código que debe reorganizar.
- Obsérvese que para dibujar una figura, antes de empezar hay que prender el cursor, y lo último que hay que hacer es apagarlo. Estas también son operaciones responsabilidad de la pantalla, y que no están incluidas en la reorganización de código.
- Nótese también que cada figura debe ser dibujada en rojo y en verde, esto es cierto tanto para las figuras que han sido implementadas, como para las que pudieran agregarse. También, hay que tener en cuenta que después de dibujar una figura, el cursor debe estar en el mismo lugar en el que estaba antes de empezar a dibujarla.

```
1  class Pantalla {
2    method dibujar(figura) {
3      self.prenderCursor()
4      if (figura.tipo() == "lineaHorizontal") {
5        // en rojo
6        self.lineaDerecha(figura.tamanio(), colorRojo)
7        self.moverCursorIzquierda(figura.tamanio())
8        // en verde
```

```
9         self.lineaDerecha(figura.tamanio(), colorVerde)
10        self.moverCursorIzquierda(figura.tamanio())
11      }
12      if (figura.tipo() == "ele") {
13        // en rojo
14        self.lineaAbajo(figura.tamanio(),colorRojo)
15        self.lineaDerecha(figura.tamanio(), colorRojo)
16        self.moverCursorIzquierda(figura.tamanio())
17        self.moverCursorArriba(figura.tamanio())
18        // en verde
19        self.lineaAbajo(figura.tamanio(),colorVerde)
20        self.lineaDerecha(figura.tamanio(), colorVerde)
21        self.moverCursorIzquierda(figura.tamanio())
22        self.moverCursorArriba(figura.tamanio())
23      }
24      if (figura.tipo() == "cuadrado") {
25        // en rojo
26        self.lineaAbajo(figura.tamanio(),colorRojo)
27        self.lineaDerecha(figura.tamanio(), colorRojo)
28        self.lineaArriba(figura.tamanio(),colorRojo)
29        self.lineaIzquierda(figura.tamanio(),colorRojo)
30        // en verde
31        self.lineaAbajo(figura.tamanio(),colorVerde)
32        self.lineaDerecha(figura.tamanio(), colorVerde)
33        self.lineaArriba(figura.tamanio(),colorVerde)
34        self.lineaIzquierda(figura.tamanio(),colorVerde)
35      }
36
37
38      if (figura.tipo() == "dobleLineaHorizontal") {
39        // en rojo
40        self.lineaDerecha(figura.tamanio(), colorRojo)
41        self.moverCursorIzquierda(figura.tamanio())
42        self.moverCursorArriba(1)
43        self.lineaDerecha(figura.tamanio(), colorRojo)
44        self.moverCursorIzquierda(figura.tamanio())
45        self.moverCursorAbajo(1)
46        // en verde
47        self.lineaDerecha(figura.tamanio(), colorVerde)
48        self.moverCursorIzquierda(figura.tamanio())
49        self.moverCursorArriba(1)
50        self.lineaDerecha(figura.tamanio(), colorVerde)
51        self.moverCursorIzquierda(figura.tamanio())
52        self.moverCursorAbajo(1)
53      }
54      self.apagarCursor()
55    }
56
57    method lineaAbajo(tamanio,color) {
58      //implementacion que no interesa ...
59    }
60    method lineaDerecha(tamanio,color) {
61      //implementacion que no interesa ...
62    }
63    method lineaArriba(tamanio,color) {
64      //implementacion que no interesa ...
65    }
66    method lineaIzquierda(tamanio,color) {
67      //implementacion que no interesa ...
68    }
69    method moverCursorDerecha(cuanto) {
70      //implementacion que no interesa ...
71    }
72
73    method moverCursorAbajo(cuanto) {
74      //implementacion que no interesa ...
75    }
76    method moverCursorIzquierda(cuanto) {
77      //implementacion que no interesa ...
78    }
79    method moverCursorArriba(cuanto) {
```

```
80      //implementacion que no interesa ...
81    }
82    method prenderCursor() {
83      //implementacion que no interesa ...
84    }
85    method apagarCursor() {
86      //implementacion que no interesa ...
87    }
88
89  }
90
91  object colorRojo {
92  }
93
94  object colorVerde {
95  }
96
97  class Figura {
98    var tamanio
99    var tipo
100
101   constructor(_tipo, _tamanio) {
102     tipo = _tipo
103     tamanio = _tamanio
104   }
105   method tamanio() { return tamanio }
106   method tipo() { return tipo }
107 }
```

Listing 22: polimorfismoPantalla.wlk

Ejercicio 13: Movimientos del ajedrez

Reorganice y modifique el siguiente código utilizando polimorfismo para mejorar la distribución de responsabilidades. Se permite (de hecho, se alienta a) agregar más clases a las planteadas, pasando parte del código a las nuevas clases.

El objetivo de este código es saber si una pieza de ajedrez puede moverse a una determinada posición. Las condiciones están simplificadas respecto de las reglas verdaderas del juego.

```
1   class TableroDeAjedrez {
2     method puedeMover(pieza, fila, columna) {
3       var puede
4       if (pieza.esPeon()) {
5         puede = (pieza.columna() == columna)
6                 and (pieza.fila() + 1 == fila)
7       }
8       if (pieza.esTorre()) {
9         puede = (pieza.columna() == columna)
10                or (pieza.fila() == fila)
11      }
12      if (pieza.esRey()) {
13        puede = ((pieza.columna() - columna).abs() <= 1)
14                and ((pieza.fila() - fila).abs() <= 1)
15      }
16      if (pieza.esAlfil()) {
17        puede = (pieza.columna() - columna).abs()
18                == (pieza.fila() - fila).abs()
19      }
20      return puede
21          and not (pieza.estaEn(fila,columna))
22          and self.esPosicion(fila,columna)
23    }
24
25    method esPosicion(fila,columna) {
```

```
26      return fila.between(1,8) and columna.between(1,8)
27    }
28  }
29
30  class Pieza {
31    var fila
32    var columna
33    var tipo
34
35    method estaEn(f,c) {
36      return f == fila and c == columna
37    }
38
39    method fila() = fila
40    method columna() = columna
41
42    method esRey() { return tipo == 'Rey' }
43    method esTorre() { return tipo == 'Torre' }
44    method esAlfil() { return tipo == 'Alfil' }
45    method esPeon() { return tipo == 'Peon' }
46  }
```

Listing 23: polimorfismoAjedrez.wlk

Ejercicio 14: Ensamblador

Hace ya unos años que Volkswagen Argentina fabrica en su planta de Pacheco los modelos: Fox y Amarok (en sus dos versiones: volante derecho y volante izquierdo). Se está pensando fabricarlos en la misma planta el modelo Vento a la línea de ensamblado, con lo cual nos pidieron modificar el sistema.

Revisando el código, encontramos lo siguiente:

```
1   object lineaDeEnsamblado {
2     method ensamblar(unAuto, unColor) {
3       if (unAuto.modelo() == "Fox") {
4         unAuto.ponerPuertasDelanteras()
5         unAuto.ponerRuedas(4)
6         unAuto.ponerRuedaAuxilio()
7         unAuto.ponerPuertasTraseras()
8         unAuto.ponerVolanteAIzquierda()
9       }
10      if (unAuto.modelo() == "Amarok Izq") {
11        unAuto.ponerPuertasDelanteras()
12        unAuto.ponerRuedas(4)
13        unAuto.ponerRuedaAuxilio()
14        unAuto.ponerCajuela()
15        unAuto.ponerVolanteAIzquierda()
16      }
17      if (unAuto.modelo() == "Amarok Der") {
18        unAuto.ponerPuertasDelanteras()
19        unAuto.ponerRuedas(4)
20        unAuto.ponerRuedaAuxilio()
21        unAuto.ponerCajuela()
22        unAuto.ponerVolanteADerecha()
23      }
24      unAuto.pintarDe(unColor)
25    }
26  }
27
28  class Auto {
29    var modelo //Un string que indica el modelo del auto
30    var color //Un string que indica el color del auto
31    var cosas = #{ } /* todas las cosas (ruedas, cajuela, etc,) que se
32              * ponen son objetos  que se agregan a la colección*/
33
34    constructor(unModelo) {
35      modelo = unModelo
```

```
36    }
37
38    method modelo() {
39      return modelo
40    }
41
42    method pintarDe(unColor) {
43      color = unColor
44    }
45
46    method color() {
47      return color
48    }
49    /*
50    * Todos los métodos poner son parecidos, se escribe la implementación
51    * de uno para tener una idea
52    */
53    method ponerPuertasDelanteras() {
54      cosas.add( new PuertasDelanteras())
55    }
56    method ponerPuertasTraseras() {
57      ...
58    }
59    method ponerRuedaAuxilio () {
60      ...
61    }
62    method ponerRuedas(unaCantidad) {
63      ...
64    }
65    method ponerCajuela(unaCantidad) {
66      ...
67    }
68    method ponerVolanteAIzquierda() {
69      ...
70    }
71
72    method ponerVolanteADerecha() {
73      ...
74    }
75  }
76
77  test "construir un Amarok con Volante a la izquierda" {
78
79      var amarok = new Auto("Amarok Izq")
80      lineaDeEnsamblado.ensamblar(amarok, "azul")
81      //solo interesa la parte del test donde se muestra la creación
82      ...
83  }
```

Listing 24: polimorfismoAutos.wlk

Se pide:

1. Criticar la solución propuesta (Especial atención en la repetición de código, la distribución de responsabilidades y el uso o ausencia de polimorfismo.

2. Indicar qué se debe modificar en este código para agregar el modelo Vento. Escribir cómo sería la línea que construye un auto Vento, es decir, a qué clase se le hace *new* y en caso de recibir parámetro/s en el constructor, cuál/es es/son.

3. Refactorizar el código para solucionar los problemas detectados en a. ¡Vale agregar nuevas clases y mensajes!. Indicar cómo quedarían las primeras dos líneas del test.

Ejercicio 15: Centrales eléctricas

En un modelo del sistema de producción y transmisión eléctrica de un país, tenemos las clases `SistemaInterconectado`, `CentralHidrica`, `CentralAtomica` y `CentralACarbon`. Un sistema inteconectado incluye varias centrales. La capacidad de generación de un sistema inteconectado es la suma de la capacidad de generación de cada una de sus centrales. Para calcularla se agregó este método

```
1  class SistemaInterconectado {
2    var centrales = #{ }
3
4    method capacidadDeGeneracion() {
5      var total = 0
6      centrales.forEach({ central =>
7          if (central.esHidrica()) {
8            total = total + central.cantTurbinas() *
9                    central.potenciaMaxTurbina() *
10                   central.caudalActual().max(central.caudadDeSaturacion()) /
11                   central.caudadDeSaturacion()
12         }
13         if (central.esAtomica()) {
14           central.sectores().forEach({ sector =>
15             total = total + sector.capacidadDeGeneracion() })
16         } if (central.esACarbon()) {
17           total = total + central.capacidadInstalada()
18         }
19     return total
20     })
21   }
22 }
```

Listing 25: polimorfismoCentrales.wlk

Se pide:

- Suponiendo que las centrales hídricas son las únicas que responden `true` cuando se les pregunta `esHidrica()` y análogamente para las otras, indicar a través de un diagrama de clase qué mensajes entienden los distintos tipos de central según lo que se puede apreciar en el método `capacidadDeGeneracion()` de la clase `SistemaInterconectado`
- Para otro sistema que está haciendo el mismo grupo de desarrolladores se deben modelar plantas de aluminio. Una planta de aluminio tiene una única central generadora de energía (que puede ser hídrica, atómica o a carbón) de la cual se va a necesitar saber su capacidad de generación. Supongamos que las centrales solamente entienden los mensajes que se utilizan en el método `capacidadDeGeneracion()` de la clase `SistemaInterconectado`. Hay un problema de asignación de responsabilidades que impide usar el modelo de centrales eléctricas para la planta de aluminio. Indicar cuál es el problema (o sea, qué objeto está haciendo una cosa que no le corresponde) y cómo corregirlo (o sea, qué método/s habría que agregar o modificar en qué clase/s, escribiendo el código correspondiente).
- Qué concepto se está utilizando en su solución del punto b que no se estaba aprovechando en la solución propuesta por el enunciado?

Ejercicio 16: Comedero

a) Analice el siguiente código y escriba un breve párrafo acerca de la distribución de responsabilidades entre los objetos/clases involucradas.

b) Modifique el código utilizando polimorfismo para mejorar la distribución de responsabilidades.

```
object comedero {
  var deposito = new Deposito()

  method alimentar(unAnimal) {
    if (unAnimal.esVaca()) {
      deposito.descontarPasto(1)
      unAnimal.darPasto(1)

      deposito.descontarAlfalfa(2)
      unAnimal.darAlfalfa(2)

      deposito.descontarVitaminas(3)
      unAnimal.darVitaminas(3)

      deposito.descontarAgua(2)
      unAnimal.darAgua(2)
    }
    if (unAnimal.esCaballo()) {
      deposito.descontarPasto(1)
      unAnimal.darPasto(1)

      deposito.descontarAlfalfa(2)
      unAnimal.darAlfalfa(2)

      //los caballos necesitan vitaminas distintas
      //con respecto a las vacas
      deposito.descontarVitaminas(4)
      unAnimal.darVitaminas(4)

      deposito.descontarAgua(2)
      unAnimal.darAgua(2)
    }

    if (unAnimal.esPerro()) {
      deposito.descontarCarne(0.5)
      unAnimal.darCarne(0.5)

      deposito.descontarAgua(2)
      unAnimal.darAgua(2)
    }
  }
}

class Animal {
  var tipoDeAnimal
  var aguaConsumida = 0
  var energia = 100
  var vitaminasConsumidas = 0

  constructor(_tipo) {
    tipoDeAnimal = _tipo
  }
  //Mensajes/Metodos para saber que animal es
  //solo usados por el comedero
  method esVaca() { return tipoDeAnimal == "vaca" }
  method esCaballo() { return tipoDeAnimal == "caballo" }
  method esPerro() { return tipoDeAnimal == "perro" }

  //Mensaje/Metodo que interesa ser consultado
```

```
60    //para cualquier animal
61    method estaContento() {
62      return ( aguaConsumida * 10 / energia > 1 )
63      or ( aguaConsumida > 10 and
64      energia > 200 and vitaminasConsumidas > 0 )
65    }
66
67    /*metodos relacionados con la alimentacion */
68    method darAgua(litros) { aguaConsumida += litros }
69    method darVitaminas(cantidad) { vitaminasConsumidas += cantidad }
70
71    method darCarne(kilos) {
72      if (not self.esPerro()) {
73        error.throwWithMessage(
74        "Solo los carnivoros pueden consumir carne")
75      }
76      energia += kilos * 2
77    }
78
79    method darAlfalfa(kilos) {
80      if (not self.esCaballo() or self.esVaca()) {
81        error.throwWithMessage("
82        Solo el ganado pueden consumir alfalfa")
83      }
84      energia += kilos
85    }
86
87    method darPasto(kilos) {
88      if (not self.esCaballo() or self.esVaca()) {
89        error.throwWithMessage(
90        "Solo el ganado pueden consumir pasto")
91      }
92      energia += kilos * 0.1
93    }
94  }
95
96  //Desde aca para abajo asumir que es todo correcto
97  //Las clases Deposito y Alacena
98  //estan escritas para contextualizar el codigo anterior
99
100 class Deposito {
101   var litrosDisponiblesAgua = new Alacena(1000)
102   var kilosDisponiblesAlfalfa = new Alacena(500)
103   var kilosDisponiblesPasto = new Alacena(500)
104   var kilosDisponiblesCarne = new Alacena(100)
105   var unidadesDisponiblesVitaminas = new Alacena(200)
106
107   method descontarPasto(kilos) { kilosDisponiblesPasto.decrementar(kilos) }
108
109   method descontarAlfalfa(kilos) {
110     kilosDisponiblesAlfalfa.decrementar(kilos)
111   }
112
113   method descontarAgua(litros) { litrosDisponiblesAgua.decrementar(litros) }
114   method descontarCarne(kilos) { kilosDisponiblesCarne.decrementar(kilos) }
115   method descontarVitaminas(unidades) {
116     unidadesDisponiblesVitaminas.decrementar(unidades)
117   }
118 }
119 class Alacena {
120   var disponible
121
122   constructor(_disponible) {
123     disponible = _disponible
124   }
125
126   method decrementar(unaCantidad) {
127     if (unaCantidad > disponible) {
128       error.throwWithMessage(
129       "No hay " + unaCantidad +
130       " disponible. Maximo: " +
```

```
131       disponible)
132     }
133     disponible -= unaCantidad
134   }
135 }
```

Listing 26: polimorfismoComedero.wlk

Ejercicio 17: Misiones

Modifique el código utilizando polimorfismo, organización del comportamiento entre las clases, y (en un caso) mensajes adecuados a las colecciones, para mejorar la distribución de responsabilidades. Evite el código duplicado.

```
1  class Mision {
2
3    var objetivo //Es una instancia de la clase
4        //Objetivo u ObjetivoMultiple)
5    var mundo  // Es una instancia de la clase Mundo
6    constructor(_objetivo, _mundo) {
7      objetivo = _objetivo
8      mundo = _mundo
9    }
10
11   method esExitosa() {
12     if(objetivo.esSuperviciencia()) {
13       return not mundo.destruido()
14           and mundo.tiempoActual() >= objetivo.tiempoDeSupervivencia()
15           and mundo.personajeEstaVivo()
16     }
17     if(objetivo.esParaDerrotarEnemigos()) {
18       return not mundo.destruido()
19           and mundo.enemigosDerrotados() >= objetivo.enemigosADerrotar()
20     }
21     if(objetivo.esParaDefenderAliados()) {
22       return not mundo.destruido()
23           and mundo.aliadosCaidos() <= objetivo.aliadosPrescindibles()
24     }
25     if(objetivo.esMultiple()) {
26       return self.resolverExitoMultiple(objetivo)
27     }
28     error.throwWithMessage("Tipo de objetivo desconocido")
29     return false
30   }
31   //metodo auxiliar que devuelve true
32   //si se cumplieron TODOS los objetivos del objetivoMultiple
33   method resolverExitoMultiple(objetivoMultiple) {
34     var valorRetorno = true
35     objetivoMultiple.objetivosInternos().forEach({unObjetivo =>
36       if(unObjetivo.esSuperviciencia()) {
37         valorRetorno = valorRetorno
38           and not mundo.destruido()
39           and mundo.tiempoActual() >= objetivo.tiempoDeSupervivencia()
40           and mundo.personajeEstaVivo()
41       }
42       if(unObjetivo.esParaDerrotarEnemigos()) {
43         valorRetorno = valorRetorno
44           and not mundo.destruido()
45           and mundo.enemigosDerrotados() >= objetivo.enemigosADerrotar()
46       }
47       if(unObjetivo.esParaDefenderAliados()) {
48         valorRetorno = valorRetorno
49           and not mundo.destruido()
50           and mundo.aliadosCaidos() <= objetivo.aliadosPrescindibles()
51       }
52       if(unObjetivo.esMultiple()) {
53         valorRetorno = valorRetorno
54           and self.resolverExitoMultiple(unObjetivo)
55           //Si tiene un objetivo multiple dentro de otro llama
```

```
56            //al mismo metodo pero con el interno esto funciona
57            //(termina la ejecucion con el resultado esperado)
58        }
59
60      })
61      return valorRetorno
62    }
63  }
64  class Objetivo {
65
66    //tiempo que si se supera la mision es exitosa
67    var tiempoDeSupervivencia
68
69    //cantidad de enemigos que se deben derrotar para el exito
70    var enemigosADerrotar
71
72    //cantidad de aliados maxima que se pueden perder
73    //Si se supera se fracasa
74    var aliadosPrescindibles
75
76    //********Metodos para configurar el objetivo************
77
78    method configurarComoSupervivencia(_tiempo) {
79      tiempoDeSupervivencia = _tiempo
80      enemigosADerrotar = null
81      aliadosPrescindibles = null
82    }
83
84    method configurarParaDerrotarEnemigos(_enemigosADerrotar) {
85      tiempoDeSupervivencia = null
86      enemigosADerrotar = _enemigosADerrotar
87      aliadosPrescindibles = null
88    }
89
90    method configurarParaDefenderAliados(_prescindibles) {
91      tiempoDeSupervivencia = null
92      enemigosADerrotar = null
93      aliadosPrescindibles = _prescindibles
94    }
95
96    //************Metodos para saber el tipo de Objetivo**********
97    method esSuperviciencia() {return tiempoDeSupervivencia != null}
98    method esParaDerrotarEnemigos() {return enemigosADerrotar != null}
99    method esParaDefenderAliados()  {return aliadosPrescindibles != null}
100   method esMultiple() {return false}
101
102   //*********Getters para exponer el estado interno ************
103
104   method tiempoDeSupervivencia() {return tiempoDeSupervivencia}
105   method enemigosADerrotar() {return enemigosADerrotar}
106   method aliadosPrescindibles() {return aliadosPrescindibles}
107 }
108
109 class ObjetivoMultiple {
110   //Es una coleccion de objetivos, instancias de las clases
111   //Objetivo y/o ObjetivoMultiple
112   var objetivosInternos
113
114   constructor (_objetivosInternos) {objetivosInternos = _objetivosInternos}
115   //************Metodos para saber el tipo de Objetivo**********
116   method esSuperviciencia() {return false}
117   method esParaDerrotarEnemigos() {return false}
118   method esParaDefenderAliados()  {return false}
119   method esMultiple() {return true}
120   //*********Getters para exponer el estado interno ************
121   method objetivosInternos(){return objetivosInternos}
122
123 }
124
125 //La implementacion de esta clase no interesa a los efectos del examen
126 //Solo se muestran los mensajes que entiende
```

```
class Mundo {
  method tiempoActual() //Devuelve el tiempo actual
  method enemigosDerrotados() // Devuelve la cantidad de enemigos derrotados
  method aliadosCaidos() //Devuelve la cantidad de aliados caidos
  method destruido() //Devuelve true si se destruyo el mundo
  method personajeEstaVivo() //Devuelve si el personaje principal esta Vivo
}
```

Listing 27: polimorfismoMisiones.wlk

```

test "ambiente" {

  var mundo = new Mundo()
  var obj1 = new Objetivo()
  obj1.configurarComoSupervivencia(1500)
  var obj2 = new Objetivo()
  obj2.configurarParaDefenderAliados(5)
  var objetivo = new ObjetivoMultiple(#{obj1, obj2})
  var mision = new Mision(objetivo, mundo)
  assert.notThat(mision.esExitosa())

}
```

Listing 28: polimorfismoMisiones.wtest

Ejercicio 18: Turismo

Modifique el código utilizando polimorfismo, organización del comportamiento entre las clases, y (en un caso) mensajes adecuados a las colecciones, para mejorar la distribución de responsabilidades. Evite el código duplicado. Mejore si corresponde los lanzamientos de errores.

```
class Paquete {

  var cantidadPersonas //un numero
  var servicios //coleccion de instancias de la clase Servicio
  var premium //un booleano
  var reservado = false

  constructor(_servicios, _cantidadPersonas, _premium) {
    cantidadPersonas = _cantidadPersonas
    servicios = _servicios
    premium = _premium
  }

  method estaReservado() {return reservado}

  //Se puede reservar si no esta reservado y
  //TODOS los servicios se pueden reservar
  method sePuedeReservar() {

    if(self.estaReservado()) {
      return false
    }
    var todosSePuedenReservar = true
    servicios.forEach({unServicio =>

      if(unServicio.esHotel()) {
        //Un hotel se puede reservar si hay lugares disponibles.
        //Ademas, si el paquete es premium el hotel tiene que ser
        //como minimo de 4 estrellas
        todosSePuedenReservar = todosSePuedenReservar and
          unServicio.lugaresDisponibles() >= cantidadPersonas and
          (not premium or unServicio.estrellas() >= 4)
      }
      if(unServicio.esVehiculoParaTralado()) {
```

```
35          //Un traslado se puede reservar si el vehiculo cuenta con
36          //lugares disponibles y tiene la verificacion tecnica al dia.
37          //Si el paquete es premium, tambien tiene que cumplir que
38          //tenga aire Acondicionado
39          todosSePuedenReservar = todosSePuedenReservar and
40            unServicio.lugaresDisponibles() >= cantidadPersonas and
41            unServicio.tieneVTV() and
42            (not premium or unServicio.tieneAireAcondicionado())
43        }
44        if(unServicio.esTour()) {
45          //Un tour tiene un vehiculo para traslado y opcionalmente
46          // seguridad privada
47          //Para que se pueda reservar se tienen que cumplir todas las
48          //condiciones sobre el vehiculo (que son las mismas que se
49          //piden  a los servicios que son vehiculos para traslados)
50          // Ademas, el tour tiene que tener espacio disponible que es
51          //independiente de los lugares del vehiculo. (Por ejemplo,
52          //se puede hacer un tour para 10 personas en una combi para 15).
53          // Si es premium, el tour tiene que tener seguridad privada
54
55          todosSePuedenReservar = todosSePuedenReservar and
56            unServicio.lugaresDisponibles() >= cantidadPersonas and
57            unServicio.vehiculoTrasladoDeTour().lugaresDisponibles()
58                >= cantidadPersonas and
59            unServicio.vehiculoTrasladoDeTour().tieneVTV() and
60            (not premium or
61            (unServicio.vehiculoTrasladoDeTour().tieneAireAcondicionado()
62              and unServicio.tieneSeguridadPrivada()))
63        }
64      })
65      return todosSePuedenReservar
66    }
67
68    //Cuando se reserva se cambia el estado y se modifican los lugares
69    //disponibles en los servicios
70    // IMPORTANTE: Analizar que sucede si se pide reservar a un paquete
71    // que no cumple  con las condiciones.
72    //Modificar en caso de que la estrategia fuera incorrecta
73    method reservar() {
74      reservado = true
75      servicios.forEach( {unServicio =>
76        unServicio.lugaresDisponibles(
77          unServicio.lugaresDisponibles()-cantidadPersonas)
78        if(unServicio.esTour()) {
79          unServicio.vehiculoTrasladoDeTour().lugaresDisponibles(
80            unServicio.vehiculoTrasladoDeTour().lugaresDisponibles()
81              - cantidadPersonas)
82            }
83      })
84    }
85
86  }
87
88  class Servicio {
89
90    var lugaresDisponibles //lugares disponibles del SERVICIO
91    var estrellas = 1//cantidad de estrellas del HOTEL
92      var aireAcondicionado = false //si el VEHICULO tiene  aire acond.
93      var vtv = true //si el VEHICULO tiene la vtv
94      var seguridadPrivada = false //si el TOUR tiene seguridad privada
95      var vehiculoParaTrasladoDeTour = null // es otra instancia de
96                          //Servicio que tiene la
97                              //informacion del vehiculo
98                              //que usa el TOUR
99
100     var esHotel = false
101     var esTour = false
102     var esVehiculoParaTraslado = false
103
104   constructor(_lugaresDisponibles) {
105     lugaresDisponibles=  _lugaresDisponibles
```

```
106   }
107   method configurarComoHotel(_estrellas) {
108     esHotel = true
109     esTour = false
110     esVehiculoParaTraslado = false
111     estrellas = _estrellas
112   }
113   method configurarComoVehiculo(_tieneAire, _tieneVtv) {
114     esHotel = false
115     esTour = false
116     esVehiculoParaTraslado = true
117     aireAcondicionado = _tieneAire
118     vtv = _tieneVtv
119   }
120   method configurarComoTour(_vehiculoTraslado, _seguridadPrivada) {
121     esHotel = false
122     esTour = true
123     esVehiculoParaTraslado = false
124     vehiculoParaTrasladoDeTour = _vehiculoTraslado
125     seguridadPrivada = _seguridadPrivada
126   }
127
128   method lugaresDisponibles() {return lugaresDisponibles}
129   method lugaresDisponibles(_lugaresDisponibles) {
130     lugaresDisponibles = _lugaresDisponibles
131   }
132   method esHotel() {return esHotel}
133   method esTour() {return esTour}
134   method esVehiculoParaTralado() {return esVehiculoParaTraslado}
135   method estrellas() {return estrellas}
136   method tieneAireAcondicionado() {return aireAcondicionado}
137   method tieneVTV() {return vtv}
138   method vehiculoTrasladoDeTour() {return vehiculoParaTrasladoDeTour}
139   method tieneSeguridadPrivada() {return seguridadPrivada}
140 }
```

Listing 29: polimorfismoTurismo.wlk

```
1  test "reservaExitosa" {
2    var luchoHotel = new Servicio(50)
3    var micro = new Servicio(30)
4    var combi = new Servicio(15)
5    var cityTour = new Servicio(10)
6    luchoHotel.configurarComoHotel(2)
7    micro.configurarComoVehiculo(true, true)
8    combi.configurarComoVehiculo(true, true)
9    cityTour.configurarComoTour(combi, false)
10   var paquete = new Paquete(#{luchoHotel, micro, cityTour}, 2, false)
11
12   assert.that(paquete.sePuedeReservar())
13   paquete.reservar()
14   assert.that(paquete.estaReservado())
15   assert.equals(48, luchoHotel.lugaresDisponibles())
16   assert.equals(28, micro.lugaresDisponibles())
17   assert.equals(8, cityTour.lugaresDisponibles())
18   assert.equals(13, combi.lugaresDisponibles())
19 }
```

Listing 30: polimorfismoTurismo.wtest

Ejercicio 19: Transporte

Modifique el código utilizando polimorfismo, organización del comportamiento entre las clases, y (en un caso) mensajes adecuados a las colecciones, para mejorar la distribución de responsabilidades.

```
1
2  class Viaje {
```

```
3
4   var medio //es una instancia de la clase MedioTransporte
5   var origen //es una ubicacion
6   var destino //es una ubicacion
7
8   constructor(_medio, _origen, _destino) {
9     medio = _medio
10    origen = _origen
11    destino = _destino
12  }
13
14  /**Devuelve true si el viaje puede ser realizado */
15  method esPosible() {
16
17    if(medio.esCaminante()) {
18      return self.hayCamino(origen, destino)
19        and self.distancia(origen, destino) < medio.distanciaMaxima()
20    }
21    if(medio.esAuto()) {
22      return self.hayCalles(origen, destino)
23        and self.distancia(origen, destino) < medio.distanciaMaxima()
24    }
25    if(medio.esColectivo()) {
26      var paradaSubir = medio.paradaCercana(origen)
27      var paradaBajar = medio.paradaCercana(destino)
28      var caminante = new MedioTransporte("caminante", 800)
29      var caminoHastaSubida = new Viaje(caminante, origen, paradaSubir)
30      var caminoDesdeBajada = new Viaje(caminante, paradaBajar, destino)
31      return caminoHastaSubida.esPosible()
32        and caminoDesdeBajada.esPosible()
33        and self.distancia(origen,destino) < medio.ditanciaMaxima()
34    }
35
36    error.throwWithMessage("No se reconoce el tipo del medio")
37  }
38
39  //Este es un metodo auxiliar usado solo en esPosible()
40  method distancia(desde, hasta) {
41    if(medio.esCaminante()) {
42      return navegador.distanciaPeatonal(desde, hasta)
43    }
44    if(medio.esAuto()) {
45      return navegador.distanciaPorCaminoTransitable(desde, hasta)
46    }
47    if(medio.esColectivo()) {
48      var tramos = medio.tramosEntre(desde, hasta)
49      var sumatoria = 0
50      tramos.foreach({unTramo =>
51          sumatoria = sumatoria
52                + navegador.distanciaPorCaminoTransitable(
53                      unTramo.origen(), unTramo.destino())
54          })
55      return sumatoria
56    }
57
58    error.throwWithMessage("No se reconoce el tipo del medio")
59  }
60
61  //Este es un metodo auxiliar usado solo en esPosible()
62  //para los autos
63  method hayCalles(orig, dest) {
64    return navegador.existeCaminoTransitable(orig, dest)
65  }
66
67  //Este es un metodo auxiliar usado solo en esPosible(),
68  //para los caminantes
69  method hayCamino(orig, dest) {
70    return navegador.existeCaminoPeatonal(orig, dest)
71  }
72 }
73
```

```
74  class MedioTransporte {
75    var tipo
76    var distanciaMaxima
77    var tramos
78
79    //constructor para autos y caminantes
80    //por eso no interesa inicializar la variable
81    //tramos
82    constructor (_tipo, _distanciaMaxima){
83      tipo = _tipo
84      distanciaMaxima = _distanciaMaxima
85    }
86
87    //si es un colectivo hay que usar este constructor
88    //porque si no fallan los metodos tramosEntre y paradaCercana
89    constructor (_tipo, _distanciaMaxima, _tramos){
90      tipo = _tipo
91      distanciaMaxima = _distanciaMaxima
92      tramos = _tramos
93    }
94
95    method esCaminante() {return tipo == "caminante"}
96    method esAuto() {return tipo == "auto"}
97    method esColectivo() {return tipo == "colectivo"}
98    method distanciaMaxima() {return distanciaMaxima}
99
100   /**
101    * Para colectivos: devuelve todos los tramos del recorrido
102    * que hay entre la parada mas cercana al origen
103      * y la parada mas cercana al destino
104    */
105   method tramosEntre(origen, destino) {
106     //ATENCION: Esta implementacion usa la variable de
107     //instancia tramos y el metodo self.paradaCercana(ubicacion)
108   }
109   /**
110     * Para colectivos: devuelve una ubicacion que esta dentro
111     * de un tramo del recorrido
112     */
113   method paradaCercana(ubicacion) {
114     //ATENCION: Esta implementacion usa la variable de instancia tramos
115   }
116 }
117 /**
118  * Modela una ubicacion en el mundo.
119  * Todas las referencias a un origen, un destino,
120  * un lugar desde o un lugar hasta
121  * (en Viaje, TramoDeColectivo y navegador),
122  * y las paradas de colectivo, son instancias de esta clase
123  */
124 class Ubicacion {
125   //implementacion
126 }
127 /**
128  * Modela un tramo de un recorrido de un colectivo
129  * El recorrido de un colectivo entre dos paradas
130  * es una lista de instancias de esta clase
131  */
132 class TramoDeColectivo {
133   method origen() {/*implementacion*/}
134   method destino() {/*implementacion*/}
135 }
136 object navegador {
137   /**Devuelve si existe al menos un camino
138    * usando caminos peatonales y/o calles */
139   method existeCaminoPeatonal(orig, dest) {/*implementacion*/}
140
141   /**Devuelve la distancia entre dos puntos
142    * usando caminos peatonales y/o calles */
143   method distanciaPeatonal(orig, dest) {/*implementacion*/}
144
```

```
145   /**Devuelve la distancia entre dos puntos usando calles */
146   method distanciaPorCaminoTransitable(orig, dest) {/*implementacion*/}
147
148   /**Devuelve si existe al menos un camino usando calles */
149   method existeCaminoTransitable(orig, dest) {/*implementacion*/}
150 }
```

Listing 31: polimorfismoTransporte.wlk

3. Referencias

Ejercicio 20: Mascotas

Teniendo en cuenta las siguientes clases:

```
class Animal {
  var duenio
  method duenio() = duenio
  method duenio(_duenio) { duenio = _duenio }
  method aguante() { return 3 }
}
class Perro inherits Animal {
  override method aguante() { return super() + 2 }
}

class Duenio {  }
```

Listing 32: referenciasMascotas.wlk

Parte 1: *Estado inicial*

Construya el grafo de objetos luego de ejecutar el siguiente código:

```
var hijitus = new Duenio()
var linyera = new Duenio()
var tom = new Duenio()
var pichichus = new Perro()
pichichus.duenio(hijitus)
var diogenes = new Perro()
diogenes.duenio(linyera)
var garfield = new Animal()
garfield.duenio(tom)
var animales = #{pichichus, diogenes, garfield}
```

Listing 33: referenciasMascotas.wpgm

Parte 2: *Estado Final*

Tomando el estado configurado como inicial, ejecute el siguiente código y construya el grafo de objetos final. En caso de que alguna línea lance un error, indique el motivo del mismo e ignórela.

```
pichichus.duenio(garfield.duenio())
hijitus = garfield.aguante()
diogenes = diogenes.aguante()
animales.remove(diogenes)
pichichus = hijitus == animales.size()
animales.add(garfield.duenio())
var duenios = animales.map({x => x.duenio()})
```

Listing 34: referenciasMascotas.wpgm

Ejercicio 21: Empresas

Teniendo en cuenta las siguientes clases:

```
class Persona {}
class Empresa {
  var director
  method director() = director
  method director(_director) {
    director = _director
  }
}
```

Listing 35: referenciasEmpresas.wlk

Parte 1: *Estado inicial*

Construya el grafo de objetos luego de ejecutar el siguiente código:

```
var mexicano = new Persona()
var chino = new Persona()
var argentino = new Persona()
var saraza = new Empresa()
var cadornaCo = new Empresa()
saraza.director(chino)
cadornaCo.director(argentino)
var empresas = [saraza, cadornaCo]
var directores
```

Listing 36: referenciasEmpresas.wpgm

Parte 2: *Estado Final*

Tomando el estado configurado como inicial, ejecute el siguiente código y construya el grafo de objetos final. En caso de que alguna línea lance un error, indique el motivo del mismo e ignórela.

```
mexicano = cadornaCo
chino.director(chino)
cadornaCo.director(mexicano)
empresas.add(chino)
directores = empresas.map({unaEmpresa => unaEmpresa.director()})
directores = empresas
directores.remove(chino)
directores = empresas.map({unaEmpresa => unaEmpresa.director()})
```

Listing 37: referenciasEmpresas.wpgm

Ejercicio 22: Sobrinos de Donald

Teniendo en cuenta las siguientes clases:

```
class Sociable {
  var amigo
  method amigo(_amigo) { amigo = _amigo }
  method amigo() { return amigo }
}

class Ermitanio {
  method amigo() { return self }
}

class Casa {
  var personas = [] //es una LISTA!! (no conjunto)

  method agregarPersona(_persona) { personas.add(_persona) }
  method personas() { return personas }
  method masNuevo() { return personas.last() }
}

class Club inherits Casa {
  override method agregarPersona(_persona) {
    super(_persona.amigo())
  }
}
```

Listing 38: referenciasSobrinosDonald.wlk

Parte 1: *Estado inicial*

Construya el grafo de objetos luego de ejecutar el siguiente código:

```
var hugo = new Sociable()
var paco = new Sociable()
var luis = new Ermitanio()
hugo.amigo(paco)
paco.amigo(luis)
var hogar = new Casa()
hogar.agregarPersona(hugo)
hogar.agregarPersona(paco)
hogar.agregarPersona(luis)
var cortapalos = new Club()
```

Listing 39: referenciasSobrinosDonald.wpgm

Parte 2: *Estado Final*

Tomando el estado configurado como inicial, ejecute el siguiente código y construya el grafo de objetos final. En caso de que alguna línea lance un error, indique el motivo del mismo e ignórela.

```
cortapalos.agregarPersona(hugo)
cortapalos.agregarPersona(luis)
hugo.amigo(hogar.masNuevo())
paco = cortapalos
paco.agregarPersona(hogar)
var amigos = paco.personas().map({x => x.amigo()})
hugo.amigo(hugo)
luis.amigo(hugo)
var amigoDeHugo = hugo.amigo()
```

Listing 40: referenciasSobrinosDonald.wpgm

Ejercicio 23: Guitarras

Teniendo en cuenta las siguientes clases:

```

class Guitarra {
  var cuerdas = 6
  method cuerdas() = cuerdas
  method cuerdas(_cuerdas) { cuerdas = _cuerdas }
  method volumen() { return cuerdas * 10 }

}
class GuitarraElectrica inherits Guitarra{
  var equipo
  method equipo() { return equipo }
  method equipo(_equipo) { equipo = _equipo }
  override method volumen() {
    return super() * equipo.ganancia()
  }
}

class Equipo {
  var ganancia = 10
  method ganancia() = ganancia
  method ganancia(_ganancia) { ganancia = _ganancia }
}
```

Listing 41: referenciasGuitarras.wlk

Parte 1: *Estado inicial*

Construya el grafo de objetos luego de ejecutar el siguiente código:

```
var criolla = new Guitarra()
var acustica12 = new Guitarra()
acustica12.cuerdas(12)
var lucille = new GuitarraElectrica()
var marshall = new Equipo()
lucille.equipo(marshall)
```

Listing 42: referenciasGuitarras.wpgm

Parte 2: *Estado Final*

Tomando el estado configurado como inicial, ejecute el siguiente código y construya el grafo de objetos final. En caso de que alguna línea lance un error, indique el motivo del mismo e ignórela.

```
marshall.ganancia(lucille.volumen())
acustica12.cuerdas(marshall.ganancia() / 100 )
var x = criolla.cuerdas() == acustica12.cuerdas()
lucille.equipo(criolla.cuerdas())
lucille.cuerdas(lucille.volumen())
lucille.equipo(marshall)
criolla = marshall
marshall = new Equipo()
marshall.ganancia(5)
var volumenFinal = lucille.volumen()
```

Listing 43: referenciasGuitarras.wpgm

Ejercicio 24: Bonificaciones

Dadas las siguientes clases:

```
class Empleado {
  var bonificacion = 0

  method bonificar(unNumero) { bonificacion = unNumero }
  method bonificacion() { return bonificacion }
  method preferido() { return self }
}

class Jefe inherits Empleado {

  var subordinados = #{}

  method addEmpleado (unEmpleado) { subordinados.add(unEmpleado) }

  override method bonificar(unNumero) {
    bonificacion = bonificacion.max(unNumero)
    subordinados.forEach(
      {subordinado => subordinado.bonificar(unNumero)}
    )
  }

  override method preferido() {
    return subordinados.min(
      {subordinado => subordinado.bonificacion()}
    )
  }

}
```

Listing 44: referenciasBonificaciones.wlk

Dibuje el grafo de objetos luego de ejecutar el siguiente código:

```
var juan = new Empleado()
var pedro = new Empleado()
var santiago = new Empleado()
var roque = new Jefe()
var luis = new Jefe()
roque.addEmpleado(juan)
roque.addEmpleado(pedro)
luis.addEmpleado(santiago)
luis.addEmpleado(roque)
```

Listing 45: referenciasBonificaciones.wpgm

Parte 1: *Estado final*

Dibuje el grafo de objetos luego de ejecutar el siguiente código:

```
luis.bonificar(10)
roque.bonificar(2)
pedro.bonificar(5)
luis.addEmpleado(roque.preferido())
luis.addEmpleado(pedro.preferido())
var preferidoLuis = luis.preferido()
```

Listing 46: referenciasBonificaciones.wpgm

Ejercicio 25: Juguetes

Dada las siguientes clases:

```
class Juguete {
  var color
  constructor(_color) {color = _color}
  method color(){return color}
  method color(_color){color = _color}
}
object pelota {
  method color() {return "rojo"}
  //a la pelota no se le puede cambiar el color
}
class GrupoDeJuguetes {
  var juguetes
  constructor(_juguetes) {juguetes = _juguetes}
  method agregarJuguete(juguete) {juguetes.add(juguete)}
  method juguetes() {return juguetes}
  method colores() {return juguetes.map({j => j.color()}).asSet()}
  method pintar(color) {juguetes.forEach({j => j.color(color)})}
}
```

Listing 47: referenciasJuguetes.wlk

Parte 1: *Estado inicial*

Dibuje el grafo de objetos luego de ejecutar el siguiente código:

```
var pepa = new Juguete("rosa")
var mcqueen = new Juguete("rojo")
var minion = new Juguete("amarillo")
var patito = new Juguete("amarillo")
var enCasa = new GrupoDeJuguetes(#{pepa,mcqueen,minion,patito,pelota})
var paraViajar = new GrupoDeJuguetes(#{minion,patito})
```

Listing 48: referenciasJuguetes.wtest

Parte 2: *Estado final*

El siguiente código se ejecuta *luego del anterior*.

```
  var x = patito == minion
  pepa = pelota
  var y = enCasa.colores().contains("rosa")
  paraViajar.pintar("rojo")
  var z = enCasa.colores().contains("amarillo")
  mcqueen.color(patito)
  var coloresDeCasa = enCasa.colores()
  enCasa.pintar("amarillo")
  paraViajar.colores().agregarJuguete(pelota)
```

Listing 49: referenciasJuguetes.wtest

Analizar para cada línea si se puede evaluar correctamente o si, por el contrario, se produce un error. Indique las líneas que producen errores y **las causas** de los mismos. Dibuje el grafo de objetos resultante de la evaluación, ignorando las líneas que producen errores. Recordar que si un objeto no entiende el mensaje se produce error. Así que sean cuidadosos porque puede pasar de manera intencional.

Ejercicio 26: Popeye

Dada las siguientes clases:

```
object popeye {
  var comio = false
  method comerEspinaca() {comio = true}
  method fuerza() {return if(comio) 1000 else 5 }
}
class Marinero {
  var fuerza = 10
  method fuerza() {return fuerza}
  method comerEspinaca() {fuerza = fuerza + 10}
}
class Barco {
  var marineros = #{}
  method agregarMarinero(marinero) {marineros.add(marinero)}
  method marineros() {return marineros}
  method alimentarMarineros() {
    return marineros.forEach({m=>m.comerEspinaca()})
  }
  method masFuerte() {return marineros.max({m=>m.fuerza()})}
}
```

Listing 50: referenciasPopeye.wlk

Parte 1: *Estado inicial*

Dibuje el grafo de objetos luego de ejecutar el siguiente código:

```
  var brutus = new Marinero()
  var olivia = new Marinero()
  var cocoliso = new Marinero()
  var perlaNegra = new Barco()
  var interceptor = new Barco()
  perlaNegra.agregarMarinero(brutus)
  perlaNegra.agregarMarinero(olivia)
  perlaNegra.agregarMarinero(popeye)
```

Listing 51: referenciasPopeye.wtest

Parte 2: *Estado final*

El siguiente código se ejecuta *luego del anterior*.

```
var x = brutus == olivia
brutus.comerEspinaca()
brutus = cocoliso
var y = perlaNegra.masFuerte()
var z = brutus == y
perlaNegra.alimentarMarineros()
interceptor.agregarMarinero(perlaNegra.masFuerte())
interceptor.agregarMarinero(perlaNegra)
interceptor.alimentarMarineros()
```

Listing 52: referenciasPopeye.wtest

Analizar para cada línea si se puede evaluar correctamente o si, por el contrario, se produce un error. Indique las líneas que producen errores y **las causas** de los mismos. Dibuje el grafo de objetos resultante de la evaluación, ignorando las líneas que producen errores. Recordar que si un objeto no entiende el mensaje se produce error. Así que sean cuidadosos porque puede pasar de manera intencional.

Ejercicio 27: Heman

Dada las siguientes clases:

```
object heman {
  var transformado = false
  method gritoDeGuerra() {transformado = true}
  method fuerza() {return if(transformado) 2000 else 10 }
}
class MasterOfUniverse {
  var fuerza = 20
  method fuerza() {return fuerza}
  method gritoDeGuerra() {fuerza = fuerza + 10}
}
class Castillo {
  var maestros = #{}
  method agregarMaestro(maestro) {maestros.add(maestro)}
  method maestros() {return maestros}
  method enfurecer() {
    return maestros.forEach({m=>m.gritoDeGuerra()})
  }
  method masFuerte() {return maestros.max({m=>m.fuerza()})}
}
```

Listing 53: referenciasHeman.wlk

Parte 1: *Estado inicial*

Dibuje el grafo de objetos luego de ejecutar el siguiente código:

```
var manatarms = new MasterOfUniverse()
var teela = new MasterOfUniverse()
var orko = new MasterOfUniverse()
var palacio = new Castillo()
var grayskull = new Castillo()
palacio.agregarMaestro(manatarms)
palacio.agregarMaestro(teela)
palacio.agregarMaestro(heman)
```

Listing 54: referenciasHeman.wtest

Parte 2: *Estado final*

El siguiente código se ejecuta *luego del anterior*.

```
var x = manatarms == teela
manatarms.gritoDeGuerra()
manatarms = orko
var y = palacio.masFuerte()
var z = manatarms == y
palacio.enfurecer()
grayskull.agregarMaestro(palacio.masFuerte())
grayskull.agregarMaestro(palacio)
grayskull.enfurecer()
```

Listing 55: referenciasHeman.wtest

Analizar para cada línea si se puede evaluar correctamente o si, por el contrario, se produce un error. Indique las líneas que producen errores y **las causas** de los mismos. Dibuje el grafo de objetos resultante de la evaluación, ignorando las líneas que producen errores. Recordar que si un objeto no entiende el mensaje se produce error. Así que sean cuidadosos porque puede pasar de manera intencional.

Ejercicio 28: Jugadores

Dada las siguientes clases:

```
class Jugador {
  var idolo
  var goles

  constructor (_idolo, _goles) {
    idolo = _idolo
    goles = _goles
  }

  method idolo() {return idolo  }

  method idolo(_idolo) {idolo = _idolo}

  method goles() {return goles}
}

object distefano {
  method idolo() {return self}
  method goles() {return 694}
}

class Equipo {
  var jugadores
  constructor(_jugadores){jugadores = _jugadores}
  method idolos() { return new Equipo(jugadores.map({j=>j.idolo()}))}
  method jugadores() {return jugadores}
  method goleador() {
    return jugadores.max({unJugador => unJugador.goles()})
  }
  method agregarJugador(jug) { jugadores.add(jug) }
}
```

Listing 56: referenciasJugadores.wlk

Parte 1: *Estado inicial*

Dibuje el grafo de objetos luego de ejecutar el siguiente código:

```
var bochini = new Jugador(distefano, 97)
var maradona = new Jugador(bochini, 352)
```

```
3  var riquelme = new Jugador(maradona, 165)
4  var francescoli = new Jugador(distefano, 244)
5  var zidane = new Jugador(francescoli, 153)
6  var estrellas = new Equipo([riquelme, zidane, maradona])
```

Listing 57: referenciasJugadores.wtest

Parte 2: *Estado final*
El siguiente código se ejecuta *luego del anterior.*

```
1  var historicos = estrellas.idolos()
2  var goleador = historicos.goleador()
3  historicos = historicos.idolos()
4  var megaGoleador = historicos.goleador()
5  bochini.idolo(maradona.idolo())
6  var goles = estrellas.sum({unJugador => unJugador.goles()})
7  historicos.agregarJugador(estrellas)
8  goleador = historicos.goleador()
```

Listing 58: referenciasJugadores.wtest

Analizar para cada línea si se puede evaluar correctamente o si, por el contrario, se produce un error. Indique las líneas que producen errores y cual es el error. Dibuje el grafo de objetos resultante de la evaluación, ignorando las líneas que producen errores.

Recordar que:

- el `map` aplicado a una lista devuelve una lista.
- una lista puede contener repetidos.
- al agregar un elemento a una lista, se agrega al final.
- si un objeto no entiende el mensaje se produce error. Así que sean cuidadosos porque puede pasar de manera intencional.