Guía 6 - Herencia Básica
Programación con Objetos 1
10/08/2018

Ejercicio 1: Golosinas

Parte 1: *Clases*
A partir de la solución del ejercicio de las golosinas de la guía 4, hacer las modificaciones necesarias para que Mariano pueda comprar varios bombones, chocolatines, caramelos, alfajores, chupetines, obleas, golosinas bañadas y pastillas tutifruti.

Escribir un test en el cual mariano compre dos bombones, un chocolatín de 50 gramos, un chocolatín de 10 gramos, dos chupetines bañados, y una pastilla tuti-fruti.

Usar constructores en aquellos casos que se necesite pasar parámetros en la inicialización (el peso del chocolatín y la golosina a bañar)

Parte 2: *Bañar Golosina*
Hacer que mariano entienda el mensaje `baniar(unaGolosina)`. El método construye una nueva golosina bañada y la agrega a la colección de golosinas.

Pensar que pasa si la Golosina ya era parte de la colección, y tomar las acciones necesarias para que el grafo de objetos quede consistente.

¿Qué pasa si se baña una golosina ya bañada?

Parte 3: *Caramelos de distintos sabores*
Hacer que pueda haber caramelos de frutilla, naranja, y chocolate. Cuando se construye el caramelo se le indica de que gusto es.

Parte 4: *Bombones duros*

Los bombones duros son bombones (tienen las mismas reglas para el precio y gusto) pero cuando reciben un mordisco, en lugar de comportarse como el resto de los bombones, pierden el 10 % de su peso

Indicar en cuál clase se encuentra el método que se ejecuta en cada caso, detallando el recorrido que realiza el method lookup.

```
var bombon = new Bombon()
bombon.mordisco()
bombon.peso()
bombon = new BombonDuro()
bombon.mordisco()
bombon.peso()
```

Parte 5: *Caramelos con corazón de chocolate (Rellenos)*
Los caramelos con corazón de chocolate son caramelos que al recibir un mordisco, además de comportarse como todos los caramelos cambian su sabor a chocolate

Indicar en cuál clase se encuentra el método que se ejecuta en cada caso, detallando el recorrido que realiza el method lookup.

```
var caramelo = new Caramelo()
caramelo.mordisco()
caramelo.peso()
caramelo.sabor()
caramelo = new CarameloRelleno()
caramelo.mordisco()
caramelo.peso()
caramelo.sabor()
```

Parte 6: *Obleas Crujientes*
Las obleas crujientes son como todas las obleas y cuando reciben un mordisco pierden el peso que pierden todas las obleas (50 % si el peso es mayor a 70g o 25 % si es menor). Pero, en los primeros 3 mordiscos pierde 3 gramos adicionales.

Indicar en cuál clase se encuentra el método que se ejecuta en cada caso, detallando el recorrido que realiza el method lookup.

```
var oblea = new Oblea()
oblea.mordisco()
oblea.peso()
oblea = new ObleaCrujiente()
oblea.mordisco()
oblea.peso()
```

Parte 7: *Chocolatines VIP y Chocolatines Premium*

Los chocolatines VIP, son como todos los chocolatines, pero se guardan en una heladera que aporta un coeficiente de humedad (un número entre 0 y 1). Este coeficiente está involucrado para el cálculo del peso: El peso de un chocolatin VIP es el peso que tendría un Chocolatín cualquiera: `(pesoInicial - gramosConsumidos)` multiplicado por `1 + humedad`.

Los chocolatines Premium son un tipo especial de chocolatines VIP que vienen con una cobertura especial que los hace más resistentes a la humedad. Por lo tanto, La humedad en estos chocolatines es la mitad de la humedad de los chocolatines VIP.

```
var chocolatin = new Chocolatin()
chocolatin.peso()
chocolatin = new ChocolatinVIP()
```

```
chocolatin.peso()
chocolatin = new ChocolatinPremium()
chocolatin.peso()
```

Ejercicio 2: Análisis de method lookup

Dada las clase A, B y C.

```
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'

}
```

Ejecutar el siguiente código en el REPL e indicar que devuelve en cada línea.

```
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()
```

Ejercicio 3: Otro de análisis de method lookup
En este ejercicio tenemos las clases `A`, `B`, `C` y `D`, que se definen así.

```
class A {
    method m1() { return 1 }
    method m3() { return 3 }
    method m5() { return self.m8() + 3 }
}

class B inherits A {
    method m2() { return self.m1() + self.m3() }
    override method m5() { return self.m6() + 5 }
    method m6() { return self.m5() + 6 }
    method m8() { return 18 }
}

class C inherits B {
    override method m2() { return super() + 1 }
    override method m3() { return 23 }
    override method m5() { return 25 }
    method m10() { return self.m9() + 4 }
}

class D inherits B {
    override method m1() { return 31 }
    override method m2() { return super() + self.m8() }
    override method m6() { return 36 }
    method m9() { return 29 }
}
```

Indicar qué pasa al intentar ejecutar cada una de las siguientes líneas:

a) `new A().m2()`
b) `new B().m2()`
c) `new C().m2()`
d) `new D().m2()`
e) `new A().m5()`
f) `new B().m5()`
g) `new B().m6()`
h) `new C().m6()`
i) `new D().m5()`
j) `new C().m10()`

Ayuda importante: dos de los items entran en loop, y tres dan error porque hay algún objeto que recibe un mensaje para el que no tiene método.

Ejercicio 4: Más de Golondrinas
Agregar funcionalidad al ejercicio de Pepita de la guía 5.

Parte 1: *Golondrina Andina*
Las golondrinas andinas cada vez que comen además de sumar su energía con la formula ultra conocida: 4 * gramos, vuelan 1 kilómetro en círculos expresando su felicidad.

Hacer un test en el cual Roque entrene a una golondrina, un gorrión, una paloma y una golondrina andina.

Parte 2: *Golondrina Parda*
Las golondrinas pardas son golondrinas que vuelan y comen con las mismas reglas de las golondrinas que ya se trabajaron en la guías anteriores[1]. Sin embargo cuando le dicen que haga lo que quiera su comportamiento es distinto: Si la última acción que hizo fue comer, entonces vuela 5 kilómetros. Por lo contrario, si la última acción que hizo fue volar, come 10 gramos de alpiste.

Aclaración: Cuando una golondrina parda es creada, se comporta como si lo último que hizo hubiese sido volar.

Modificar el test de la parte anterior para que Roque entrene también a una golondrina parda. Hacer un segundo test que verifique el comportamiento de una golondrina parda para ambos casos: que haya comido o que haya volado previamente.

Ejercicio 5: Más sobre el camión
Modificar el ejercicio de 4 la guía 4 (camión).

Parte 1: *Distintos embalajes*
Cambiar la implementación del objeto embalaje de seguridad para que sea construido a partir de una clase, de manera tal que dos cargas distintas puedan tener su propio embalaje de seguridad. Por ejemplo, que bumblebee esté embalado, y por otro lado que knight rider lo esté.

Pregunta: ¿La solución admite embalar un objeto que ya está embalado? ¿Cómo se comporta en ese caso?

Parte 2: *Distintos modelos de embalajes*
Todos los embalajes tienen en común que envuelven a una cosa a transportar con el objetivo de disminuir su peligrosidad. Pero hay distintos modelos de embalajes:

- Estándar: el nivel de peligrosidad es la mitad de lo que envuelve y no aporta nada al peso. (Es el comportamiento que tenía el objeto en la guía anterior).
- Liviano: Es muy eficaz con los objetos que pesan menos de 200 kilos, ya que el nivel de peligrosidad para éstos es 0. Si el objeto que envuelve es más pesado que 200, entonces reduce la peligrosidad a la mitad, como si fuera un embalaje estándar. Tampoco agrega peso: el peso es el mismo que el del contenido.

[1] cuando comen alpiste aumentan una cantidad de joules equivalentes a 4 veces el peso del alpiste consumido y cuando vuelan gastan 10 de costo fijo más un joule por kilómetro

- Bolsa de aserrín: Disminuye 10 unidades de peligrosidad del objeto que contiene por cada kilogramo de aserrín que hay en la bolsa. El peso es el del contenido más la cantidad de aserrín.

Parte 3: *Embalar hasta 2 cosas*

Modificar los embalajes para que cada uno pueda envolver a 1 o 2 cosas. Agregar los constructores necesarios para que sea cómodo construir un embalaje con 1 o con 2 cosas. **¡Ojo con esto!** El embalaje tiene que andar bien ya sea teniendo un contenido o dos.

El peso del embalaje es la suma de sus contenidos. En el caso de la bolsa de aserrín también suma el peso del aserrín.

En cuanto a la peligrosidad. El estándar devuelve la peligrosidad de su contenido más peligroso. El liviano devuelve cero si la suma de los pesos de su contenido es superior a 200. En caso contrario se comporta como el estándar. La bolsa de aserrín disminuye 10 unidades por kilo de aserrín, tomando como base el contenido más peligroso.