

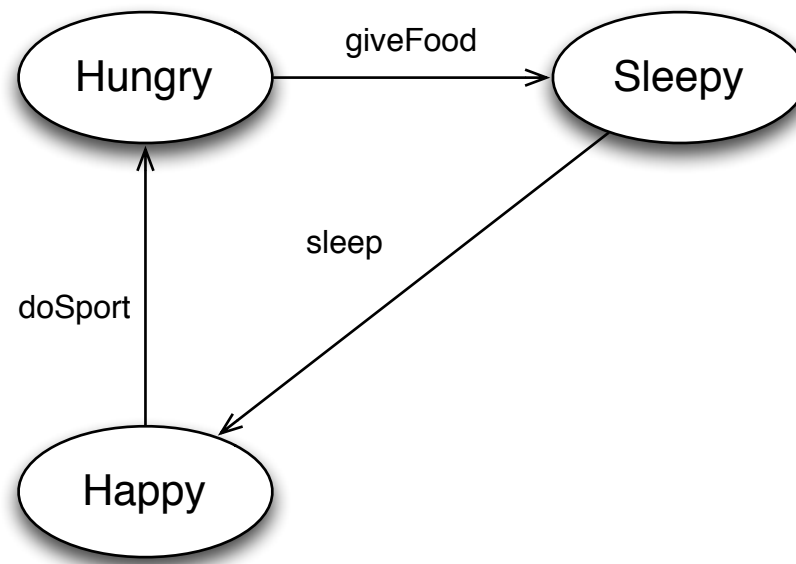


Exercise

Nancy Hitschfeld
Matías Toro

Tamagotchi

Un Tamagotchi es un pequeño animal virtual. Como todo los seres vivos, un tamagotchi tiene necesidades. A continuación se muestra un pequeño diagrama de estado:



Tamagotchi

Al principio un Tamagotchi esta feliz (happy). Si él hace deporte, tiene hambre (hungry). Al darle comida, le da sueño y se pone a dormir (sleepy). Después haber dormido, esta de nuevo feliz.

Exercise

Parte A - Implementa el Tamagotchi.

Parte B - Para poder hacer “tests”, queremos tener una forma de saber en que estado esta el tamagotchi.

Por ejemplo, tenemos `new Tamagotchi().isHappy()` que retorna `true`.

Tenemos Tamagotchi `t = new Tamagotchi(); t.doSport(); t.isHungry()` retorna `true` y `t.isHappy()` retorna `false`. Proponga una implementación para los métodos `isHappy()`, `isHungry()` y `isSleepy()`.

A solution

```
class Tamagotchi() {  
  private var state: State = new Happy()  
  
  def isHappy(): Boolean = state.isHappy  
  def isHungry(): Boolean = state.isHungry  
  
  def doSport(): Unit = state.doSport()  
  def giveFood(): Unit = state.giveFood()  
  def sleep(): Unit = state.sleep()  
  
  def isSleepy(): Boolean = state.isSleepy()  
  
  def setState(s: State): Unit = {  
    state = s  
    s.setTamagotchi(this)  
  }  
}
```

A solution

```
abstract class State {  
  protected var tama: Tamagotchi = null  
  
  def isHappy() = false  
  def isHungry() = false  
  def isSleepy() = false  
  
  def error(): Unit = throw new AssertionError("Wrong state!")  
  
  def doSport(): Unit = this.error()  
  def giveFood(): Unit = this.error()  
  def sleep(): Unit = this.error()  
  
  def setState(aState: State): Unit = tama.setState(aState)  
  
  def setTamagotchi(tamagotchi: Tamagotchi): Unit = {  
    tama = tamagotchi  
  }  
}
```

A solution

```
class Happy extends State {  
  override def isHappy = true  
  override def doSport(): Unit = this.setState(new Hungry())  
}  
  
class Hungry extends State {  
  override def isHungry = true  
  override def giveFood(): Unit = this.setState(new Sleepy())  
}  
  
class Sleepy extends State {  
  override def isSleepy = false  
  override def sleep(): Unit = this.setState(new Happy())  
}
```

A solution

```
class TamagotchiTest extends munit.FunSuite {  
  private var tamagotchi: Tamagotchi = null  
  
  override def beforeEach(context: BeforeEach): Unit = {  
    tamagotchi = new Tamagotchi()  
  }  
  
  test("testHappy"){  
    assert(tamagotchi.isHappy())  
  }  
  
  test("testDoSport"){  
    tamagotchi.doSport()  
    assert(!tamagotchi.isHappy())  
    assert(tamagotchi.isHungry())  
  }  
  
  test("testWrongState"){  
    val e = Assert.assertThrows(classOf[AssertionError], () => tamagotchi.giveFood())  
    assertEquals("Wrong state!", e.getMessage)  
  }  
}
```


License



Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

You are free to:

- Share: copy and redistribute the material in any medium or format
- Adapt: remix, transform, and build upon the material for any purpose, even commercially

The licensor cannot revoke these freedoms as long as you follow the license terms



Attribution: you must give appropriate credit



ShareAlike: if you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original

Complete license: <https://creativecommons.org/licenses/by-sa/4.0/>



dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

www.dcc.uchile.cl

f @ in / DCCUCHILE