Guía 2: Relaciones entre objetos, self y Polimorfismo
Programación con Objetos 1
Versión del 10/08/2018

En estos ejercicios se busca trabajar a los objetos que programamos no solo como receptores de mensajes (guía 1), si no también como emisores. De esta manera vamos a ver las dos partes del envío de los mensajes.

Tener en cuenta que un objeto puede mandarse un mensaje a sí mismo usando la palabra *self* como referencia. Esto es útil para tener código más prolijo y para evitar duplicación de código. Esto es muy importante en el desarrollo de software, porque genera programas más mantenible. Ejemplo:

```
method expresar() {
     if(self.contento())
          self.reir()
     else
          amigo.abrazar()
}
```

Pasado los primeros ejercicios, se empieza a explorar el concepto de *polimorfismo*: dos (o más) objetos entienden el mismo mensaje que es enviado por un tercero. Se busca que el tercero trate indistintamente a los objetos *polimórficos*. Esta característica es de las más importantes a la hora de programar con objetos.

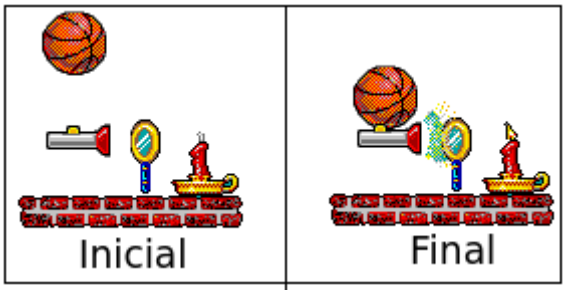
Los ejercicios están pensados para resolverse en orden. Esto responde por un lado a una cuestión pedagógica, pero también a una práctica, ya que algunos requieren de piezas de código realizadas anteriormente en esta guía o en la guía 1.

Ejercicio 1: TIM

TIM *(The Incredible Machine)* es un videojuego clásico que consta de conectar distintos artefactos al mejor estilo *Coyote* del *Correcaminos*. Cada artefacto puede actuar sobre otro, generando una simpática cadena de efectos que logra un objetivo.

Se pide modelar con objetos la siguiente secuencia de eventos para encender una vela que inicialmente está apagada:

1. Cuando la pelota cae, enciende una linterna
2. Cuando la linterna se enciende, ilumina una lupa
3. Cuando la lupa es iluminada, se prende la vela



Se espera que luego de ejecutar la siguientes líneas:

```
pelota.caer()
vela.encendida()
```

la respuesta sea: `true`

Ejercicio 2: Pepita independiente

Hacer que la golondrina Pepita (ejercicio 3 y 4 de la guía 1) entienda el mensaje `haceLoQueQuieras()`. El comportamiento que tiene pepita al recibir ese mensaje es:

- Si Pepita está débil, come 20 gramos.
- Si Pepita está feliz, vuela la cantidad de km que quiera volar (usar el mensaje `cuantoQuiereVolar()` ya resuelto en la guía 1)
- Si Pepita no está ni débil ni feliz, entonces no hace nada

Ejercicio 3: Calculadora avanzada

Agregar las siguientes funcionales a la calculadora desarrollada en el ejercicio 5 de la guía 1

Parte 1: *Memoria*

La calculadora debe permitir guardar un valor en memoria con el cual se puede operar. Para eso debe entender los siguientes mensajes

- `cargarMemoria()`: Guarda el valor actual en la memoria.
- `limpiarMemoria()`: Guarda 0 en la memoria.
- `memoria()`: Devuelve el valor de la memoria.
- `sumarMemoria()`: actualiza el valor actual sumándole el valor de la memoria.
- `restarMemoria()`: actualiza el valor actual restándole el valor de la memoria.
- `multiplicarMemoria()`: actualiza el valor actual multiplándole el valor de la memoria.

Nota: Se asume que nunca se invoca a un método que usa la memoria sin haberse llamado a `cargarMemoria()` previamente.

¡Ojo con esto!: No vale repetir código. la operación de suma debe estar en un solo lugar, al igual que la resta y la multiplicación.

***Parte 2:** Deshacer*
La calculadora debe permitir poder deshacer la última operación a través del mensaje `deshacer()`. Esto consiste en restaurar el valor actual al que tenía previo a una operación de suma, resta, multiplicación o cargado de número, independientemente de si se operó con la memoria o con un parámetro.

La calculadora solo permite deshacer la última operación. No se debe tener en cuenta los casos de intentar deshacer cuando aún no se ha realizado ninguna operación, ni enviar dos veces seguidas el mensaje `deshacer()`.

Ejercicio 4: Jueguitos
A Pepe le gusta jugar a hacer *jueguitos* con la pelota de fútbol en sus tiempos libres. Mientras juega le cuesta mantener la concentración para llevar la cuenta, así que espera usar un pequeño aparatito para eso. El aparato tiene dos botones y un visor led; y es capaz de ejecutar código wollok.

Con el primer botón, pepe indica que hizo un *jueguito*, por lo tanto el sistema tiene que sumar en una unidad un contador. Con el otro botón, Pepe indica que se le cayó la pelota al piso. Por lo tanto el contador se pone en cero y verifica si se ha hecho un nuevo record. El visor led indica el record actual que Pepe quiere batir y la cantidad de jueguitos que lleva acumulados en la jugada actual.

Se pide modelar el dominio de la aplicación que se ejecuta dentro del aparato. Para eso, se va a programar un objeto que represente a Pepe, el cual va a entender 4 mensajes. Además, se va a utilizar el contador programado en el ejercicio 1 de la guía 1.
¡Ojo con esto!: Es **obligatorio** usar el contador, y no vale agregarle cosas.

Los mensajes que entiende el objeto Pepe son:

- `jueguito()`: Incrementa en 1 el contador.
- `pique()`: Usado para indicar que la pelota picó en el piso, actualiza el record si es necesario y resetea el contador. El record inicial es cero y cada vez que el método es invocado se queda con el valor máximo entre el record actual y el valor que tiene el contador en ese momento.
- `acumulado()`: Devuelve el valor actual del contador
- `record()`: Devuelve la máxima cantidad de *jueguitos* que hizo pepe sin que la pelota pique en el suelo.

Nota: Pensar acerca de quien tiene que tener la responsabilidad de mantener el record de pepe.

Ejercicio 5: Agregados al Enterprise
Ampliar el modelo de la nave Enterprise que se describe en la guía 1, para que entienda los siguientes mensajes.

Parte 1: *Más mensajes*

a. `seLaBanca()`
   La Enterprise se la banca si no necesita fortalecerse, y además su fortaleza defensiva es >= 30.

b. `diaDeSuerte()`
   Indica que la Enterprise tuvo un día de suerte, o sea, que encuentra una pila atómica y un escudo.

c. `recibirDobleAtaque(puntosAtaque1, puntosAtaque2)`
   Indica que la Enterprise recibe dos ataques.

Parte 2: *Más cosas para encontrar:*
Modificar el modelo del enterprise para favorecer la distribución de responsabilidades y mejorar la interfaz (mensajes que recibe) hacia los usuarios de la enterprise.

Para lograr ese objetivo, hacer que la enterprise entienda el mensaje `encontrar(unaCosa)` y que deje de entender `encontrarPilaAtomica()` y encontrarEscudo().

Cuando el enterprise encuentra una cosa, el efecto que sufrirá dependerá de que es lo que se encontró:

- Una pila atómica: aumenta su potencia en 10. El máximo de potencia en 25. El máximo de potencia es 100.
- Un escudo: aumenta la coraza en 10. El máximo de coraza es 20.
- Khan: Cuando se encuentra a Khan, la enterprise recibe un ataque de 50 puntos.
- Reina Borg: Cuando se encuentra a la Reina Borg, el enterprise recibe dos ataques, el primero de 30 puntos y el segundo de 70.

Ejercicio 6: Pepita Turista
Nuestra golondrina pepita de la guía anterior encuentra reconfortante irse de vacaciones. El lugar donde vacaciona le otorga a pepita cierta cantidad de energía revitalizadora, la cual incrementa directamente su energía (la que hasta el momento solo se modificaba al volar y comer).

Parte 1: *Inicial*
Éstos son los lugares posibles y sus características:

- **Patagonia** aporta 30 joules de energía revitalizadora

- **SierrasCordobesas** aporta 70 joules de energía revitalizadora

Parte 2: *Mar del plata*
Agregar entre los lugares en los que pepita puede vacacionar a Mar del plata. La energía revitalizadora que aporta depende de si se trata de temporada alta o baja. En temporada baja aporta 80 joules, mientras que en alta resta 20 joules. TIP: se puede pensar que en temporada alta la energía que aporta es -20.

Cosas para pensar: ¿Quién tiene la responsabilidad de saber si Mar del Plata está en temporada alta o baja? ¿Cómo se cambia de temporada?

Parte 3: *Noroeste*
Agregar entre los lugares en los que pepita puede vacacionar al Noroeste argentino. La energia revitalizadora que recibe es el 10 % de su propia energía (la de pepita). Cosas para pensar: ¿Cómo sabe el noroeste la energía de pepita para aplicarle el 10 %? ¿Es necesario cambiar el mensaje? ¿Qué pasa con el resto de los lugares ya programados?

Ejercicio 7: Entrenamiento

Parte 1: *Entrenamiento de Pepita*
Roque es el entrenador de Pepita. Cuando entrena a un pepita usa su rutina para aves diseñada por él.

1. Volar 10 kms.
2. comer 300 gramos.
3. Volar 5 kms.
4. Que el ave haga lo que quiera como premio.

Se pide modelar a Roque en objetos y que entienda el mensaje `entrenar()`

Parte 2: *Entrenamiento de otros pájaros*
Pepón es un gorrión que también sabe comer, volar y hacer lo que quiera, pero lo hace de manera diferente a Pepita.

- comer: El aparato digestivo de Pepón no anda muy bien, por eso solo puede aprovechar la mitad de las calorías que aporta un alimento. Por ejemplo, si come 20 gramos, en lugar de aumentar su energia en 80 (resultado de hacer 20 * 4 según la fórmula ya conocida), lo hace en 40 (20 * 4 / 2).
- volar: Gasta 1 joule fijo y 0.5 joules por cada kilómetro recorrido.
- hacer lo que quiera: Pepón siempre vuela 1 km en este caso.

Pipa es una paloma de la cual se desconce las reglas que alteran su energía. Por eso en este sistema, lo que se espera del objeto pipa es que simplemente se acuerde cuántos

kms vuela y cuántas calorías ingiere. Esta información se puede consultar a través de los mensajes `kmsRecorridos()` y `caloriasIngeridas()`. Cuando le piden que haga lo que quiera, Pipa no hace nada.

Se pide modelar a Pepón y Pipa y hacer que Roque pueda por momentos entrenar a Pepita y en otros a Pepón o Pipa.

Ejercicio 8: Teclado Mejorado

Se pide agregar funcionalidad al teclado del ejercicio 6 y 7 de la guía 1

Parte 1: *Diferenciar Llamadas*
Se debe soportar la capacidad de diferenciar la cantidad de llamadas a celular y la cantidad de llamadas a número fijo. Para eso el objeto teclado debe ser renombrado como teléfono y entender los siguientes mensajes:

- `cantLlamadasFijos()`
- `cantLlamadasCelular()`

Tener en cuenta que hay dos lugares en el código donde se necesita distinguir si el número ingresado es un número de celular o uno fijo: en el método `esNumeroValido()` y en el método `llamar()`. Para no duplicar código en esos métodos, el objeto se debe enviar a sí mismo los mensajes `esNumeroFijo()` y `esNumeroCelular()`

Parte 2: *Facturación*
Se necesita saber cuánto dinero hay que pagar por las llamadas realizadas. Este valor depende de la duración de cada llamada y tiene valores diferentes según sea una llamada a fijo o celular.

Se debe modificar el mensaje `llamar()` para que reciba por parámetro la cantidad de tiempo que dura esa llamada en segundos.

Agregar también los mensajes

- `tiempoTotalCelulares()`: Devuelve la sumatoria del tiempo de las llamadas realizadas a números de celular.
- `tiempoTotalFijo()`: Devuelve la sumatoria del tiempo de las llamadas realizadas a números fijos.

Cuando el teléfono reciba un mensaje `totalFacturacion()` debe devolver:
`tiempoTotalCelulares * precioCelular + tiempoTotalFijo * precioFijo`.

Sin embargo, los precios son dependientes de las compañia a la que está adherido el teléfono, que puede ser cualquiera entre *RRHH* y *Estrella Fugaz*.

Compañía	Precio Fijo	Precio Celular
RRHH	0,45	0,70
Estrella Fugaz	0,50	0,60

Tabla 1: Precios por compañía.

Parte 3: *Políticas de facturación por compañía*
Las empresas están en una ardua lucha por dominar el mercado, por lo que cada una elije una manera distinta de facturar.

- RRHH regala los primeros 1000 segundos de las llamados a números fijos.
- Estrella fugaz aplica un 10 % de descuento sobre los números fijos si los mismos superan los 1500 segundos y un 15 % de descuento sobre los celulares si éstos están distribuidos en más de 120 llamadas.

Ejercicio 9: Silvestre, Tweety y Speedy González
En este jueguito online, el objetivo es manejar al personaje Silvestre, el cual va a correr a cierta velocidad, que depende de su energía. Además, la energía de Silvestre aumenta cuando come y disminuye cuando corre.

La energía inicial de Silvestre es 100

Parte 1: *Velocidad*
Silvestre debe entender el mensaje `velocidad()`. La velocidad con la que se puede mover depende de su energía y es de 5 m/s más la energía medida en joules dividido 10 joules/m/s.
La fórmula es: `velocidad = 5 + energía / 10`

Parte 2: *Correr*
Silvestre debe entender el mensaje `correr(unaDistancia)`. Lo cual provoca que disminuya su energía. El gasto de energía depende tanto de la distancia recorrida como de la velocidad al inicio del movimiento según la siguiente fórmula:
`energía consumida = 0.5 * distancia * velocidad`

Parte 3: *Comer*
Cuando Silvestre come a Tweety, la energía aumenta en 12 joules + un joule por cada gramo del peso de Tweety. Al inicio del juego el peso de Tweety es 50 gramos, pero puede variar arbitrariamente (por eso tweety necesita entender el mensaje `peso(gramos)` para indicarle el peso actual.

Cuando Silvestre come a Speedy Gonzalez, la energía que incrementa es 100.000 joules (Speedy tiene mucha energía para poder correr tan rápido).

Se pide que Silvestre entienda el mensaje `comer(alguien)`.

Al ejecutar:
`silvestre.comer(tweety)`

`silvestre.energia()`
El resultado debe ser 162.

Si luego se ejecuta:
`silvestre.comer(speedyGonzalez)`
`silvestre.energia()`
El resultado debe ser 100162.

Parte 4: *Conveniencia*
Como ayuda al usuario del juego, Silvestre debe saber si le conviene comer o no a otro personaje que está a cierta distancia. Le conviene comer si la energía que obtiene es mayor a la energía que gasta corriendo hasta el lugar donde se encuentra. Para eso, silvestre debe entender el mensaje:
`convieneComer(unPersonaje,unaDistancia)`.

Ejercicio 10: Cuentas Bancarias
Pepe y Julian son hermanos que se mudaron de la casa de sus padres buscando su independencia económica. Ahora viven juntos. Para controlar los gastos que realizan quieren un sistema programado en objetos. A continuación, se explican los requerimientos del mismo.

Parte 1: *Cuentas básicas*
Hay distintas cuentas en el sistema. Lo que se le pide a cada una de ellas son las mismas 3 cosas: saber el saldo, extraer y depositar. Para eso, las cuentas entienden los mensajes:

- `saldo()` devuelve un número que es el saldo en pesos de la cuenta
- `depositar(unaCantidadPesos)` incrementa el saldo
- `extraer(unaCantidadPesos)` disminuye el saldo

Nota: Asumir que nunca se le pide a una cuenta extraer una cantidad de pesos superior al saldo disponible.

Si bien todas las cuentas cumplen con esas responsabilidades, cada cuenta lo realiza de una manera particular.

- La cuenta de Pepe es una cuenta en pesos normal: el saldo incrementa con los depósitos y disminuye con las extracciones en exactamente la misma cantidad de pesos que indican los parámetros.
- La cuenta de Julián está embargada, esto significa que de cada depósito solo el 80 % incrementa el saldo, el 20 % restante se descarta. Por ejemplo, si Julián tiene $100 de saldo, y deposita otros $100, el saldo es $180. Además, cada vez vez que extrae se le descuenta al saldo $5 adicionales en concepto de gastos administrativos. Este gasto es aplicado solo en aquellos casos en que el gasto no deje la cuenta con saldo negativo.

- La cuenta del papá de ambos está dolarizada: mantiene internamente un saldo en dólares. Se usa igual que una cuenta en pesos (se extrae y se depositan pesos), pero en cada operación hay que convertir la moneda usando los precios de compra y de venta según corresponda. Los valores iniciales son $15.10 para la venta y $14.70 para la compra. Obviamente, estos valores pueden cambiar con el tiempo.
  - `saldo`: Devuelve `saldoEnDolares * precioDeCompra`.
  - `depositar(unaCantidadDePesos)`: incrementa el saldo dolarizado en `unaCantidadDePesos/precioDeVenta`.
  - `extraer(unaCantidadDePesos)`: Decrementa el saldo dolarizado en `unaCantidadDePesos/precioDeCompra`.

Programar todas las cuentas y probarlas.

Parte 2: *Casa de Julián y Pepe*
Modelar la casa donde viven los hermanos, de la cual interesa las compras que se haga. Para los gastos de la casa se usa una cuenta que se tiene que configurar, ya que hay épocas en que paga Julián, otras que paga Pepe, y lamentablemente, otras que lo banca el papá.

Ante cada compra que se realiza en la casa, se hace el retiro correspondiente de la cuenta. Además, se espera saber si la casa:

- es derrochona: Si la suma de todas las compras supera los 5000 pesos
- es bacán: si la casa puede afrontar una compra de 40.000 pesos

Parte 3: *Cuentas combinada*
Juan y Pepe se pusieron de acuerdo para que en ciertas épocas, afronten los gastos juntos. Para eso es necesario generar una cuenta combinada, que une otras dos. Tiene una cuenta primaria y una secundaria. La cuenta combinada se comporta de la siguiente manera:

- `saldo()`: Devuelve la suma de los saldos de las cuentas primaria y secundaria.
- `depositar(unaCantidadDePesos)`: si el saldo de la cuenta secundaria es menor a 1000 deposita en dicha cuenta. En cualquier otro caso deposita en la primaria.
- `extraer(unaCantidadDePesos)`: Extrae todo lo que puede de la cuenta primaria y el resto de la secundaria.

Programar la cuenta combinada y probarla cambiando cuales son las cuentas primaria y secundaria. La cuenta de Julián, la de Pepe y la del Papá pueden ser usadas como cuenta primaria o secundaria. Hacer que la cuenta combinada sea usada por la casa.

Ejercicio 11: Sueldo de Pepe
En un sistema de pago de liquidación de sueldos se necesita calcular el sueldo de Pepe.

Parte 1: *Sueldo*
En esta primer parte, el sueldo de pepe sólo se compone por su sueldo neto: `sueldo = neto`

El sueldo neto depende de la categoría y hay dos categorías posibles: los gerentes ganan $15.000 y los cadetes ganan $20.000. Pepe tiene solo una de estas categorías, la cual varía según Pepe sea promovido o degradado en su trabajo.

Utilizar a pepe desde el REPL cambiándole la categoría y preguntarle en cada caso su sueldo.

Parte 2: *Situación sindical*
Ahora el sueldo de pepe depende también de la situación de Pepe frente al sindicato, lo cual le va a provocar un descuento: `sueldo = neto - descuento por sindicato`

La situación sindical depende de la elección personal de Pepe, y hay tres posibilidades:

- Porcentual: descuenta un 3 % del neto
- Comprometido: descuenta un 1 % del neto más $1500 fijos
- No Sindicalizado: No descuenta nada

Utilizar a pepe desde el REPL cambiándole la categoría, la cantidad de días que falta y su bono por presentismo; preguntarle en cada caso su sueldo.

Parte 3: *Bono por presentismo*
Finalmente, el sueldo de pepe depende también de un extra que le puede otorgar un bono basado en el presentismo:
`sueldo = neto - descuento por sindicato + extra por presentismo`

Hay tres bonos por presentismo posibles.

- El normal: Otorga $2000 si la persona a quien se aplica no faltó nunca. $1000 si faltó sólo una vez y $0 en cualquier otro caso.
- El de época de ajuste: Otorga $10 si el empleado nunca faltó, $0 en cualquier otro caso.
- El demagógico: Le da $500 pesos si el sueldo neto es menor a $18.000 ó $350 en el caso opuesto. Para este bono no importa si el empleado faltó o no.

El bono que corresponde a Pepe no se sabe a priori, es una decisión de la empresa que contrató a pepe la selección del mismo.

Utilizar a pepe desde el REPL cambiándole la categoría, la cantidad de días que falta, su bono por presentismo y su situación sindical; preguntarle en cada caso su sueldo.

Ejercicio 12: Mensajeros de Película
Contexto:

Steven Spielberg está por cumplir años y George Lucas quiere realizarle un regalo. Por eso compró un álbum de figuritas de ET y lo envolvió en un paquete para ser entregado por una empresa de mensajería, que ofrece un menú de distintos mensajeros para llevar un paquete a distintos lugares. George pagó $10 por el envío, pero no estaba seguro del lugar en el cual está Steven, es posible que ande paseando por Brooklyn o que esté haciendo de las suyas en la matrix.

Requerimiento del sistema:

La empresa utiliza un sistema desarrollado en objetos para identificar que mensajero llevará ese paquete.

De todos los requerimientos que tiene el sistema, en este ejercicio sólo se pide resolver el siguiente: **Determinar si ese *paquete* puede ser entregado por un *mensajero* en un determinado *destino***. Pero cumplir el objetivo no es tan sencillo, porque hay varias reglas que respetar.

Reglas:

1. Para que el paquete pueda ser entregado debe estar pago. En el ejemplo mencionado está pago, pero no siempre es así. Puede variar.
2. Cada destino le pone restricciones a los mensajeros que quieren llegar a él. Existen dos destinos posibles:
   - Puente de Brooklyn: deja pasar a todo lo que pese hasta una tonelada (1000 kilos).
   - La Matrix: deja entrar a quien pueda hacer una llamada.
3. Por el momento, nuestra empresa de mensajería tiene 3 mensajeros, con estas características:
   - Chuck Norris: Chuck siempre pesó y seguirá pesando 900 kg por toda la eternidad. Puede llamar a cualquier persona del universo con sólo llevarse el pulgar al oído y el meñique a la boca.
   - Neo: Neo vuela, así que no pesa nada. Y anda con celular, el muy canchero. El tema es que a veces no puede llamar porque se queda sin crédito.
   - Lincoln Hawk: Tiene un peso propio que varía con el tiempo. Viaja en bicicleta ó camión. A ese peso propio se le suma el peso de su vehículo. La bici pesa 10kg. En cambio, el camión pesa media tonelada. Si el camión tiene acoplados, hay que agregar media tonelada adicional por cada uno de ellos. Hawk no tiene un mango, gracias que tiene cubiertas, y no puede llamar a nadie.

Aclaración: Para el cálculo del peso, el peso del paquete es despreciable.

Algunos casos de prueba:

- El paquete de George que no está pago no puede ser llevado por Neo a la matrix.

- El paquete de George que sí está pago puede ser llevado por Chuck a la matrix
- El paquete de George que sí está pago no puede ser llevado por Lincoln Hawk (80kg) a Brooklyn si es que utiliza un camión con un acoplado .
- La entrega anterior puede hacerse si Lincoln Hawk usa una bicicleta