Objeto, mensaje, método, variable
Programación con Objetos 1
Versión del 10/08/2018

Antes de los ejercicios, algunos mensajes que entienden los Strings.

- con `+` se obtiene la concatenación de dos Strings, p.ej. `"pepi" + "ta"` es `"pepita"`.
- con `size()` se les pide la longitud, p.ej. `"pepita".size()` es 6.
- con `startsWith(string)` se les pregunta si empiezan con un determinado substring o no.

  P.ej. `"pepita".startsWith("pep")` es `true`, `"pepita".startsWith("sumo")` es `false`.
- con `substring(desdeDonde,hastaDonde)` se obtiene una porción de un String. Para `desdeDonde`, arrancar en 0, para `hastaDonde` arrancar en 1.

  P.ej. `"pepita".substring(0,4)` es `"pepi"`, `"pepita".substring(2,5)` es `"pit"`.

1. Implementar un objeto que modele un contador. Un contador se puede incrementar o decrementar, recordando el valor actual. Al resetear un contador, se pone en cero. Además es posible indicar directamente cual es el valor actual. Este objeto debe entender los siguientes mensajes:

   - `reset()`
   - `inc()`
   - `dec()`
   - `valorActual()`
   - `valorActual(nuevoValor)`

   P.ej. si se evalúa la siguiente secuencia
   ```
   contador.valorActual(10)
   contador.inc()
   contador.inc()
   contador.dec()
   contador.inc()
   contador.valorActual()
   ```
   el resultado debe ser 12.

2. Agregar al contador del ejercicio 1, la capacidad de recordar un String que representa el último comando que se le dio. Los Strings posibles son `"reset"`, `"incremento"`, `"decremento"` o `"actualizacion"` (para el caso de que se invoque `valorActual` con un parámetro). Para saber el último comando, se le envía al contador el mensaje `ultimoComando()`.

En el ejemplo del ejercicio 1, si luego de la secuencia indicada se evalúa
`contador.ultimoComando()`
el resultado debe ser `"incremento"`.

3. Implementar un objeto que modele a Pepita, una golondrina de la que nos interesa saber qué energía tiene en cada momento, medida en joules. En el modelo simplificado que nos piden implementar, las únicas acciones que vamos a contemplar son:

   - cuando Pepita come una cantidad de comida especificada en gramos, en este caso adquiere 4 joules por cada gramo, y
   - cuando Pepita vuela una cantidad de kilómetros, en este caso gasta un joule por cada kilómetro, más 10 joules de "costo fijo" de despegue y aterrizaje.

   La energía de Pepita nace en 0. El objeto que implementa este modelo de Pepita, debe entender los siguientes mensajes:

   - `comer(gramos)`
   - `volar(kilometros)`
   - `energia()`

   P.ej. si sobre un REPL recién lanzado se evalúa la siguiente secuencia
   ```
   pepita.comer(100)
   pepita.volar(10)
   pepita.volar(20)
   pepita.energia()
   ```
   el resultado debe ser 350.

4. Agregar al modelo de Pepita del ejercicio 3, la capacidad de entender estos mensajes:

   - `estaDebil()`, Pepita está débil si su energía es menos de 50.
   - `estaFeliz()`, Pepita está feliz si su energía está entre 500 y 1000.
   - `cuantoQuiereVolar()`, que es el resultado de la siguiente cuenta. De base, quiere volar (energía / 5) kilómetros, p.ej., si tiene 120 de energía, quiere volar 24 kilómetros. Si la energía está entre 300 y 400, entonces hay que sumar 10 a este valor, y si es múltiplo de 20, otros 15. Entonces, si Pepita tiene 340 de energía, quiere volar 68 + 10 + 15 = 93 kilómetros. Para probar esto, sobre un REPL recién lanzado darle de comer 85 a Pepita, así la energía queda en 340.

     Para saber si n es múltiplo de 20 hacer: `n % 20 == 0`. Probarlo en el REPL.

5. Implementar un objeto que represente una calculadora sencilla, que permita sumar, restar y multiplicar. Este objeto debe entender los siguientes mensajes:

   - `cargar(numero)`
   - `sumar(numero)`

- `restar(numero)`
- `multiplicar(numero)`
- `valorActual()`

P.ej. si se evalúa la siguiente secuencia
```
calculadora.cargar(0)
calculadora.sumar(4)
calculadora.multiplicar(5)
calculadora.restar(8)
calculadora.multiplicar(2)
calculadora.valorActual()
```
el resultado debe ser 24.

6. Implementar un objeto que represente el teclado de un teléfono de Coronel Espingarda, donde los números tienen 5 dígitos. Se le indica a este objeto el número de teléfono que se estḿarcando, dgito por dgito. Debe memorizar el número que se está marcando. Después de ingresar todos los dígitos, se le indica al teclado que se hace la llamada, en este momento el teclado se debe resetear para poder marcar otro número.

   Aunque los dígitos se ingresan como cifras numéricas, conviene manejar el número ingresado completo como un String. P.ej. luego de ingresar un 8, un 5 y un 9 en ese orden, el número ingresado es `"859"`.

   También se le tiene que preguntar al teclado si el número ingresado es válido o no. Tengamos en cuenta reglas sencillas: un número es válido si, o bien tiene 5 cifras, o bien tiene 7 cifras y comienza por `"15"`.

   Este objeto debe entender los siguientes mensajes:

   - `agregarDigito(digito)`
   - `llamar()`
   - `numeroIngresado()`
   - `esNumeroValido()`

   P.ej. si se evalúa la siguiente secuencia
   ```
   tecladoEspingarda.agregarDigito(8)
   tecladoEspingarda.agregarDigito(9)
   tecladoEspingarda.agregarDigito(0)
   tecladoEspingarda.agregarDigito(2)
   tecladoEspingarda.numeroIngresado()
   ```
   el resultado debe ser `"8902"`. Si luego de esto se evalúa
   ```
   tecladoEspingarda.esNumeroValido()
   ```
   el resultado debe ser `false`, porque el número ingresado no tiene 5 cifras. Si después se sigue evaluando así
   ```
   tecladoEspingarda.agregarDigito(3)
   ```
   el `tecladoEspingarda.numeroIngresado()` debe ser `"89023"`, y si se pregunta `tecladoEspingarda.esNumeroValido()` debe responder `true`, porque ahora el número tiene 5 dígitos.

Si después se sigue evaluando
`tecladoEspingarda.llamar()`
`tecladoEspingarda.agregarDigito(7)`
`tecladoEspingarda.agregarDigito(7)`
el `tecladoEspingarda.numeroIngresado()` debe ser `"77"`, al llamar se limpia el número.

7. Agregar al teclado del ejercicio 6:

   - que entienda el mensaje `borrarUltimoDigito()`. P.ej., luego de la primer secuencia del ejercicio 6, si se agrega
     `tecladoEspingarda.borrarUltimoDigito()`
     `tecladoEspingarda.numeroIngresado()`
     el resultado debe ser `"890"`.
   - que entienda el mensajes `cantLlamadas()`. Para esto, cada vez que se hace una llamada, hay que sumar uno a a un contador.

8. Se está pensando en el diseño de un juego que incluye la nave espacial Enterprise. En el juego, esta nave tiene un nivel de potencia de 0 a 100, y un nivel de coraza de 0 a 20. La Enterprise puede

   - encontrarse con una pila atómica, en cuyo caso su potencia aumenta en 25.
   - encontrarse con un escudo, en cuyo caso su nivel de coraza aumenta en 10.
   - recibir un ataque, en este caso se especifican los puntos de fuerza del ataque recibido. La Enterprise "para" el ataque con la coraza, y si la coraza no alcanza, el resto se descuenta de la potencia. P.ej. si la Enterprise con 80 de potencia y 12 de coraza recibe un ataque de 20 puntos de fuerza, puede parar solamente 12 con la coraza, los otros 8 se descuentan de la potencia. La nave debe quedar con 72 de potencia y 0 de coraza. Si la Enterprise no tiene nada de coraza al momento de recibir el ataque, entonces todos los puntos de fuerza del ataque se descuentan de la potencia.

   La potencia y la coraza tienen que mantenerse en los rangos indicados, p.ej. si la Enterprise tien 16 puntos de coraza y se encuentra con un escudo, entonces queda en 20 puntos de coraza, no en 26. Tampoco puede quedar negativa la potencia, a lo sumo queda en 0.

   La Enterprise nace con 50 de potencia y 5 de coraza.

   Implementar este modelo de la Enterprise, que tiene que entender los siguientes mensajes:

   - `potencia()`
   - `coraza()`
   - `encontrarPilaAtomica()`
   - `encontrarEscudo()`
   - `recibirAtaque(puntos)`

P.ej. sobre un REPL recién lanzado, después de esta secuencia
```
enterprise.encontrarPilaAtomica()
enterprise.recibirAtaque(14)
enterprise.encontrarEscudo()
```
la potencia de la Enterprise debe ser 66, y su coraza debe ser 10.

9. Agregar al modelo de la Enterprise del ejercicio 8, la capacidad de entender estos mensajes.

   - `fortalezaDefensiva()`, que es el máximo nivel de ataque que puede resistir, o sea, coraza más potencia.
   - `necesitaFortalecerse()`, tiene que ser `true` si su coraza es 0 y su potencia es menos de 20.
   - `fortalezaOfensiva()`, que corresponde a cuántos puntos de fuerza tendría un ataque de la Enterprise. Se calcula así: si tiene menos de 20 puntos de potencia entonces es 0, si no es (puntos de potencia - 20) / 2.

10. Un taller de diseño de autos quiere estudiar un nuevo prototipo. Para eso, nos piden hacer un modelo concentrado en las características del motor. El prototipo de motor tiene 5 cambios (de primera a quinta), y soporta hasta 5000 RPM.

    La velocidad del auto se calcula así: (rpm / 100) * (0.5 + (cambio / 2)). P.ej. en tercera a 2000 rpm, la velocidad es 20 * (0.5 + 1.5) = 40.

    También nos interesa controlar el consumo. Se parte de una base de 0.05 litros por kilómetro. A este valor se le aplican los siguientes ajustes:

    - Si el motor está a más de 3000 rpm, entonces se multiplica por (rpm - 2500) / 500.
      P.ej., a 3500 rpm hay que multiplicar por 2, a 4000 rpm por 3, etc.
    - Si el motor está en primera, entonces se multiplica por 3.
    - Si el motor está en segunda, entonces se multiplica por 2.

    Los efectos por revoluciones y por cambio se acumulan. P.ej. si el motor está en primera y a 5000 rpm, entonces el consumo es 0.05 * 5 * 3 = 0.75 litros/km.

    El modelo debe entender estos mensajes:

    - `arrancar()`, se pone en primera con 500 rpm.
    - `subirCambio()`
    - `bajarCambio()`
    - `subirRPM(cuantos)`
    - `bajarRPM(cuantos)`
    - `velocidad()`
    - `consumoActualPorKm()`