Guía de Ejercicios

Ingeniería de Software I

Parte 1. Introducción al paradigma, el lenguaje y sus herramientas

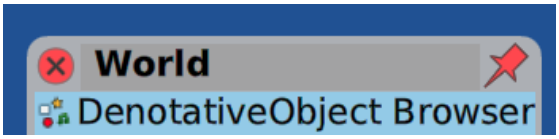
Sección 1. Introducción al paradigma por medio de objetos ejemplares

Instalar y abrir el ambiente de programación, ejecución e inspección Cuis Smalltalk.

Se obtiene de la pagina de la materia: https://www.isw2.com.ar/cuisuniversity

1. Mi primer objeto ejemplar

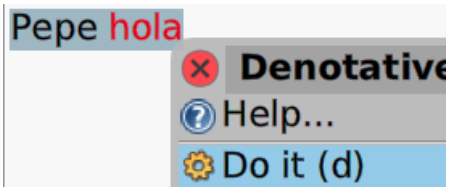
Crea tu primer objeto desde el browser de prototipos "Denotative Object Browser" (lo encontrarás en el menú que se abre al hacer click en el fondo azul).



Elige un nombre para tu primer objeto, por ejemplo ‘Pepe’:

```
ObjectBuilder create: #Pepe
    collaboratorNames: ''
    in: 'Ejercicio 2'
```

Envíe el mensaje *hola* al nuevo objeto y analice qué pasa.



Defina que el mensaje *hola* sea respondido con un texto que diga *chau*. (El *return* se escribe con el caracter '^'.)

Enviar el mensaje hola al objeto y asegurarse que responda del modo esperado.

2. Bill, el zombie

Bill es un zombie que sabe caminar y comer cerebros, y posee una energía medida en 'días de vida'.

Represente a Bill en el ambiente de programación, sabiendo que su energía se modifica de la siguiente manera:

- Cuando camina, por cada kilómetro que realiza consume un día de vida más 3 días de “costo fijo” solo por emprender la caminata.

- Cuando come, adquiere 4 días de vida por cada kilo de cerebro que come. No olvidar considerar que Bill nace con una cierta cantidad de energía (28 días de vida).

Bill sabe responder los siguientes mensajes comer: `unaCantidadDeKilosDeCerebro`, `caminar: unaCantidadDeKilometros`, y `energia` (este último es respondido con la cantidad de días de vida que tiene Bill)

3. Testear y modelar Verdadero, Falso y operaciones lógicas

Represente la verdad y la falsedad con objetos. Hágalo en castellano, dado que en inglés (True y False) ya existen en el ambiente. Llame al objeto que representa la verdad: Verdadero; y aquel que representa la falsedad: Falso.

Ambos objetos deben ser polimórficos respecto de un cierto protocolo.

¿Qué quiere decir esto? 'Protocolo' quiere decir: conjunto de mensajes que un objeto sabe responder. Dos objetos son polimórficos cuando saben responder semánticamente igual a un mismo conjunto de mensajes. 'Semánticamente igual' implica que sus respuestas serán siempre polimórficas respecto a un cierto protocolo.

El protocolo a implementar para Verdadero y Falso es el siguiente:

- `no`
- `y: unBooleano`
- `o: unBooleano`
- `siVerdadero: unaAccionARealizarUIgnorar`
- `siFalso: unaAccionARealizarUIgnorar`

Antes de implementar cada funcionalidad, escriba un test que inicialmente falla a esa funcionalidad. Luego haga pasar correctamente el test implementando lo mínimo necesario en su modelo.

4. Representar los números naturales y las operaciones aritméticas básicas

4.1. Con condicionales

Nos inspiramos en la definición axiomática de Peano de los números naturales (si no la conoce busquela en la web). El matemático Giuseppe Peano define *qué* son los números. A nosotros como programadores nos importa también *cómo* son, por lo cual la definición de Peano es una fuente parcial del dominio. Nos interesará en particular operar aritméticamente con nuestros números.

Llame al primer número I, al segundo número II, al tercero III, al cuarto IIII, al cinco IIIII y así sucesivamente.

Inicialmente concentrense en representar el I y el II.

Vaya definiendo los métodos necesarios para que los números por usted definidos sepan responder los mensajes:

- *`previous`*
- *`next`*
- *`+ anAdder`*
- *`- sustrahend`*
- *`* aMultiplicand`*
- *`/ dividend`*.

Siga ese orden, deje la división para el final que es más difícil.

Cuando al II se le envía el mensaje `next`, automáticamente se debe crear el III y así sucesivamente. Cuando se multipliquen números también automáticamente se deberán crear los números que esa multiplicación abarque, entre ellos el resultado (por ejemplo al evaluar la expresión II * II, se generará el III y el IIII si aún no están representados, y se retornará el IIII).

Para la división, puede definirla de modo que retorne la parte natural (IIII/III retorna I) o bien que solo funcione para divisiones de resultado natural y cuando se pretenda dividir números con resto no sea nulo se genere un error (ej IIII/III genera un error).

Con los números I, II, III, y IIII existentes en el ambiente, cargue el archivo 'Numeros Naturales Tests.st'. Asegúrese que todos los test pasan. Deberá incorporar algunos mensajes de error en el modelo para que coincidan con los definidos en algunos test (el 5 y el 12).

4.2. Eliminar condicionales if de la suma, la resta y la multiplicación

Intente que no haya condicionales `ifTrue` o `ifFalse` en la definición de estas operaciones. Debe agregar más mensajes a los números que les permitan decidir, según quien recibe el mensaje, qué hacer.

Analice qué dificultades surgen cuando se intenta reemplazar el if en la división.

4.3. Transformar los números "con palitos" en números romanos

Al crear el número siguiente de todo número, la construcción del nombre del próximo debe estar en un solo lugar, en un método similar cuya definición sea similar a:

```
nameOfNext

    ↑ self name, 'I'
```

Debe modificar esa definición agregando ifs. Siga para ello el siguiente ejemplo, donde para el número III no deja de ser IIII el siguiente y pasa a ser IV.

```
nameOfNext

    (self name endsWith: 'III') ifTrue:[↑(self name withoutSuffix: 'III'), 'IV'].
    ↑self name, 'I'
```