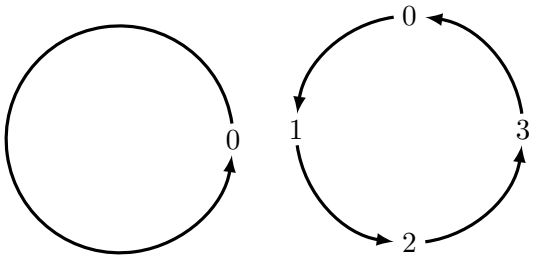
Trabajo Práctico 2
Programación Orientada a Objetos

Paradigmas de Lenguajes de Programación — 2do cuat. 2019

Fecha de entrega: Jueves 31 de Octubre

1. Introducción

Durante este taller modelaremos anillos como objetos en Javascript. Un *anillo* es lo que en Teoría de Grafos se conoce como un ciclo simple. Es decir, un conjunto de vértices y arcos que forman un camino cerrado que no repite arcos ni nodos a excepción del vértice inicial.



Notar que un anillo de un solo elemento se tiene como siguiente a sí mismo.

Representación utilizada

En este caso, definiremos cada elemento del anillo como un objeto que sabe responder los siguientes mensajes:

1. ***dato***: que retorna el dato del nodo
2. ***siguiente***: que retorna el siguiente elemento del anillo

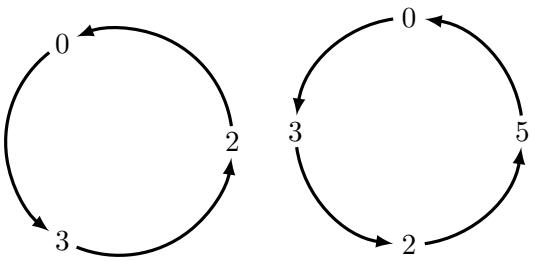
No utilizaremos una representación particular para el concepto de anillo, sino que trataremos de manera indistinta a un anillo y a los nodos que lo componen (cada nodo puede verse como un anillo cuyo ciclo empieza con él).

2. Ejercicios

Ejercicio 1

Definir el objeto `AnilloCero`, que representa a un anillo con un único elemento de valor 0 y cuyo siguiente elemento es él mismo.

Este objeto debe responder además el mensaje `agregar(d)` que, dado un nuevo dato `d`, agrega un nuevo nodo con dato `d` como su elemento siguiente dentro del anillo (y retorna el nodo receptor modificado). El nuevo nodo tiene que poder responder a los mismos mensajes que el original, y se debe mantener la estructura de anillo.



Ejemplo de anillo antes y después de enviar el mensaje `agregar`(5) al nodo con valor 2.

Ejercicio 2

Redefinir en anilloCero el método correspondiente al mensaje `toString()` para que el String retornado muestre el camino que define el anillo empezando por sí mismo. Por ejemplo para el siguiente caso:

```
anilloCero.agregar(2).agregar(1).toString()
> "0 ⇝ 1 ⇝ 2"
```

(Se permite cambiar el estilo de la flecha.)

Ejercicio 3

Definir la función constructora `Anillo(d)` que permita construir un anillo con un único elemento de dato **d**. Los anillos construidos mediante esta función deben poseer los atributos `dato` y `siguiente`, y responder los mensajes `agregar` y `toString` de la misma manera que los construidos anteriormente (sin repetir el código de estos mensajes).

Ejercicio 4

Definir los métodos necesarios para que todos los anillos construidos mediante la función constructora del ejercicio anterior puedan responder los siguientes mensajes:

1. `map(f)` que, dada una función `f`, devuelve un nuevo anillo con la misma estructura que el receptor pero cuyos nodos tienen como valor el resultado de aplicar la función `f` a los elementos del anillo original.

2. `copiar()`, que devuelve una copia del anillo receptor (se debe copiar el anillo completo, no solamente el nodo receptor).

3. `cantidad()`, que devuelve la longitud del ciclo (es decir, su cantidad de elementos).

Ejercicio 5

Definir el método para que los anillos respondan el mensaje `ponerAnteriores()`, que transforma el anillo receptor en un anillo bidireccional. Es decir, define para cada uno de sus elementos el atributo `anterior` que apunta al elemento inmediatamente anterior dentro del

anillo. Además, este nuevo invariante debe mantenerse cuando un anillo bidireccional recibe el mensaje `agregar()`. En otras palabras, si se agrega un nuevo elemento a un anillo que haya recibido el mensaje `ponerAnteriores()`, este nuevo elemento también debe contar con el atributo `anterior`, y el ciclo formado por este nuevo atributo debe actualizarse adecuadamente, al igual que el ciclo habitual marcado por el atributo `siguiente`.

Importante: los anillos que no hayan recibido el mensaje `ponerAnteriores()` deben seguir comportándose como antes.

3. Pautas de Entrega

Se debe entregar el código impreso con la implementación de los ejercicios. Cada función o método asociado a los ejercicios debe contar con ejemplos que muestren que exhibe la funcionalidad solicitada. Además, se debe enviar un e-mail conteniendo el código fuente en JavaScript a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-POO]` seguido inmediatamente del nombre del grupo.
- El código JavaScript debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto con nombre `taller.js` (puede adjuntarse un **.zip** o un **.tar.gz**).

El código debe poder ejecutarse en los navegadores de los laboratorios. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté **adecuadamente** comentado (son comentarios adecuados los que ayudan a entender lo que no es evidente o explican decisiones tomadas; no son adecuadas las traducciones al castellano del código). Los objetivos a evaluar son:

- Corrección.
- Declaratividad.
- Prolijidad: evitar repetir código innecesariamente y usar adecuadamente los métodos previamente definidos.

Importante: se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

Referencias del lenguaje JavaScript

Como principales referencias del lenguaje de programación JavaScript, mencionaremos:

- **W3Schools JavaScript Reference:** disponible online en: https://www.w3schools.com/jsref/default.asp.
- **MDN Web Docs: Mozilla - Referencia de JavaScript**: disponible online en: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia.