PRACTICO N° 7

- *En todos los problemas considere la importancia de interpretar correctamente los enunciados, en particular cuando incluye notación simbólica.*
- *En todas las soluciones considere la importancia de diseñar algoritmos correctos, eficientes y legibles seleccionando estructuras de control adecuadas.*
- *En todas las implementaciones puede agregar métodos para favorecer la modularización.*

EJERCICIO 1. Un profesor desea automatizar el mantenimiento de la planilla de notas de sus alumnos.
Todas las notas están dentro del rango 0 a 10, la nota mínima de aprobación es 4. Implemente la clase Notas de acuerdo al siguiente modelo, establezca casos de prueba adecuados y escriba un tester que verifique la clase:

Notas
<<atributos de instancia>> examen: entero []
<<constructor>> Notas (cantAlu : entero) <<comandos>> establecerNota (p, n : entero) <<consultas>> obtenerNota (p : entero) : entero promedio () : real promedioAprobados( ) : real cantidadAprobados():entero mitadAprobados():boolean

establecerNota(p,n)
Requiere 0 <= p < cantAlu
0 <=n<=10

obtenerNota(p):entero
Requiere 0 <= p < cantAlu

promedio () : real retorna el promedio general
promedioAprobados( ) : real retorna el promedio de los alumnos que aprobaron
cantidadAprobados():entero retorna la cantidad de alumnos que han aprobado
mitadAprobados():boolean retorna true si al menos la mitad de los alumnos han aprobado

EJERCICIO 2. Implemente la clase micro modelada en el siguiente diagrama:

Micro
numero: entero destino:String precioPasaje: real asientos [] boolean
<<Constructor>> Micro(n,cantAsientos:entero,d:String,p:real) <<Comandos>> reservar(a:entero) cancelar(a:entero) actualizarPrecio(p:real) <<Consultas>> estaDisponible(a:entero):boolean cantAsientos():entero obtenerNumero():entero obtenerDestino():String obtenerPrecioPasaje():real

reservar(a) y cancelar(a)
Requiere 1 <= a <=cantAsientos()

El constructor crea un micro con *cantAsientos* asientos, inicialmente todos están disponibles, de modo que todas las componentes del arreglo son true. Los métodos reservar y cancelar establecen un mapeo para convertir el número de asiento en subíndice.

EJERCICIO 3. La clase NumerosEnteros permite representar una secuencia de números enteros mediante un arreglo de acuerdo a lo que especifica el diagrama:

NumerosEnteros
<<atributos de instancia>> S : entero[]
<<constructores>> NumerosEnteros (n : entero) <<comandos>> intercambiar(p1,p2:entero) reemplazar(v1,v2:entero) reemplazarPrimera (v1,v2 : entero) reemplazarUltima(v1, v2 : entero) establecerEntero(pos, elem:entero) <<consultas>> obtenerEntero(pos:entero):entero toString():String

El constructor crea un arreglo de n componentes y lo inicializa con n números generados al azar.
intercambiar(p1,p2:entero): *Intercambia* los elementos de las posiciones p1 y p2.
reemplazar(v1,v2:entero): Reemplaza **todas** las apariciones del elemento v1 por el elemento v2.
reemplazarPrimera (v1,v2 : entero): Reemplaza la **primera** aparición del elemento v1 por el elemento v2.
reemplazarUltima(v1, v2 : entero): Reemplaza la **última** aparición del elemento v1 por el elemento v2.

a. Implemente la clase NumerosEnteros
b. Establezca casos de prueba adecuados para verificar cada servicio.
c. Escriba una clase tester que permita verificar los servicios con casos de prueba elegidos. Utilice el método **toString()** para poder mostrar los valores en consola.

EJERCICIO 4. Dado el siguiente diagrama:

Palabra
<<atributos de instancia>> S : char[]
<<constructores>> Palabra (s: String) <<consultas>> longitudPalabra(): entero esSufijo(c:Palabra):boolean esPrefijo(c:Palabra):boolean esIgual(c:Palabra):boolean esMenor(c:Palabra):boolean
El constructor requiere que la cadena s esté conformada únicamente por letras

a. Proponga un planteo recursivo para decidir si una palabra c es sufijo de la palabra que recibe el mensaje esSufijo (Ejemplo: **AL** es sufijo de "RAM**AL**" pero no de "RAMA").
b. Proponga un planteo recursivo para decidir si una palabra c es prefijo de la palabra que recibe el mensaje esPrefijo (Ejemplo: **SOL** es prefijo de "**SOL**AR" pero no de "SOPA").
c. Implemente la clase Palabra, observe que los métodos esSufijo y esPrefijo deben ser consistentes con los planteos.

EJERCICIO 5. Dado el siguiente diagrama:

Oracion
<<atributos de instancia>> **S : char[]**
<<constructores>> Oracion (s: String) <<consultas>> esPalabra(pos:entero):boolean esPrimeraPalabra(pos:entero):boolean esUltimaPalabra(pos:entero):boolean longitud(): entero contarPalabras (): entero contarCortas(): entero masLarga():entero primerPalabra():Palabra histograma():NumerosEnteros

- ***esPalabra(pos:entero):boolean*** *retorna verdadero siempre que en la posición pos haya una letra.*
- ***esPrimeraPalabra(pos:entero):boolean*** *retorna verdadero siempre que en la posición pos haya una letra y sea la primera de una palabra.*
- ***esUltimaPalabra(pos:entero):boolean*** *retorna verdadero siempre que en la posición pos haya una letra y sea la ultima de una palabra.*
- ***contarCortas(): entero*** cuenta cuántas palabras de 3 letras o menos contiene la oración.
- ***masLarga():entero*** retorna la longitud de la palabra más larga
- ***histograma():NumerosEnteros*** retorna un objeto de clase NumerosEnteros que registra la cantidad de apariciones de cada letra del alfabeto en la oración que recibe el mensaje. Observe que la cantidad de elementos de NumerosEnteros corresponde a la cantidad de letras del alfabeto.

a. Implemente y verifique la clase Oracion modelada en el diagrama. Se asume que una oración está formada por ***palabras separadas por un blanco.*** La oración contiene al menos una palabra y ***termina siempre con un blanco.***
b. Implemente y verifique la clase Oracion modelada en el diagrama. Se asume que una oración está formada por ***palabras separadas por un blanco o un signo seguido de un blanco. Las palabras contienen solo letras y los signos son coma o punto y coma.*** La oración contiene al menos una palabra y ***termina siempre con punto.***
c. Implemente y verifique la clase Oracion considerando que ***las palabras pueden estar separadas por uno o más blancos o por un signo seguido de uno o más blancos.*** La oración contiene al menos una palabra y ***termina siempre con punto.***

PISTA: CONSIDERE USAR LAS CONSULTAS ESPALABRA, ESPRIMERAPALABRA Y ESULTIMAPALABRA SIEMPRE QUE SEA POSIBLE PARA MODULARIZAR MEJOR EL CÓDIGO.

EJERCICIO 6. Dado el siguiente diagrama:

OracionS
<<atributos de instancia>> **S : String**
<<constructores>> OracionS (s: String) <<consultas>> longitud(): entero contarPalabras (): entero contarCortas(): entero masLarga():entero

- ***contarCortas(): entero*** cuenta cuántas palabras de 3 letras o menos contiene la oración.
- ***masLarga():entero*** retorna la longitud de la palabra más larga

Implemente y verifique la clase OracionS modelada en el diagrama. Se asume que una oración está formada por palabras separadas por un blanco. La oración contiene al menos una palabra y termina siempre con un blanco.

EJERCICIO 7. Extender la clase NumerosEnteros implementando métodos:

NumerosEnteros
<<atributos de instancia>> ...
<<constructores>> ... <<comandos>> copy(ne:NumerosEnteros) <<consultas>> clone():NumerosEnteros todosImpares():NumerosEnteros contarCoincidencias (ne:NumerosEnteros):entero longitudMayorSecuenciaCreciente(): entero

- ***todosImpares():NumerosEnteros*** genera un objeto de clase NumerosEnteros con los elementos del objeto que recibe el mensaje que son impares.
- ***contarCoincidencias(ne: NumerosEnteros)****:entero* calcula cuántos elementos del arreglo asociado al objeto que recibe el mensaje, coinciden en posición y contenido con el arreglo asociado al objeto que se pasa como parámetro. Por ejemplo (1,3,**2**,0,**3**,7) y (0,9,**2**,1,**3**,6) tiene 2 coincidencias.
- ***longitudMayorSecuenciaCreciente():entero*** calcula cuál es la longitud de la mayor secuencia de elementos creciente. Por ejemplo (2,1,7,8,3,1,**0,5,9,11,17**,13,14) tiene tres secuencias crecientes, la más larga tiene longitud 5.

EJERCICIO 8. Extienda la implementación de la clase Oración con los servicios reducirBlancos y ajustar.

- ***reducirBlancos()*** reemplaza las secuencias de dos o más blancos por un solo blanco.
- ***ajustar(n:entero):Oracion*** si n es mayor que la cantidad de caracteres de la oración, genera una nueva oración con las mismas palabras, pero separadas por la cantidad de blancos que corresponda agregar para alcanzar n caracteres en total. Los blancos se distribuyen entre palabras en forma balanceada.

EJERCICIO 9. Extienda la clase NumerosEnteros y su tester agregando

i. Un constructor que inicialice el arreglo con valores leídos de un archivo secuencial.
ii. Implementaciones recursivas de los siguientes servicios:

a. boolean tieneReflejo()

Dado un objeto de clase NumerosEnteros que representa una secuencia **S** = a_1 a_2 .. a_k b_k .. b_2 b_1 de **longitud par** (2k) se dice que "tiene algún reflejo" si existe i en [1..k] tal que a_i=b_i. Por ejemplo:

- **S** = 1 **20** 3 5 **20** 12 (longitud=6) "tiene algún reflejo" (ya que a_2=b_2=20).
- **S** = 2 4 3 1 (longitud=4) **NO** "tiene algún reflejo"
- Si **S** es vacía, **NO** "tiene algún reflejo".

Dada la consulta:

```
public boolean tieneReflejo() {
   return tieneReflejoAux(0, S.length-1);  }
```

Implemente un método recursivo tieneReflejoAux (i,f) que decida si la secuencia comprendida entre las posiciones i y f del arreglo, tiene algún reflejo.

```
private boolean tieneReflejoAux(int i, int f) { ... }
```

b. ***int sumaDeProductos()***
Dado un objeto de clase NumerosEnteros que representa una secuencia S= d_1 d_2 d_3 ... d_n computa la suma de productos de los números definida como:
$d_1 * d_n + d_2 * d_{n-1} + \ldots + d_{n/2} * d_{n/2+1}$ cuando **n** es par y
$d_1 * d_n + d_2 * d_{n-1} + \ldots + d_{n/2} * d_{n/2+2} + d_{n/2+1}$ cuando **n** es impar.

c. boolean esCreciente()
Dado un objeto de clase NumerosEnteros que representa una secuencia S= d_1 d_2 d_3 ... d_n, retorna verdadero sí y solo sí $d_i < d_{i+1}$ para 1<=i<n

d. void invertir()
Dado un objeto de clase NumerosEnteros que representa una secuencia S= d_1 d_2 d_3 ... d_n modifica el estado interno del objeto de modo que represente a la secuencia S´= d_n ... $d_3 d_2 d_1$.

e. void primeroOrdenado()
Modifica el estado interno del arreglo de modo que el primer elemento (pivote) se reacomoda de forma tal que a su izquierda quedan los menores que él y a su derecha los mayores que él. Utilice recursividad cruzada y considere que el pivote se tiene que comparar una sola vez con cada uno de los demás elementos.

Objetivos del práctico
Recorridos iterativos y recursivos. Recorridos exhaustivos y no exhaustivos.
La clase tester. Correctitud, eficiencia y legibilidad