Programación I

Práctica N°0 – Introducción, repaso y recursividad

Lectura de datos ingresados por el usuario

Desde Java, para leer datos ingresados por el usuario es necesario importar la biblioteca `java.util.Scanner` usando la instrucción:

```
import java.util.Scanner;
```

y luego utilizar la siguiente instrucción en el `main` (o desde donde se quieran leer los datos):

```
Scanner scan = new Scanner(System.in);
```

Una vez hecho esto, se puede utilizar la variable `scan` como un lector de datos del teclado de la siguiente manera:

- `scan.nextLine()`: lee una cadena de caracteres finalizada con la tecla `Enter`. La cadena puede o no contener espacios.
- `scan.next()`: lee una palabra ingresada por el usuario. Si el usuario ingresó más de una palabra sólo se leerá la primera de ellas, quedando las demás pendientes de lectura.
- `scan.nextInt()`: lee un valor numérico de tipo entero.
- `scan.nextFloat()`: lee un valor numérico de tipo decimal.

1. Variables, expresiones y tipos

Ejercicio 1

Escribir el programa "¡Hola, mundo!".

Ejercicio 2

Escribir un programa que te pregunte tu nombre y a continuación imprima un saludo del estilo "Hola *nombre*".

Ejercicio 3

Escribir un programa que te pregunte por dos números, y a continuación imprima un mensaje del estilo "La suma es: " y el valor de la suma de ambos números. .

Ejercicio 4

Imprimir desde Java las siguientes expresiones e interpretar el valor que arrojan.

- 1/2
- 1.0/2.0
- 1.0/2
- 1/2.0

- "1"/"2"
- 1+2
- "1"+"2"
- 16/2*4
- 16/(2*4)

Ejercicio 5

Escribir un programa que te pregunte por dos números, y a continuación imprima un mensaje del estilo "El promedio es: " y el valor del promedio de ambos números.

2. Métodos y condicionales

Ejercicio 6

Escribir un método **static void** `imprimirSuma`(**int** `a`, **int** `b`) que al igual que el ejercicio 3 imprima la suma de los dos parámetros. Modificar el programa de dicho ejercicio para que utilice este método.

Ejercicio 7

Análogamente al ejercicio anterior, escribir un método **static void** `imprimirPromedio`(**int** `a`, **int** `b`) que imprima el promedio de los dos parámetros.

Ejercicio 8

Escribir un método **static void** `ponerNota`(**double** `x`, **double** `y`) que toma dos números decimales y los promedia. En caso que el promedio sea mayor o igual a 7, deberá imprimir `''Promocionado''`, si es mayor o igual a 4 pero menor que 7, imprime `''Aprobado''` y si es menor que 4 imprime `''Debe recuperar''`. Probarla llamándola desde el main con distintos números. Luego, pedirle ambos números al usuario (usando `nextFloat`() del `Scanner`) para pasárselos a `ponerNota`.

Ejercicio 9

Escribir un método **static void** `imprimirFecha`(**int** `dia`, **int** `mes`, **int** `anio`) que imprime la fecha pasada como parámetro en formato del estilo "5 de Julio de 2030".

3. Métodos con resultados e iteración

Ejercicio 10

Escribir un método **static int** `sumatoria`(**int** `n`) que devuelve la sumatoria de los números desde 1 hasta n.

Ejercicio 11

Escribir un método **static int** `sumatoriaPares`(**int** `n`) que devuelve la sumatoria de los números pares desde 2 hasta n.

Ejercicio 12

Escribir un método **static double** `potencia`(**double** `x`, **int** `a`) que toma un número racional x y un entero a y calcula x^a.

Ejercicio 13

Escribir un método **static double** `factorial`(**int** `n`) que toma un entero positivo n y calcula $n!$ (el factorial de n) que se define como el producto de todos los naturales desde 1 hasta n. Por ejemplo $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$. Ojo: por definición, el factorial de 0, es 1 ($0! = 1$).

Ejercicio 14

Escribir un método **static int** `cantCifras`(**int** `n`) que devuelve la cantidad de cifras de n. Probarlo adecuadamente llamándola desde el main.

Ejercicio 15

Escribir un método **static boolean** `esDivisible`(**int** `n`, **int** `m`) que devuelve `true` si n es divisible por m y `false` en caso contrario. Probarlo adecuadamente llamándola desde el *main*.

4. Cadenas

Ejercicio 16

a) Escribir un programa que pida por pantalla un texto y a continuación lo imprima de atrás para adelante. Para obtener las letras de una cadena de caracteres pueden utilizar el método `charAt` de `String`. Por ejemplo, `cadena.charAt(0)` devuelve el primer caracter del String `cadena`.
b) Mover el código que imprime la cadena al revés a un método **static void** `imprimirReversa`(`String cadena`)
c) Escribir un método **static** `String reversa`(`String cadena`) que dado un String, devuelve otro String con los caracteres invertidos. Por ejemplo, `reversa`(“`hola`”) debería devolver el String “`aloh`”.
d) Modificar el método `imprimirReversa` para que utilice el método definido en el punto anterior.

Ejercicio 17

Escribir un método **static int** `cantidadApariciones`(`String s`, **char** `c`) que dada una cadena y un caracter, cuenta la cantidad de veces que aparece `c` en `s`.

Ejercicio 18

Escribir un método **static int** `cantidadVocales`(`String s`) que dada una cadena que contiene sólo letras minúsculas sin acentuar, devuelve la cantidad de vocales en dicha cadena. Nota: se puede utilizar el método definido en el ejercicio anterior.

Ejercicio 19

Una palabra se dice que es “abecedaria” si las letras en la palabra aparecen en orden alfabético. Por ejemplo, las siguientes son todas palabras abecedarias del idioma castellano.

adiós, afín, afinó, ágil, bello, celos, cenó, chinos dijo, dimos, dios, fijos, finos, hijos, hilos, himno

1. Describí un algoritmo para decidir si una palabra dada es abecedaria, asumiendo que la misma contiene sólo letras minúsculas.
2. Implementar el algoritmo en un método **static boolean** `esAbecedaria`(`String s`).
3. ¿Funciona el algoritmo si le pasamos como parámetro `‘‘ágil’’`? En caso negativo, ¿por qué te parece que puede ser? ¿Cómo lo solucionarías?

Ejercicio 20

Escribir el método **static boolean** `esCapicua`(`String s`) que dada una cadena, devuelve `true` si la cadena es igual de atrás hacia adelante o de adelante hacia atrás. En caso contrario, devuelve `false`.

Ejercicio 21

Escribir un método **static boolean** `esSinRepetidos`(`String s`) que dada una cadena, devuelve `true` si no hay letras repetidas en la cadena. En caso contrario, devuelve `false`. No utilizar el método del ejercicio 22.

Ejercicio 22

Escribir un método **static** `String sinRepetidos`(`String s`) que dada una cadena, devuelve una nueva cadena donde cada uno de los caracteres que aparecían en `s`, aparecen sólo una vez. Se debe mantener la posición relativa de los caracteres: para aquellos que se encuentren repetidos puede conservarse cualquiera de sus apariciones. Por ejemplo, para la palabra `‘‘casos’’` puede devolver `‘‘caso’’` o `‘‘caos’’`, conservando la primera o la segunda letra `s` respectivamente.

Ejercicio 23

Para el desarrollo de un sistema generador de juegos de palabras cruzadas, se necesita programar la siguiente función. Dadas 3 cadenas, `a`, `b` y `c`, se quiere saber si puede colocarse `b` de manera vertical de modo que `a` y `c` se coloquen de manera horizontal atravezando a `b`. Se necesita también que `a` esté más arriba que `c` y tengan al menos un renglón de diferencia. Por ejemplo, `a` = ”JUGO”, `b` = ”BUENO”, `c` = ”ANANA” pueden colocarse como muestra la Figura 1, por lo tanto el método debe devolver `Verdadero`. En cambio, `a` = ”JUGO”, `b` = ”FEO”, `c` = ”ANANA” no pueden colocarse, y por ende debería devolver `Falso`. El encabezado del método debe ser **static boolean** `puedenColocarse`(`String a`, `String b`, `String c`).

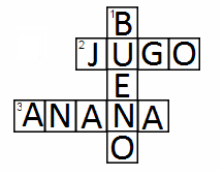

Figura 1: Ejemplo

Ejercicio 24

Decimos que una cadena de caracteres es un *doblete capicúa* si la cadena es la concatenación de dos cadenas capicúas. Por ejemplo:

- "neuquenoro" es doblete capicúa ya que es la concatenación de "neuqen" y "oro".
- "banana" es doblete capicúa ya que es la concatenación de "b" y "anana", ambas capicúa. Lo mismo sucede con "eter" ya que es la concatenación de "ete" y "r", ambas capicúa.
- "sanas" es doblete capicúa ya que es la concatenación de "sanas" y "".
- "nunca" no es doblete capicúa ya que no hay manera de formarla concatenando dos capicúas.

Escribir un método **static boolean** `esDoblete`(`String s`) que devuelve `true` cuando la cadena es doblete capicúa y `false` en caso contrario.

5. Arreglos

Ejercicio 25

Escribir un método **static int** `maximo`(**int**[] `a`) que dado un arreglo de enteros no vacío, devuelve el valor más alto que aparece.

Ejercicio 26

Escribir un método **static int** `maximoIndice`(**int**[] `a`) que dado un arreglo de enteros no vacío, devuelve el índice del valor más alto que aparece.

Ejercicio 27

Escribir un método **static int** `suma`(**int**[] `a`) que dado un arreglo de enteros, devuelve el valor de la suma de todos sus elementos.

Ejercicio 28

Escribir un método **static boolean** `estaOrdenado`(**int**[] `a`) que dado un arreglo de enteros, devuelve verdadero si el arreglo está ordenado crecientemente de menor a mayor, y falso en caso contrario.

Ejercicio 29

Escribir un método **static double** `promedio`(**double**[] `a`) que dado un arreglo de números con coma, devuelve el valor del promedio de todos los elementos.

6. Recursividad

Ejercicio 30

Escribir las versiones recursivas de los siguientes métodos de la Sección 3:

a) `sumatoria`: **static int** `sumatoriaRec`(**int** `n`)
b) `sumatoriaPares`: **static int** `sumatoriaParesRec`(**int** `n`)
c) `potencia`: **static int** `potenciaRec`(**double** `x`, **int** `n`)
d) `factorial`: **static int** `factorialRec`(**int** `n`)

Ejercicio 31

La sucesión de Fibonacci es una sucesión de números naturales que describe, por ejemplo, el número de individuos en una población de conejos tras varias generaciones. Esta sucesión tiene la particularidad de estar presente en muchos elementos de la naturaleza, y que a medida que se aproxima al infinito, el cociente entre dos elementos consecutivos, se aproxima a la proporción áurea. Los numeros de la sucesión se obtienen de la siguiente manera:

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_n &= f_{n-2} + f_{n-1} \end{aligned}$$

Los primeros números de la sucesión serían entonces: $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

Implementar el método que devuelve el n-ésimo elemento de la sucesión:

a) usando recursividad, con la siguiente signatura: **static int** `fibrec`(**int** `n`)
b) usando un ciclo, con la siguiente signatura: **static int** `fibiter`(**int** `n`)

Responder:

a) Comparar los tiempos entre una implementación y otra. ¿Cuál es el término más grande que puede calcular cada una de las implementaciones en un tiempo menor a 10 segundos? ¿Hay diferencia? ¿Te imaginás por qué?
b) ¿Cuánto vale f_{47}? Interpretá este resultado.

Ejercicio 32

La sucesión de Collatz se define de la siguiente manera. Se comienza del número n y se prosigue así:

- Si n es par, entonces el siguiente número es $n/2$
- Si n es impar, entonces el siguiente número es $3 * n + 1$
- Cuando n vale 1, no hay siguiente número.

Escribir un método **static void** `collatz`(**int** `n`) que toma un natural n e imprime, en líneas separadas, los números de la sucesión.

Ejercicio 33

(Ejercicio 5.10 del libro) El siguiente algoritmo es conocido como el algoritmo de Euclides ya que aparece en los *Elementos* de Euclides (Libro 7, año 300 a.c.). Es probablemente el algoritmo no trivial más antiguo que se conoce.

El algoritmo se basa en la observación de que, si r es el resto de dividir a por b, entonces los divisores en común entre a y b son los mismos que los divisores en común entre b y r. Así, podemos usar la ecuación

$$mcd(a, b) = mcd(b, r)$$

para reducir reiteradamente el problema de calcular un MCD(máximo común divisor) al problema de calcular el MCD de pares de enteros cada vez más pequeños. Por ejemplo:

$$mcd(36, 20) = mcd(20, 16) = mcd(16, 4) = mcd(4, 0) = 4$$

implica que el MCD entre 36 y 20 es 4. Se puede demostrar que para cualquier par de números iniciales, esta reducción repetida eventualmente genera un par en el cual el segundo número es 0. Así, el MCD es el otro número del par.

Escribir, **utilizando recursividad**, un método **int** `mcd`(**int** `a`, **int** `b`) que calcula el máximo común divisor entre a y b.

Ejercicio 34

Escribir una función que tome una cadena como parámetro y la imprima por consola intercalando un '∗' entre cada letra. Por ejemplo, si la función toma la cadena "hola" como parámetro, deberá imprimir "h∗o∗l∗a". Se puede dar por hecha la función **static** `String resto`(`String s`) que devuelve una cadena igual a `s` pero sin su primer caracter.

Ejercicio 35

Escribir una función que tome una cadena como parámetro y devuelva otra sin caracteres repetidos contiguos. Por ejemplo,

- si la función toma la cadena "pollos" como parámetro, deberá devolver "polos".
- Si la función toma "gggeeeeeeudddddduu", deberá devolver "geudu".

Se deberá usar la función **static** `String resto`(`String s`) que devuelve una cadena igual a `s` pero sin su primer caracter.

Ejercicio 36

Escribir el método recursivo **static int** `prodCifras`(**int** `n`), que devuelve el producto de las cifras distintas a 0 de un número entero positivo. Por ejemplo:

- `prodCifras`(2034) debe devolver 24.
- `prodCifras`(52) debe devolver 10.
- `prodCifras`(9) debe devolver 9.
- `prodCifras`(11020) debe devolver 2.

Ejercicio 37

Escribir el método recursivo **static** `String estaPrimera`(`String s1`, `String s2`) que toma dos String `s1` y `s2` y devuelve el String que está primero en el diccionario. Por ejemplo:

- `estaPrimera`("`piedra`", "`cantor`") debe devolver "`cantor`".
- `estaPrimera`("`hielo`", "`holanda`") debe devolver "`hielo`".
- `estaPrimera`("`candelabro`", "`canario`") debe devolver "`canario`".

Ejercicio 38

Escribir una función recursiva **static** `String rotacion`(`String s`) que devuelve el String que resulta de rotar `n` posiciones a la izquierda los caracteres de `s`. Por ejemplo:

- `rotacion`("`abcdefghi`", 1) debe devolver "`bcdefghia`".
- `rotacion`("`abcdefghi`", 3) debe devolver "`defghiabc`".
- `rotacion`("`abc`", 3) debe devolver "`abc`".
- `rotacion`("`abc`", 5) debe devolver "`cab`".