



## Práctica 1

### Estructuras de control y estructuras de datos básicas en Java. Recursión

**Nota:**

- Cree un proyecto llamado "AYED" para guardar todas las clases que implemente durante la cursada.
- Implemente cada ejercicio en un paquete que contenga los números del TP y del ejercicio. Ejemplo tp1.ejercicio3
- Para resolver esta práctica haremos uso de las estructuras de datos array y "listas" provistas por Java: [ArrayList](#) y [LinkedList](#).

1. Escriba tres **métodos de clase (static)** que reciban por parámetro dos números enteros (tipo **int**) a y b e impriman todos los números enteros comprendidos entre a; b (inclusive), uno por cada línea en la salida estándar. Para ello, dentro de una nueva clase escriba un método por cada uno de los siguientes incisos:
  - a. Que realice lo pedido con un **for**.
  - b. Que realice lo pedido con un **while**.
  - c. Que realice lo pedido **sin utilizar estructuras de control iterativas** (*for, while, do while*).

Por último, escriba en el método de clase **main** el llamado a cada uno de los métodos creados, con valores de ejemplo. En su computadora, **ejecute el programa** y verifique que se cumple con lo pedido.

2. Escriba un método de clase que dado un número **n** devuelva un nuevo arreglo de tamaño **n** con los **n** primeros múltiplos enteros de **n** mayores o iguales que 1.

Ejemplo:  $f(5) = [5; 10; 15; 20; 25]$ ;  $f(k) = \{n*k \text{ donde } k : 1..k\}$

Agregue al programa la posibilidad de probar con distintos valores de **n** ingresandolos por teclado, mediante el uso de *System.in*. La clase **Scanner** permite leer de forma sencilla valores de entrada.

**Ayuda:** Como ejemplo de uso, para contar la cantidad de números leídos hasta el primer 42 se puede hacer:

```
import java.util.Scanner;

public class Contador {

    public static int contar42 ( ) {
        Scanner s = new Scanner(System.in) ;
        int cantidad = 1 ;
        while (s.nextInt() != 42) {
            cantidad++;
        }
        return cantidad;
    }
}
```



## Algoritmos y Estructuras de Datos

### 3. Creación de instancias mediante el uso del operador **new**

- a. Cree una clase llamada **Estudiante** con los atributos especificados abajo y sus correspondientes métodos *getters* y *setters* (*haga uso de las facilidades que brinda eclipse*)
  - nombre
  - apellido
  - comision
  - email
  - direccion
- b. Cree una clase llamada **Profesor** con los atributos especificados abajo y sus correspondientes métodos *getters* y *setters* (*haga uso de las facilidades que brinda eclipse*)
  - nombre
  - apellido
  - email
  - catedra
  - facultad
- c. Agregue un método de instancia llamado **tusDatos()** en la clase **Estudiante** y en la clase **Profesor**, que retorne un **String** con los datos de los atributos de las mismas. Para acceder a los valores de los atributos **utilice los getters previamente definidos**.
- d. Escriba una clase llamada **Test** con el método **main**, el cual cree un arreglo con **2** objetos **Estudiante**, otro arreglo con **3** objetos **Profesor**, y luego recorra ambos arreglos imprimiendo los valores obtenidos mediante el método **tusDatos()**. Recuerde asignar los valores de los atributos de los objetos **Estudiante** y **Profesor** invocando los respectivos métodos *setters*.
- e. Agregue dos breakpoints, uno en la línea donde itera sobre los estudiantes y otro en la línea donde itera sobre los profesores
- f. Ejecute la clase **Test** en modo debug y avance **paso a paso** visualizando si el estudiante o el profesor recuperado es lo esperado.

### 4. Pasaje de parámetros en Java

- a. Sin ejecutar el programa en su computadora, sólo analizándolo, indique qué imprime el siguiente código.

```
public class SwapValores {
    public static void swap1 (int x, int y) {
        if (x < y) {
            int tmp = x ;
            x = y ;
            y = tmp;
        }
    }

    public static void swap2 (Integer x, Integer y) {
        if (x < y) {
            int tmp = x ;
            x = y ;
            y = tmp;
        }
    }
}
```



## Algoritmos y Estructuras de Datos

```
public static void main(String[] args) {  
    int a = 1, b = 2;  
    Integer c = 3, d = 4;  
    swap1(a,b);  
    swap2(c,d);  
    System.out.println("a=" + a + " b=" + b) ;  
    System.out.println("c=" + c + " d=" + d) ;  
}  
}
```

- b. Ejecute el ejercicio en su computadora, y compare su resultado con lo esperado en el inciso anterior.
  - c. Inserte un breakpoint en las líneas donde se indica: `y = tmp` y ejecute en modo debug ¿los valores que adoptan las variables `x`, `y` coinciden con los valores impresos por consola?
5. Dado un arreglo de valores tipo **entero** se desea calcular el valor máximo, mínimo, y promedio en un único método. Escriba tres métodos de clase, donde respectivamente:
- a. Devuelva lo pedido por el mecanismo de retorno de un método en Java (`"return"`).
  - b. Devuelva lo pedido interactuando con algún parámetro (el parámetro no puede ser de tipo arreglo).
  - c. Devuelva lo pedido sin usar parámetros ni la sentencia `"return"`.
6. Análisis de las estructuras de listas provistas por la API de Java.
- a. ¿En qué casos `ArrayList` ofrece un mejor rendimiento que `LinkedList`?
  - b. ¿Cuándo `LinkedList` puede ser más eficiente que `ArrayList`?
  - c. ¿Qué diferencia encuentra en el uso de la memoria en `ArrayList` y `LinkedList`?
  - d. ¿En qué casos sería preferible usar un `ArrayList` o un `LinkedList`?
7. Uso de las estructuras de listas provistas por la API de Java. Para resolver este punto cree el paquete **tp1.ejercicio7**
- a. Escriba una clase llamada `TestArrayList` cuyo método `main` recibe una secuencia de números, los agrega a una lista de tipo `ArrayList`, y luego de haber agregado todos los números a la lista, imprime el contenido de la misma iterando sobre cada elemento.
  - b. Si en lugar de usar un `ArrayList` en el inciso anterior hubiera usado un `LinkedList` ¿Qué diferencia encuentra respecto de la implementación? Justifique
  - c. ¿Existen otras alternativas para recorrer los elementos de la lista del punto 7a.?
  - d. Escriba un método que realice las siguientes acciones:
    - cree una lista que contenga 3 estudiantes
    - genere una nueva lista que sea una copia de la lista del inciso i
    - imprima el contenido de la lista original y el contenido de la nueva lista
    - modifique algún dato de los estudiantes
    - vuelva a imprimir el contenido de la lista original y el contenido de la nueva lista. ¿Qué conclusiones obtiene a partir de lo realizado?
    - ¿Cuántas formas de copiar una lista existen? ¿Qué diferencias existen entre ellas?
  - e. A la lista del punto 7d, agregue un nuevo estudiante. Antes de agregar, verifique que el estudiante no estaba incluido en la lista.
  - f. Escriba un método que devuelva verdadero o falso si la secuencia almacenada en la lista es o no capicúa:  

```
public boolean esCapicua(ArrayList<Integer> lista)
```

Ejemplo:



## Algoritmos y Estructuras de Datos

- El método devuelve verdadero si la secuencia ingresada es: 2 5 2
- El método devuelve falso si la secuencia ingresada es: 4 5 6 3 4

- g. Considere que se aplica la siguiente función de forma recursiva. A partir de un número  $n$  positivo se obtiene una sucesión que termina en 1:

$$f(n) = \begin{cases} \frac{n}{2}, & \text{si } n \text{ es par} \\ 3n + 1, & \text{si } n \text{ es impar} \end{cases}$$

Por ejemplo, para  $n = 6$ , se obtiene la siguiente sucesión:

$$f(6) = 6/2 = 3$$

$$f(3) = 3*3 + 1 = 10$$

$$f(10) = 10/2 = 5$$

....

Es decir, la sucesión 6, 3, 10, 5, 16, 8, 4, 2, 1. Para cualquier  $n$  con el que se arranque siempre se llegará al 1.

- Escriba un programa recursivo que, a partir de un número  $n$ , devuelva una lista con cada miembro de la sucesión.

```
public class EjercicioSucesion {  
  
    public List<Integer> calcularSucesion (int n) {  
  
        //código  
  
    }  
  
}
```

- h. Implemente un método recursivo que invierta el orden de los elementos en un ArrayList.

```
public void invertirArrayList(ArrayList<Integer> lista)
```

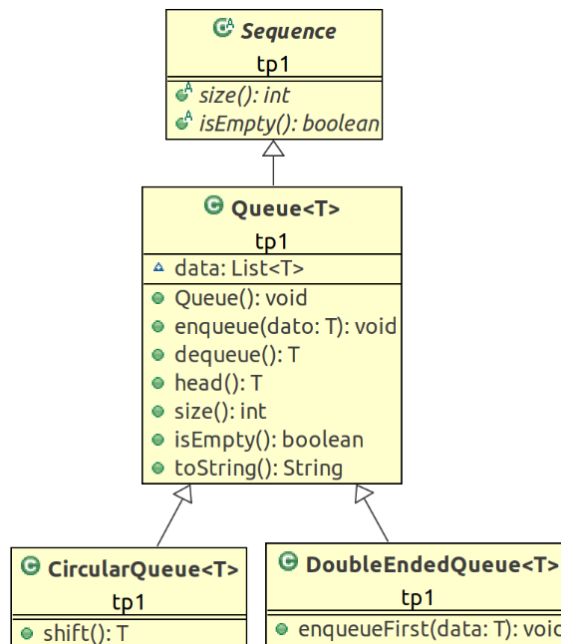
- i. Implemente un método recursivo que calcule la suma de los elementos en un LinkedList.

```
public int sumarLinkedList(LinkedList<Integer> lista)
```

- j. Implemente el método "combinarOrdenado" que reciba 2 listas de números **ordenados** y devuelva una nueva lista también ordenada conteniendo los elementos de las 2 listas.

```
public ArrayList<Integer> combinarOrdenado(ArrayList<Integer> lista1,  
ArrayList<Integer> lista2);
```

8. El objetivo de este punto es ejercitar el uso de la API de listas de Java y aplicar conceptos de la programación orientada a objetos. Sean las siguientes especificaciones de cola, cola circular y cola con 2 extremos disponibles, vistas en la explicación teórica:



a. Implemente en JAVA la clase **Queue** de acuerdo con la especificación dada en el diagrama de clases. Defina esta clase dentro del paquete **tp1.ejercicio8**.

- **Queue()** Constructor de la clase
- **enqueue(dato: T): void** Inserta el elemento al final de la cola
- **dequeue(): T** Elimina el elemento del frente de la cola y lo retorna. Si la cola está vacía se produce un error.
- **head(): T** Retorna el elemento del frente de la cola. Si la cola está vacía se produce un error.
- **isEmpty(): boolean** Retorna verdadero si la cola no tiene elementos y falso en caso contrario
- **size(): int** Retorna la cantidad de elementos de la cola.
- **toString(): String** Retorna los elementos de la cola en un String

b. Implemente en JAVA las clase **CircularQueue** de acuerdo con la especificación dada en el diagrama de clases. Defina esta clase dentro del paquete **tp1.ejercicio8**.

- **shift(): T** Permite rotar los elementos, haciéndolo circular. Retorna el elemento encolado.

c. Implemente en JAVA la clase **DoubleEndedQueue** de acuerdo con la especificación dada en el diagrama de clases. Defina esta clase dentro del paquete **tp1.ejercicio8**.

- **enqueueFirst(): void** Permite encolar al inicio.

9. Considere un *string* de caracteres *S*, el cual comprende únicamente los caracteres: (, ), [, ], {, }, . Decimos que *S* está balanceado si tiene alguna de las siguientes formas:

*S* = "" *S* es el *string* de longitud cero.

*S* = "(T)"

*S* = "[T]"

*S* = "{T}"

*S* = "TU"



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

## Algoritmos y Estructuras de Datos

Donde ambos T y U son *strings* balanceados. Por ejemplo, "{ ( ) [ ( ) ] }" está balanceado, pero "( [ ) ]" no lo está.

- Indique qué estructura de datos utilizará para resolver este problema y cómo la utilizará.
- Implemente una clase llamada **tp1.ejercicio9.TestBalanceo**, cuyo objetivo es determinar si un String dado está balanceado. El String a verificar es un parámetro de entrada (no es un dato predefinido).

Para los ejercicios 10 y 11, solo es necesario responder las preguntas. No es necesario realizar ninguna implementación.

10. Considere el siguiente problema: Se quiere modelar la cola de atención en un banco. A medida que la gente llega al banco toma un ticket para ser atendido, sin embargo, de acuerdo a la LEY 14564 de la Provincia de Buenos Aires, se establece la obligatoriedad de otorgar prioridad de atención a mujeres embarazadas, a personas con necesidades especiales o movilidad reducida y a personas mayores de setenta (70) años. De acuerdo a las estructuras de datos vistas en esta práctica, ¿qué estructura de datos sugeriría para el modelado de la cola del banco?
11. Considere el siguiente problema: Se quiere modelar el transporte público de la ciudad de La Plata, lo cual involucra las líneas de colectivos y sus respectivas paradas. Cada línea de colectivos tiene asignado un conjunto de paradas donde se detiene de manera repetida durante un mismo día. De acuerdo a las estructuras de datos vistas en esta práctica, ¿qué estructura de datos sugeriría para el modelado de las paradas de una línea de colectivos?