



# **Examen técnico**

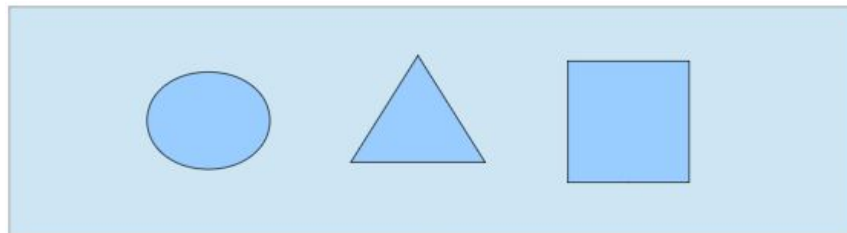
## **Ingeniería de Software**

Desarrollador JAVA

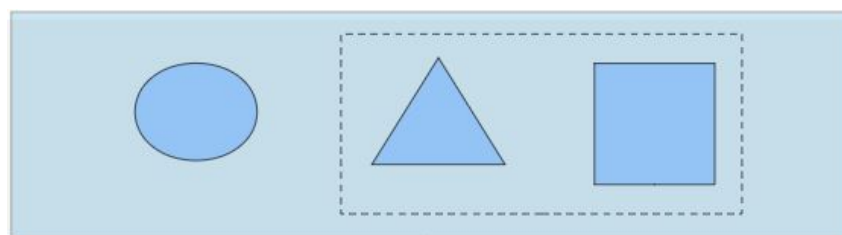
## Diseño Orientado a objetos

### Figuras de Word

En una hoja de Word se pueden agregar dibujos o figuras.



Word nos permite agrupar las figuras seleccionadas, de forma tal que las figuras agrupadas pasan a formar parte de una nueva figura que las engloba. Ahora, sobre la nueva figura, pueden aplicarse las mismas operaciones que nos eran permitidas sobre las figuras básicas, como modificar el tamaño, color, ancho de las líneas, etc. Además Word 'memoriza' varios niveles de agrupamiento que, uno por uno, pueden deshacerse con el botón de desagrupar.



### Consigna

Modele lo descrito anteriormente.

Se espera un diagrama de clases que modele lo que considere lo más importante del problema.

## Árbol Genealógico

Modele una especie de Árbol Genealógico con un diagrama de clases que tenga en cuenta relaciones de padre, hijo, pareja, tío, abuelo, etc.

### Consigna

Encontrar una solución al problema. No es necesario un diseño detallado, pero sí una idea general.

## Problemas de programación

Todos los problemas de programación tienen que contemplar los casos bordes y los errores. Usar sintaxis JAVA sin importar si compila o no.

### Iterador sobre múltiples secuencias

Crear una clase SequenceIterator que permita obtener de forma ordenada los elementos de un conjunto de secuencias de entrada iterables:

Ejemplo:

**Entrada:** { [1,3,5,7,9,...] , [2,4,6,8,...], [0,10,20,30,...], .... }

**Salida:** Un iterador sobre la secuencia 0,1,2,3,4,5,6,7,8,9,10...

Las secuencias de entrada se encuentran ordenadas y las mismas pueden ser arbitrariamente grandes (**no es posible cargarlas enteras en memoria**).

La clase debe contener estos métodos públicos, (pueden agregarse los que hagan falta):

```
public class SequenceIterator {  
    public SequenceIterator(Collection<Iterator<Comparable>> inputs) {  
        ...  
    }  
    public boolean hasNext() {  
        ...  
    }  
    public Comparable next() {  
        ...  
    }  
}
```


## Concurrencia

Se tiene la clase

```
class Item {
    private int value1;
    private int value2;
    public void setValue1(int v) {
        value1=v;
    }
    public void setValue2(int v) {
        value2=v;
    }
    public int getValue1() {
        return value1;
    }
    public int getValue2() {
        return value2;
    }
}
```

Implementar la interfaz:

```
public interface ConcurrentMemoryStore {
    /**
     * Almacena un Item asociado a una clave key
     * @throws IllegalArgumentException Si ya existe un valor
     * asociado a la clave
     */
    void store(String key, Item item) throws IllegalArgumentException;
    /**
     * Actualiza los valores del Item asociado a key.
     * La instancia de Item que queda almacenada no debe cambiar.
     */
    void update(String key, int value1, int value2);
    /**
     * Retorna un iterador sobre los Items contenidos
     */
    Iterator<Item> valueIterator();
    /**
```



```
* Borra el Item con clave key
*/
void remove(String key);
}
```

La implementación debe ser thread-safe y lo más eficiente posible. El iterador retornado por `valueIterator()` debe ser recorrible sin riesgo de una `ConcurrentModificationException`.