

# Laboratorio de Software - 2025

## Práctica nº 3

Temas

- Clases anidadas y clases internas.
- Clases anónimas
- Módulos
- Tipos enumerativos

**(1)** Complete el código de la clase `Stack` en el paquete `practica3`, de manera que implemente una pila de `String`:

```
public class Stack {  
    private java.util.ArrayList items;  
    public Stack() { . . . }  
    public void push(Object item) { . . . }  
    public Object pop() { . . . }  
    public boolean isEmpty() { . . . }  
}
```

a) Implemente un método `main()` para probar la pila. Agregue `Strings` a la pila y recórrala para imprimir sus valores. ¿Cuántas veces puede recorrerla?

Para imprimir los valores tengo que obtener el objeto de la pila, pudiendo recorrerla una única vez.

b) Agregue una clase anidada llamada `StackIterator` que provea un objeto de tipo `Iterator` para recorrer la pila.

c) Agregue en la clase `Stack` un método para que retorne una instancia de `StackIterator`. ¿Cuántas veces puede recorrer la pila ahora?

Ahora la pila puede recorrerse infinitas veces.

d) ¿Es posible crear objetos `StackIterator` desde una clase diferente a la clase `Stack` con el operador `new`?, ¿cómo lo hace?

Para poder crear este tipo de objetos, la clase anidada debe ser `public`. Sin embargo, no puede crearse directamente con un `new`, si no a través de una instancia de `Stack`.

e) ¿Cómo haría para evitar crear instancias de una clase anidada desde una clase que no sea la que la definió?

Declarando la clase como `privada`.

**(2)** Analice el código que figura debajo.

```
class InnerStatic {
    static double PI = 3.1416;
    static class Circulo {
        static double radio = 2;
        static double getArea() {
            double a= PI* Math.pow(radio,2);
            System.out.println("El area es: "+a);
            return a;
        }
        static double getLongitudCircunsferencia(){
            double l= 2*PI*radio;
            System.out.println("La longitud es: "+l);
            return l;
        }
    }
    ...
}
```

- a) Modifique el código de la **clase interna estática** para que el valor inicial del radio sea ingresado por el usuario en el momento de la ejecución.
- b) Defina una clase llamada InnerTest en el paquete practica3 con un método main() que imprime en la pantalla el área y la longitud de la circunferencia. Ejecútela varias veces ingresando distintos radios.
- c) Reemplazar `PI* Math.pow(radio,2)` por `PI* pow(radio,2)`, siendo `pow()` el método de la clase `java.lang.Math`.

Para que funcione debe importarse `java.lang.Math.pow` como static.

**(3)** Implemente una clase llamada `StringConverterSet` como subclase de `AbstractSet`, la cual permita realizar todas las operaciones contempladas para los Set, con la salvedad que el método `iterator()` retorne un `Iterator` que al recorrerlo devuelva cada uno de los elementos como `Strings`. Para su solución, defina un Adapter llamado `IteratorStringAdapter` como una clase anidada de `StringConverterSet` para cumplir lo solicitado.

**(4)** Indicar si son verdaderas o falsas las siguientes afirmaciones sobre las **clases anónimas** y en cada caso justifique su respuesta:

- Se pueden instanciar más allá del punto en donde fueron declaradas.
  - FALSO → Las clases anónimas no tienen nombre y se declaran e instancian en el mismo punto. No existe una referencia a un tipo declarado con nombre.
- Unos de los usos más comunes de este tipo de clases es la creación de objetos función y procesos *on the fly*.
  - VERDADERO → El objetivo de las clases anónimas es la creación de implementaciones rápidas y puntuales (*on the fly*). Se usan para callbacks y listeners.

- Se puede utilizar el `instanceof` siempre y cuando la interfaz de la que deriva la clase anónima sea de tipo `marker`.
  - FALSO → La clase anónima no tiene nombre, no se le puede indicar al operador contra qué comparar el tipo de la clase.  
El operador se puede usar con cualquier tipo accesible que la clase anónima implemente o extienda.
- No se puede implementar múltiples interfaces o extender clases e implementar interfaces al mismo tiempo.
  - VERDADERO → Por sintaxis de Java.

**(5)** Modifique el código de la clase `Stack`, para que ahora la clase anidada `StackIterator`, se convierta en una clase anónima.

a) ¿En que situación es conveniente definir a una clase cómo anónima?

Conviene definir una clase anónima cuando necesitamos crear una implementación concreta de una interfaz o de una clase abstracta sin tener que declarar una nueva clase con nombre.

Casos:

- La implementación es muy corta (pocas líneas de código).
- Se va a usar solo una vez en ese punto del programa.
- No tiene sentido volver a reutilizarla en otras partes.

b) Si tuviera que inicializar valores de la clase anónima (cuando se crea una instancia de la misma), ¿cómo lo haría?

Se logra mediante un bloque de inicialización. El bloque de inicialización funciona como un constructor para la clase anónima. Se ejecuta cada vez que se crea una instancia.

El uso del bloque de inicialización para definir constructores es limitado dado que NO es posible definir constructores sobrecargados.

**(6)** Defina una clase llamada `Estudiante` que contenga las siguientes variables de instancia: `apellido`, `nombre`, `edad`, `legajo` y `materiasAprobadas`. Se necesita poder ordenar un arreglo con estos objetos por los siguientes criterios:

- Por cantidad de materias aprobadas en forma ascendente.
- Por edad en forma descendente.
- Por legajo en forma ascendente.
- Por nombre y apellido en forma descendente.

Implemente un método `main()` que imprima los resultados de las distintas ordenaciones utilizando clases anónimas y el método `Arrays.sort()`.

## **(7) Uso de Módulos**

Retomando la clase `Logger` que implementaron en el ejercicio 4 del TP 1, realice ahora un proyecto modular:

a) Defina un módulo llamado `loggingutils` que contenga la clase `Logger`.

b) En cada uno de sus métodos (`logInfo(String mensaje)`, `logWarning(String mensaje)`, `logError(String mensaje)`), en lugar de imprimir directamente en consola, utilice la clase `java.util.logging.Logger` para registrar los mensajes.

c) ¿Para qué sirve el archivo `module-info.java`? En el módulo `loggingutils`, ¿qué declaración debe incluirse en `module-info.java` para exponer su paquete al resto de los módulos? Para usar `java.util.logging.Logger` desde `loggingutils`, ¿es necesario declarar alguna dependencia en `module-info.java`? ¿Por qué?

El archivo `module-info.java` es el descriptor del módulo. Define el nombre del módulo, indica qué paquetes exporta para que otros módulos puedan usarlos, indica qué dependencias requiere de otros módulos, puede controlar encapsulación, etc. Organiza el código en módulos más seguros y con dependencias explícitas.

Para exponer el paquete al resto de los módulos debe agregarse el `export`. Ejemplo:

```
module loggingutils {  
    exports loggingutils;  
}
```

Para usar `java.util.logging.Logger` desde `loggingutils` no es necesario declarar ninguna dependencia porque forma parte del módulo base de Java.

d) Luego, cree un segundo módulo llamado `test` que contenga una clase con el método `main()`. Desde el `main()`, utilice el `Logger` del módulo `loggingutils` para mostrar los tres tipos de mensajes en la salida.

El proyecto debería quedar estructurado de la siguiente manera:

```
project/  
├── loggingutils/  
│   ├── src/  
│   │   ├── module-info.java  
│   │   └── loggingutils/  
│   │       └── Logger.java  
└── test/  
    ├── src/  
    │   ├── module-info.java  
    │   └── test/  
    │       └── Main.java
```

La documentación de la clase `java.util.logging.Logger` está disponible en:  
<https://docs.oracle.com/en/java/javase/24/docs/api/java.logging/java/util/logging/Logger.html>

## **(8) Declaración e implementación de Tipos Enumerativos**

- a) Implemente un tipo enumerativo llamado Notas que defina los valores de las notas musicales y con su correspondiente cifrado americano (almacenado en un String).
- b) Implemente un tipo enumerativo llamado FrecuenciasDeLA que represente las siguientes frecuencias estándares de afinación:
- 440 Hz: Organización Internacional de Estandarización ISO 16.
  - 444 Hz: Afinación de cámara.
  - 446 Hz: Renacimiento.
  - 480 Hz: Órganos alemanes que tocaba Bach.
- c) Sobrecargue los métodos hacerSonar() y afinar() de la interface InstrumentoMusical del ejercicio 1b) de la práctica 2 de manera que el nuevo hacerSonar(Notas n, int duracion) reciba como parámetro una nota musical y una duración, y el nuevo método afinar(FrecuenciaDeLA f) reciba como parámetro una frecuencia de LA.
- d) Defina una clase llamada Piano que implemente la interface InstrumentoMusical y una clase TestPiano que permita probar los métodos implementados.
- e) Implemente el patrón de diseño Singleton mediante un tipo Enumerativo el cual represente a Fito Páez. Fito cuenta con un instrumento musical (piano) y en algún momento se le puede pedir que toque una canción (especificando un arreglo de notas musicales con sus tiempos).