

Laboratorio de Software

Práctica nº 3

Temas

- Clases anidadas y clases internas.
- Clases anónimas
- 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?

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?

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

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

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 $PI * \text{Math.pow}(\text{radio}, 2)$ por $PI * \text{pow}(\text{radio}, 2)$, siendo **pow()** el método de la clase `java.lang.Math`.

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 recorreo 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.
- ☐ Unos de los usos más comunes de este tipo de clases es la creación de objetos función y procesos *on the fly*.
- ☐ Se puede utilizar el **instanceof** siempre y cuando la interfaz de la que deriva la clase anónima sea de tipo **marker**.
- ☐ No se puede implementar múltiples interfaces o extender clases e implementar interfaces al mismo tiempo.

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?

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

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 resultado de las distintas ordenaciones utilizando clases anónimas y el método **Arrays.sort()**.

7. Declaración e implementación de **Tipos Enumerativos**

a) Implemente un tipo enumerativo llamado **Notas** que define 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).

