Práctica 5

Curso (semestre de primavera) 2019 – 2020

Descripción de la aplicación.

Se va a realizar un programa en JAVA que permita mostrar gráficamente la señal proporcionada por varios generadores de señales discretas periódicas. La señal periódica generada es una secuencia de valores de tipo real, simétrica respecto al eje *x*, que se repite cada intervalo de *T* muestras, es decir cada *T* valores la señal vuelve a ser la misma y por tanto presentará una secuencia igual a la ya emitida (*T* se denomina periodo de la señal y debe ser un número entero – Por ejemplo: la señal se repite cada 5 muestras o la señal se repite cada 128 muestras).

En concreto, habrá tres generadores de señal: sinusoidal, cuadrada, y sinusoidal con saturación. Para dibujar la señal de salida de los generadores se proporciona **YA CODIFICADA** una clase denominada SondaGrafica. Algunos ejemplos obtenidos ejecutando el programa para tres tipos de señales distintos son los que se muestran a continuación:

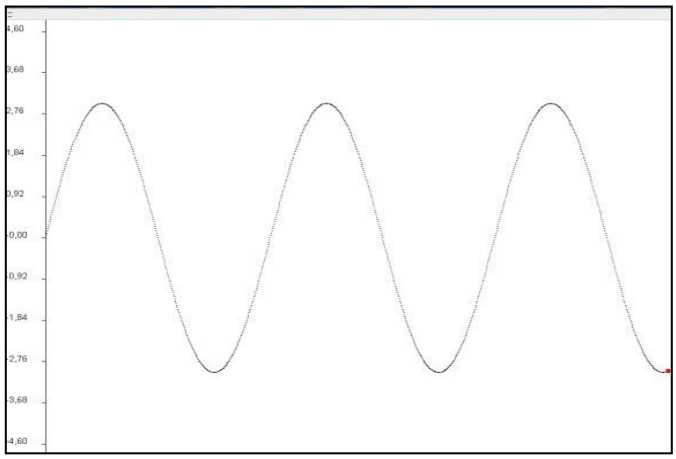

Figura 1. Ejemplo de salida para generador de señal sinusoidal

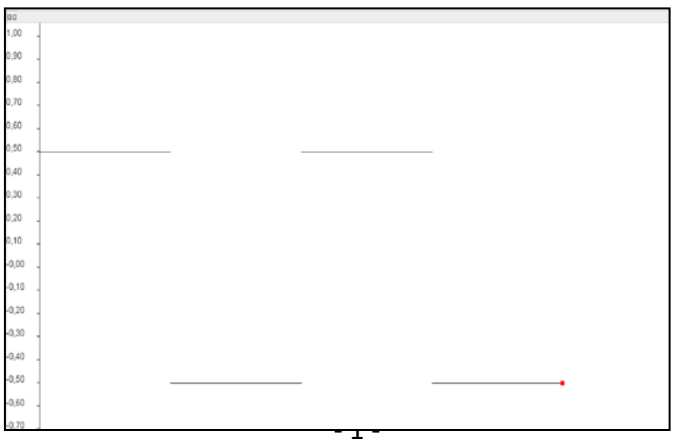

Figura 2. Ejemplo de salida para generador de señal cuadrada

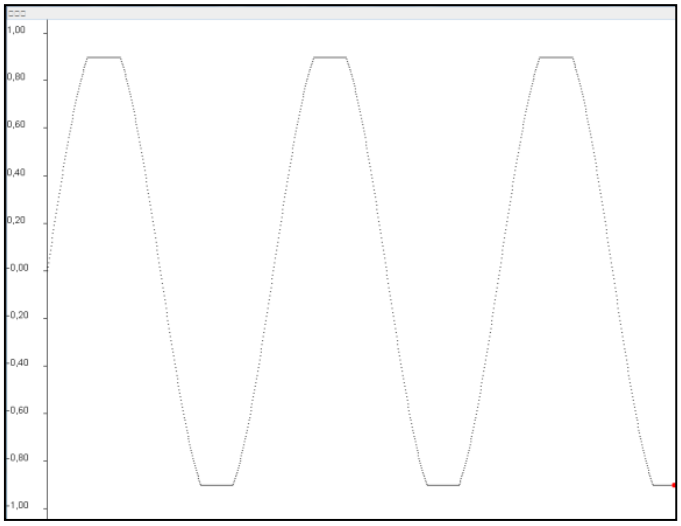

Figura 3. Ejemplo de salida para generador de señal sinusoidal con saturación

Desarrollo de la práctica

Para desarrollar el programa se ha decidido definir la siguiente estructura de paquetes y clases:

- Paquete **generadores**
  - Generador, GeneradorSinusoidal, GeneradorCuadrada, GeneradorSinusiodalConSaturacion
- Paquete **usuario**
  - P5Generador
- Paquete **grafico**
  - SondaGrafica

Como referencia, a continuación, se muestra un diagrama de clases UML de la práctica:

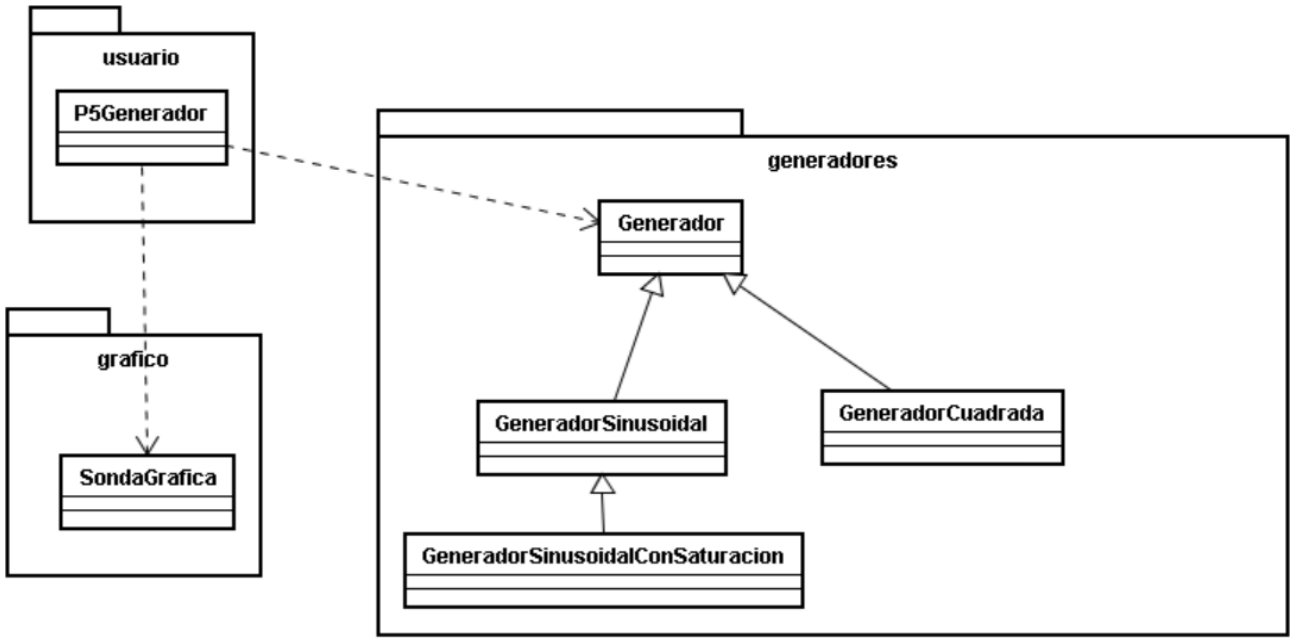

Figura 4. Diagrama UML con las clases y paquetes

A continuación, se describen las citadas clases que deberán ser codificadas por el alumno conforme a la documentación *javadoc* que se proporciona en Moodle junto a este enunciado.

- **Clase `Generador`**. Es una clase abstracta que modela la parte común a cualquier generador de señal periódica. En su constructor se permite asignar un nombre a cada generador, la frecuencia de la señal generada (entendida exclusivamente como la inversa del periodo → 1/T; Recuerde que el periodo debe ser un número entero), y su amplitud máxima. Su método `double getSalida()` es un método abstracto que deberá ser implementado por las subclases. Cada generador específico debe especializar esta clase y reescribir este método para que proporcione, uno a uno, los valores (o muestras) en secuencia que constituyan la señal generada. Todos los generadores tienen como características comunes, por tanto, un nombre que permite identificarlo, la frecuencia de la señal que se genera (como 1/T), y la amplitud máxima de la señal generada.

- **Clase `GeneradorSinusoidal`** es una especialización de la clase Generador. Aporta la parte específica de un generador que origina una señal sinusoidal de tiempo discreto. Su método `double getSalida()` produce la secuencia de valores de una señal sinusoidal de amplitud *A*, según la función de generación:

$$Salida[tn] = A*\text{seno}\ ((2*\text{PI}*f*tn) + fase0)$$

Siendo:

f: frecuencia, calculada exclusivamente como la inversa del periodo de la señal (1/*T*);

tn: el tiempo discreto de la muestra generada de la señal, que en cada llamada al método *getSalida()* se incrementará en 1 internamente de forma automática. O sea, la primera muestra se generará en la primera llamada y se corresponderá con *tn*=0, la segunda muestra se generará en la segunda llamada y se corresponderá con *tn*=1 y así sucesivamente;

fase0: la fase inicial de la señal a generar (con un rango de –PI/2 a PI/2).

A: el valor máximo de la amplitud de la señal.

Nota: La clase Math, incluida en el paquete java.lang, proporciona métodos para manejar funciones sinusoidales, así como para manejar la constante matemática PI.

- **Clase `GeneradorCuadrada`** es una especialización de la clase `Generador`. Aporta la parte específica de un generador que produce una señal cuadrada de tiempo discreto de amplitud (-*A*, *A*). En concreto, su método `double getSalida()` produce una señal cuadrada, según la función de generación (para un periodo):

$$Salida[tn] = \begin{cases} A & \text{para } 0 <= tn < T/2 \\ -A & \text{para } T/2 <= tn < T \end{cases}$$

Siendo:

T: el periodo de la señal generada en número de muestras, que se calculará como la inversa de la frecuencia y es un número entero;

tn: el tiempo discreto de la muestra generada de la señal, que en cada llamada al método *getSalida()* se incrementará en 1 internamente. O sea, la primera muestra se generará en la primera llamada y se corresponderá con *tn*=0, la segunda muestra se generará en la segunda llamada y se corresponderá con *tn*=1 y así sucesivamente;

A: el valor máximo de la señal.

- Clase **`GeneradorSinusoidalConSaturacion`**, que modela un generador de señal sinusoidal, pero con un valor máximo de saturación (umbral) que no será sobrepasado, según la función de generación:

$$Salida[tn] = \begin{cases} \text{umbral, si la salida del GeneradorSinusoidal} > \text{umbral} \\ \text{salida del GeneradorSinusoidal (GS), si } |\text{salida de GS}| \leq \text{umbral} \\ -\text{umbral, si la salida de GeneradorSinusoidal} < -\text{umbral} \end{cases}$$

Siendo:

umbral: un valor de saturación, tal que 0 < umbral ≤ *A*.

La clase `GeneradorSinusoidalConSaturacion` debe especializar la clase `GeneradorSinusoidal`. El constructor de esta nueva clase debe recibir además como uno de sus parámetros el umbral de saturación como un valor de tipo double.

De esta clase **no** se proporciona información en Moodle, el alumno deberá codificarla de acuerdo al resto de clases y conservando los requisitos de la aplicación.

- Clase principal **`P5Generador`**. Complete la clase `P5Generador` que contiene el método principal `main`. El programa hace lo siguiente:

    1. Instancia uno de los generadores de señal. El tipo de generador de señal a instanciar lo elige el usuario al que se preguntará por consola, leyendo los caracteres 'C', 'S' ó 'U' (C: cuadrada, S: sinusoidal, U: sinusoidal con saturación). Los siguientes valores se establecen fijos: el periodo de la señal discreta será de 500 muestras, por lo que se implanta una frecuencia de 1/500; la fase inicial de las señales sinusoidales será 0 rad; la amplitud máxima es 1; y para el umbral, en el caso del generador con saturación, se utilizará por defecto el valor 0,8.

    2. Obtiene, uno a uno, los primeros 1000 valores de la señal generada por el generador elegido, lo que permitirá mostrar dos periodos completos de la misma, proporcionárselos a un objeto de la clase `SondaGrafica` para que realice la representación gráfica de la señal en ese espacio temporal en la ventana correspondiente. En el título de la ventana se añadirá el nombre del generador elegido, su amplitud y su frecuencia.

En Moodle se ha dejado una plantilla de este programa donde el alumno sólo debe codificar el método:

```
private static void mostrarSonda(Generador generador, int muestras)
```

Documentación adicional necesaria para la realización de la práctica

Para poder realizar esta práctica y obtener una representación gráfica de la señal de los generadores es necesario que el alumno se descargue y use los siguientes recursos de Moodle:

- La clase `SondaGrafica` que nos permitirá representar una señal. Esta clase se proporciona totalmente codificada. Cada sonda visualiza la señal que se le proporciona (valor a valor de señal, mediante el método `addMuestra()`) en una ventana gráfica independiente. Puede utilizar dos o más sondas si desea ver gráficamente dos o más señales. Además, la implementación de esta sonda introduce un pequeño retardo tras dibujar cada punto de la señal, lo cual permite ralentizar su representación para observar más lentamente su proceso de trazado. Se utilizará el constructor que permite definir el título de la ventana:

  ```
  public SondaGrafica(String titulo)
  ```

- Los ficheros *jar* se utilizan en Java para empaquetar y distribuir ficheros binarios java (.class). En esta práctica la clase `SondaGrafica` utiliza la biblioteca *SimpleGUI.jar* para representar las señales de forma gráfica. Por ello, para que la clase `SondaGrafica` compile sin errores y funcione, debe incorporar esta biblioteca, contenida en un fichero denominado `SimpleGUI.jar`, a su proyecto de Eclipse como fichero externo. Para ello:
  - Hacer click con el botón derecho del ratón en el proyecto java
  - Seleccionar “Build Path”
  - Seleccionar “Configure Build Path”
  - Seleccionar la pestaña Libraries
  - Hacer click en el botón “Add External JARs”
  - Ir al directorio donde se encuentra el .jar, seleccionarlo y hacer click en el botón abrir
  - Hacer click en el botón “Apply and Close”
  - El paquete debe aparecer en “Referenced Libraries”

NOTA: Para codificar la práctica no necesita conocer nada acerca de la biblioteca *SimpleGUI*. Sólo necesita conocer la INTERFAZ reducida de uso de la clase `SondaGrafica` (para lo cual puede consultar el *javadoc* proporcionado en Moodle).

Información adicional a tener en cuenta

La práctica, aunque compile y se ejecute sin errores y cumpla con las funcionalidades pedidas, **debe respetar de forma completa las especificaciones del enunciado**. El alumno no debe cambiar las definiciones de las clases ni de los métodos públicos, ni añadir nuevos métodos/atributos públicos, salvo que previamente se lo justifique al profesor y éste lo autorice.

- Objetivos que se presuponen alcanzados en prácticas anteriores:

    - Los atributos de las clases tienen la visibilidad adecuada (que nunca será pública).
    - Se evita la definición de un atributo/método como *static* cuando no está adecuadamente justificado.
    - Se definen, instancian o inicializan solo los objetos y variables necesarios en cada momento.
    - Se definen como atributos únicamente aquellos que realmente deben ser un atributo. Por ejemplo, se penaliza definir como atributo algo que realmente debería ser una variable local.
    - Se utilizan tipos de datos adecuados a la finalidad del dato. Por ejemplo, se penaliza el uso de enteros cuando realmente debería usarse un booleano, o definir una tabla de una dimensión arbitraria e injustificada.
    - Se utilizan estructuras de control claras y eficientes para lo que necesita hacer el programa.
    - Se utilizan nombres de identificadores (atributo, variable…) significativos.
    - Se utilizan las normas de estilo de Java (mayúsculas, etc.) para los identificadores de clases, atributos, métodos, etc.
    - El código fuente está adecuadamente comentado con *Javadoc* (descripción, parámetros, valor de retorno si lo hay…).

    - La interacción con el usuario (leer de teclado o escribir en pantalla), si es necesaria, se realiza únicamente en la clase principal.

- Objetivos específicos de esta práctica:

    - Aplicando el mecanismo de herencia, la subclase redefine únicamente aquello que es necesario y aprovecha todo lo que sea posible de la superclase (por tanto, no repite código ya presente en la superclase).
    - Se utiliza polimorfismo para trabajar con objetos que tienen una superclase común (jerarquía de clases), o que implementan una misma interfaz.