Laboratorio de Software
Enunciado del Segundo Entregable del Trabajo Final
Cursada 2015
Entrega 25/11/2015

Proyecto HERMES versión 2

Objetivo: construir el módulo de comunicación del **Monitor** de HERMES basado en ***threads*** y en el protocolo HTTP junto con un **testeador** que utiliza ***anotaciones.***

La entrega consiste en la implementación del **Monitor** completo (GUI de la primera entrega del proyecto integrado con el módulo de comunicación) y el **Testeador** para probar el **Monitor**.

1) Módulo de comunicación del **Monitor**

El **Monitor** recibirá a través del método **POST** del protocolo HTTP múltiples notificaciones provenientes de los clientes Android, codificadas en formato JSON.
El mensaje JSON puede incluir una o varias notificaciones. Tenga en cuenta que en el caso de perderse la conexión de red, el cliente Android puede "guardar" las notificaciones y luego cuando se establece la conexión, enviar el mensaje JSON con todas las notificaciones guardadas.
La implementación del **Monitor** está basada en **threads**. El main-thread es la GUI del **Monitor**. A su vez este **thread** creará otro separado encargado de instanciar e iniciar un servidor HTTP (clase com.sun.net.httpserver.HttpServer) que escuchará en el puerto especificado en el archivo de configuración "conf.properties" los mensajes JSON provenientes de los clientes Android.
Para configurar el servidor HTTP se usará el método newCachedThreadPool() de la clase java.util.concurrent.Executors, responsable de crear un pool de **threads.** Éstos ejecutarán concurrentemente objetos HttpHandler (interface com.sun.net.httpserver.HttpHandler) responsables de procesar los mensajes JSON y persistir las notificaciones en la base de datos.

Tenga en cuenta que la GUI del **Monitor** debe actualizarse "automáticamente" a medida que recibe notificaciones entrantes, siempre y cuando no se haya aplicado un filtro. Es decir la GUI tendrá 2 vistas de notificaciones entrantes:

- "Default": se muestran todas las notificaciones ordenadas de la más reciente a la más antigua y a medida que llegan notificaciones se actualiza esta vista.
- "Filtrado": se muestran las notificaciones que cumplan con el criterio del filtro y si llegan notificaciones nuevas, se informará este estado en la GUI con una leyenda.

Para volver a la vista "Default" se debe contar con un botón en la GUI ("Mostrar Todo").

2) **Testeador del Monitor**

Este **Testeador** tiene por objetivo generar notificaciones de prueba y enviarlas en formato JSON por la red. La funcionalidad del Testeador será reemplazada posteriormente en la versión final de HERMES por el comunicador Android.

Para la implementación del **Testeador** se usarán dos módulos: un generador de objetos de prueba basado en anotaciones JAVA y otro módulo que se encargará de serializar esos objetos de prueba en formato JSON y enviarlos por la red.

El módulo generador de objetos consta de un conjunto de anotaciones y un procesador de las mismas, llamado **MockGenerator.**

Las anotaciones que se deben implementar son las siguientes:

- Anotación de clase: public @interface **Mock**{}
- Anotaciones de atributos:

  public @interface **MockStringAttribute** {String[] value();}

  public @interface **MockTodayAttribute** {}

  Ejemplos de uso:

  @MockStringAttribute ({“descriptivo1”, “descriptivo2”, “descriptivo3”})

  @MockStringAttribute ({“Establo-Terapia”, “Pista”, “Hogar”}),

  @MockStringAttribute ({“Predeterminada“, “Emociones”, “Alimentos”, “Actividades y Paseos”})

  @MockStringAttribute({“Juan”, “Pedro”, “Juana”, “Manuela”})

  @MockTodayAttribute

El procesador de anotaciones **MockGenerator** será el encargado de generar instancias de prueba de las clases anotadas con **@Mock** y cuyos datos de prueba estarán determinados por las anotaciones de atributo **@MockStringAttribute** y **@MockTodayAttribute.** El **MockGenerator** generará una lista de de objetos de prueba invocando al método genérico createMockInstances(Class<T> elClass, int cant). Este método crea una lista con *cant* objetos de prueba con valores de atributos tomados al azar. Considere un *cant* no menor a 40.

A continuación se describe la clase **MockGenerator:**

```
public class MockGenerator {
    public static <T> List<T> createMockInstances(Class<T> elClass, int cant){ //TODO }
}
```

El **Testeador** utilizará el módulo generador de objetos de prueba anotando con **@Mock** la clase **Notificacion** y con **@MockStringAttribute** y **@MockTodayAttribute** sus atributos**.** Luego utilizará **MockGenerator** para obtener una lista de notificaciones de prueba. El **Testeador** a partir de esta lista construye un mensaje JSON que contiene un arreglo que representa todas las notificaciones. Este mensaje es enviado por la red al puerto que está

“escuchando” el servidor HTTP del Monitor

Entrega

1) un JAR ejecutable para el **Monitor**

2) un JAR ejecutable para el **Testeador** (sustituto del comunicador)

3) Los proyectos Eclipse de 1) y 2)

Ambos ejecutables deben comunicarse. Con el propósito de prueba el **Testeador** debe ejecutarse más de una vez para simular la interacción de varios comunicadores.