Introducción

La meta de Taller de Programación 1 es introducirnos en la prueba o “testing” de software. Para testear es preciso contar con un software a probar y contra qué probarlo (los requerimientos según los cuales se construyó dicho software). El documento de requerimientos establece lo que pide quien lo solicita.

Este práctico propone varios conjuntos de requerimientos, cada uno describe un software específico a ser desarrollado por un grupo de alumnos (hasta cinco integrantes), el mismo será la base del trabajo en los sucesivos prácticos.

Los requerimientos de este práctico no están completos ni cumplen todos los requisitos de una buena especificación, puesto que es tarea de los alumnos identificar las ambigüedades y requerimientos faltantes, con el fin de desarrollar el código. Antes de codificar, debe completarse el documento de requerimientos, que podrá ser corregido durante la primera fase de programación.

Práctico 1: Programa Análisis

Objetivo:

Construir una aplicación capaz de administrar el almacenamiento de un conjunto de objetos de una clase determinada (Análisis) en una estructura persistente, permitiendo agregar, eliminar, y consultar a través de líneas de comando.

Estructura de la aplicación:

La aplicación estará compuesta por dos módulos:
Almacén (modelo): Módulo que contiene la colección de datos (Análisis), que recibe comandos en texto plano y mantiene una estructura persistente, además debe tener un nombre único con el que será identificado. Los comandos son: crear, cargar, cerrar, insertar, eliminar y consultar.

Frontend: Módulo gráfico con comandos de control, que permite introducir y ejecutar una secuencia de instrucciones.

Requerimientos:
1. Comunes a los dos módulos

1.1 El archivo de persistencia se guarda en un subdirectorio llamado “Datos”, ubicado dentro del directorio donde se encuentran los ejecutables de la aplicación. La persistencia se realizará a través de XML.

1.2 El objeto a almacenar es de la clase Análisis, la cual debe contener los siguientes atributos:

- **id** : int (clave primaria).
- **fecha** : Gregorian Calendar
- **apellido** : String
- **nombre** : String

◦ **domicilio** : String
◦ **medico** : String
◦ **estudios**: tipocoleccion<String,Double>: siendo el primer elemento el **nombre del estudio** y el siguiente el resultado. (Ej: (Glucosa,2) (“Glob_Rojos”,3752.67331)

1.3 La clase debe contener un método double valorEstudio(Nombre_Estudio String), el cual debe devolver el valor del atributo que corresponde al nombre, siempre que se encuentre en la colección, caso contrario debe devolver una excepción que deberá verse reflejado en pantalla como un mensaje que aclare cuál nombre no fue encontrado. Ej: int valor_glucosa = analisis.valorEstudio(“glucosa”)

1.4. Los comandos se van a expresar mediante esta sintaxis: (Ver glosario más abajo de ser necesario)

- **crear** <nombre_Almacen> : Inicializa por primera vez el almacén en memoria.

- **cargar** <nombre_Almacen> : Carga el almacén a memoria desde el disco.

- **guardar**: Guarda a disco el almacén abierto.
  - Si no hay un almacén abierto, lanza una excepción informando lo ocurrido.

- **importar** <nom_archivo>: dicho archivo estará en la carpeta **import** y deberá ser el XML de un análisis. (Utilizar parser de XML para la carga). Este análisis importado se cargará en el almacén que se tenga en memoria. La id que esté en el XML debe ser ignorada y se le debe asignar una id que corresponda según el orden actual.
  - En caso de no haber un almacén cargado en memoria, deberá informar a través de una excepción lo ocurrido.
  - En caso de que el parseo del archivo XML falle, también deberá lanzar una excepción notificando en pantalla que el archivo es erróneo.

- **eliminar** <id_analisis> : elimina un análisis. Si no existe, se debe lanzar una excepción e informar por pantalla lo ocurrido. Una id que fue eliminada no vuelve a usarse nunca más.
  - En caso de no haber un almacén cargado en memoria, deberá informar a través de una excepción lo ocurrido.
  - En caso de que el id_analisis no corresponda a un id válido, deberá informar a través de una excepción lo ocurrido.

- **consulta** <parámetros>: ejecuta la consulta indicada, si no se puede, lanza una excepción, en caso de éxito muestra los resultado en la vista (y podría también ser archivo) si al menos un análisis cumple con la condición establecida o avisa en la vista que no hay ningún análisis que la cumpla en caso de que sea así. El resultado de la consulta es un listado de todos los análisis que cumplan con la condición dada. En el glosario se detallan los parámetros a consultar

  Glosario:

  ➔ crear, cargar, guardar, eliminar, importar y consulta: son los comandos posibles.

- ➔ <nombre_almacen> es el nombre que tendrá el archivo de persistencia. Ej: “Arch33.xml”

- ➔ <id_analisis>: clave primaria.

- ➔ <nom_archivo>: es el nombre que tendrá el archivo de donde se importará. Ej: “Arch315.xml”

- ➔ <parametro(s)>: String que contiene 3 o 5 palabras separadas por un espacio:
  <nombre_estudio> <operador> <valor> [toFile <nombre archivo destino>].

- ➔ <operador>: “==” (igual) , “!=” (distinto). “>” (mayor) ,”<” (menor), “>=” (mayor o igual) y “<=” (menor o igual)

- ➔ toFile: palabra reservada que indica que el resultado de la consulta debe también volcarse en un archivo, indicado por <nombre archivo destino>

  Ejemplos:
  consulta glucosa >= 50 debe devolver un listado de todos los análisis cuyo resultado de glucosa sea mayor o igual a 50
  consulta glob_rojos == 0 toFile genteSinGlobRojos.txt debe mostrar una lista de análisis cuyo valor glob_rojos sea 0 en la vista, y también listarlos en el archivo genteSinGlobRojos.txt

2. Módulo Almacén

2.1. El módulo debe recibir como parámetros la cadena con el comando, interpretarla y ejecutarla.

2.2. Debe evaluar la sintaxis y emitir los mensajes de error necesarios, si los hubiera:

➔ Error000: Comando mal formado: No cumple con la sintaxis descrita (falta estructura, operaciones dispositivos, operadores ó están en orden equivocado).
Ejemplos:
crear
guardar **yaa**
consulta colesterol < 30 **toFile**

➔ Error001: Comando inexistente: En el lugar donde se espera la palabra reservada que indica la operación no figura una palabra válida.
Ejemplos:
evolucionar arch.txt
imprimir todo

- ➔ Error002: Consulta mal construida (atributos desconocidos, operadores desconocidos etc.)
  Ejemplos:
  consulta **estatura** < 160 toFile salida.txt

consulta creatinina **es_menor** 30
consulta leucocitos **?** 30
consulta creatinina **><<<<<<<** 30 toFile salida.txt

2.3. Debe evaluar la semántica (sintácticamente correcto pero aún irrealizable).

➔ Error003: Archivo inexistente.
Ejemplos:
importar **no_existo.xml**

➔ Error004: Operación no realizable.
Ejemplos:
guardar (Y no hay almacén abierto)
crear **nombre_de_almacen_que_ya_existe**

➔ Error005: Análisis inexistente.
Ejemplo:
eliminar **-5**

2.4. Los mensajes de error deben ser visualizados en el Frontend

2.5. Una vez validado el comando sintácticamente (comando bien formado) y semánticamente (existen todos sus componentes), este debe ser ejecutado.

2.6. Al término debe visualizarse si la ejecución terminó normalmente ó hubo un fallo.

3. Frontend:

Consiste en:

3.1. Un panel de edición de texto donde se redacta una secuencia de comandos.
3.2. Un panel de mensajes donde se despliegan los mensajes de error
3.3. Un panel que muestra los resultados en pantallas.
3.4. Un botón que dispara la ejecución de la secuencia de instrucciones. Esta ejecución continúa hasta que termina la secuencia de instrucciones o sucede un error.
3.5. El programa continúa mientras la pantalla no se cierra, pudiendo modificar la secuencia y pedir reiteradas ejecuciones.