Departamento de Ingeniería Telemática
Escuela Superior de Ingenieros
Universidad de Sevilla

FUNDAMENTOS DE LA PROGRAMACIÓN II

Practica Final de Programación Orientada a Objetos

Índice

1. Introducción

1.1 Objetivos

Este documento contiene la definición de la práctica final de programación orientada a objetos que los alumnos de la asignatura de Fundamentos de la Programación II, que se presenten a la convocatoria de Junio del curso 2011–2012, deberán desarrollar y presentar en la fecha correspondiente según las normas de la asignatura.

El objetivo del trabajo es el aprendizaje del manejo de los conceptos básicos de la programación orientada a objetos así como del entorno de trabajo utilizado. Para ello se propone la implementación de un programa, en el lenguaje de programación Java, que simule el funcionamiento de un sistema cuyo comportamiento se describe en este documento.

La información que procesará el programa, así como cualquier otra información auxiliar necesaria, se obtendrá mediante la lectura de ficheros o de la entrada estándar. Los resultados obtenidos después de ejecutar el programa se escribirán en la salida estándar (**`System.out`**), y en la salida de errores (**`System.err`**).

1.2. Recomendaciones iniciales

Esta práctica final utiliza los conceptos desarrollados durante las sesiones prácticas. Los ejemplos han sido orientados para facilitar la implementación de la práctica final de programación orientada a objetos.

Si algunos de los aspectos relativos al trabajo no le quedan claros, acuda al foro de la asignatura disponible en la plataforma de ***enseñanza virtual*** de la ***Universidad de Sevilla***, donde se centralizarán todas las preguntas y respuestas relacionadas con el trabajo. Si en el foro no encuentra la respuesta, plantee en él la consulta.

También puede plantear las dudas a su profesor de prácticas, durante el desarrollo de las mismas o en horario de tutoría.

2. Presentación

2.1. Descripción del sistema.

El trabajo consistirá en modelar el funcionamiento de una **Agenda**. Una Agenda es un conjunto de contactos. Un contacto consta de un número de teléfono, una persona, un domicilio y un correo electrónico. Sobre la Agenda se podrán realizar ciertas operaciones.

El programa debe manejar los siguientes objetos:

- **Contacto**: Un Contacto mantiene información útil sobre una persona o institución. Los contactos pueden contener al menos la siguiente información:
    1. Teléfono
    2. Domicilio
    3. Persona
    4. Correo electrónico

- **Teléfono**: Es una secuencia de dígitos decimales. Por ejemplo, **954487384**.
- **Domicilio**: Es una dirección postal.

- **Persona**: Contiene los datos identificativos de una persona. La información que mantendrá un objeto del tipo Persona es:
    1. Nombre.
    2. Primer Apellido.
    3. Segundo Apellido.
- **Correo Electrónico**: Contiene la información del correo electrónico de un contacto.

- **Agenda**: Una agenda consta de varios Contactos.

2.2. Modelado del programa.

Se proporcionan interfaces que deberán ser implementadas por el alumno, para modelar sistema descrito.

3. Especificación del Programa

El programa a realizar como práctica final de programación orientada a objetos de la asignatura de Fundamentos de la Programación II para la convocatoria de Junio del curso 2011-2012 se ajustará a lo especificado en esta sección.

Todas las clases que el alumno implemente deben pertenecer al paquete **`fp2.poo.pfpooXXX`**, siendo **`XXX`** el login del alumno proporcionado por el Centro de Cálculo (CDC). En el makefile proporcionado y en el código proporcionado aparece **`fp2.poo.pfpoofp2`**, y se deberán realizar las modificaciones oportunas.

Se proporcionan las siguientes interfaces en el paquete **`fp2.poo.utilidades`**:

- **`ContactoInterfaz`**
- **`TelefonoInterfaz`**
- **`DomicilioInterfaz`**
- **`PersonaInterfaz`**
- **`CorreoElectronicoInterfaz`**
- **`AgendaInterfaz`**

También se proporcionan las siguientes excepciones en el paquete **`fp2.poo.utilidades.Excepciones`**:

- **`TelefonoIncorrectoExcepcion`**

  Se lanzará esta excepción cuando se intente instanciar un objeto del tipo **`Telefono`** con un formato incorrecto. Un objeto del tipo **`Telefono`** debe constar de 9 dígitos decimales.

- **`CorreoElectronicoIncorrectoExcepcion`**

  Se lanzará esta excepción cuando se intente instanciar un objeto del tipo **`CorreoElectronico`** con un formato incorrecto. Para verificar que el formato es correcto se deberá comprobar que contiene un símbolo @ y sólo uno.

- **`OperacionNoPermitidaExcepcion`**

  Esta excepción se lanza cuando se intenta realizar una operación en la Agenda que no está permitida.

En esta práctica final se deben implementar todas las interfaces mencionadas anteriormente.
La interfaz **AgendaInterfaz** se implementará en la clase **Agenda**. La clase **Agenda** representa un conjunto de Contactos agrupados en una matriz de objetos del tipo **ContactoInterfaz**. El tamaño se proporciona en la interfaz **AgendaInterfaz**, Mediante el atributo estático:

```
public final static int MAX_NUM_CONTACTOS = 1000;
```

A continuación se muestra la interfaz **AgendaInterfaz**, donde se presentan los métodos de esta interfaz comentados (que deben ser implementados).

```
/*
 *  @(#)AgendaInterfaz.java
 *
 *  Fundamentos de Programacion II. GITT.
 *  Departamento de Ingenieria Telematica
 *  Universidad de Sevilla
 */

package fp2.poo.utilidades;

import fp2.poo.utilidades.ContactoInterfaz;
import fp2.poo.utilidades.TelefonoInterfaz;
import fp2.poo.utilidades.Excepciones.OperacionNoPermitidaExcepcion;

/**
 * Esta es una clase que representa una agenda de telefonos.
 * Mantiene un conjunto de contactos.
 *
 * @version version 1.0 Abril 2012
 * @author  Fundamentos de Programacion II
 */
public interface AgendaInterfaz {

    /**
     * Maximo numero de contactos que puede incluir la agenda.
     */
    public final static int MAX_NUM_CONTACTOS = 1000;

    /**
     *  Inserta un nuevo contacto, proporcionandolo como parametro.
     *  En el caso de superar el numero maximo de contactos permitidos
     *  o en el caso de que ya exista un contacto con el mismo numero de telefono
     *  se lanzara la excepcion OperacionNoPermitidaExcepcion-
     */
    public void insertarContacto(ContactoInterfaz contacto)
           throws OperacionNoPermitidaExcepcion;

    /**
     *  Modifica el telefono de un contacto.
     *  Se proporciona como parametro el telefono del contacto a modificar
     *  y el nuevo telefono.
     *  Si el numero de telefono no existe se lanzara la excepcion
     *  OperacionNoPermitidaExcepcion.
     */
    public void modificarTelefono(TelefonoInterfaz telefonoAntiguo,
                                  TelefonoInterfaz telefonoNuevo)
            throws OperacionNoPermitidaExcepcion;

    /**
     *  Elimina un contacto de la agenda.
     *  Se proporciona como parametro el numero de telefono del contacto a eliminar
     *  Si el telefono indicado no esta en la agenda
     *   se lanzara la excepcion OperacionNoPermitidaExcepcion.
     */
    public void eliminarContacto(TelefonoInterfaz telefono)
            throws  OperacionNoPermitidaExcepcion;
```

```
    /**
     *  Muestra por la salida estandar (System.out) todos los contactos que hay en la
     *  agenda.
     *  El orden de salida de los datos es el siguiente:
     *  Numero de telefono, Nombre, Primer Apellido, Segundo Apellido, Domicilio y
     *  Correo electronico.
     *  Cada uno de estos datos debe ir en una l¡nea diferente.
     *  No se inserta  ninguna informacion adicional a la especificada.
     */

    public void mostrarContactos();

}
```

AgendaInterfaz.java

La siguiente tabla presenta un resumen de las clases e interfaces utilizados y los ficheros en los que se implementan.

Clase	*Fichero en que se implementa la clase*	*Interfaz*	*Fichero en que se especifica la interfaz*
`Contacto`	`Contacto.java`	`ContactoInterfaz`	`ContactoInterfaz.java`
`Telefono`	`Telefono.java`	`TelefonoInterfaz`	`TelefonoInterfaz.java`
`Domicilio`	`Domicilio.java`	`DomicilioInterfaz`	`DomicilioInterfaz.java`
`Persona`	`Persona.java`	`PersonaInterfaz`	`PersonaInterfaz.java`
`CorreoElectronico`	`CorreoElectronico.java`	`CorreoElectronicoInterfaz`	`CorreoElectronicoInterfaz.java`
`Agenda`	`Agenda.java`	`AgendaInterfaz`	`AgendaInterfaz.java`

También se proporcionan la interfaz **`Iteracion`** implementada en la clase **`DatosDeEntrada`**. **`DatosDeEntrada`** permite la lectura de los datos de los contactos de la agenda desde un fichero (Se especificará en la sección Lectura del Fichero de Entrada).

3.1 Argumentos de la Línea de Comandos.

El programa aceptará un argumento en línea de comandos. Este argumento es el nombre del fichero que contiene los datos de los contactos de la agenda.
Si no se proporciona el nombre del fichero o si se genera algún error en la apertura del fichero, se lanza la excepción **`OperacionNoPermitidaExcepcion`**. Se imprime por la salida de errores un mensaje y finaliza el programa. Los mensajes correspondientes son los siguientes:

- `"Numero de argumentos incorrecto"`
- `"Error en apertura de fichero"`

3.2 Fichero de entrada.

El fichero de entrada cuyo nombre se proporciona como argumento en la línea de comandos se ubicará en el directorio **`./pfpoo/src/fp2/poo/datos`**.

3.3 Formato del fichero de entradas.

En el fichero de entradas se especifican los datos de cada cuenta en el siguiente orden:

- Teléfono
- Domicilio
- Nombre
- Primer Apellido
- Segundo apellido y
- Correo electrónico.

En el fichero de entradas, cada contacto va separado del siguiente por una línea de guiones.

3.4 Lectura de los datos de entrada.

Para la lectura del fichero de contactos se proporciona la clase **`DatosDeEntrada`**, (ya implementada) que implementa la interfaz **`Iteracion`** que contiene dos métodos:

- **`hayMasContactos`**`()`
  (**`public boolean hayMasContactos()`**) este método devuelve **`true`** si hay más contactos en el fichero.
- **`siguienteContacto`**`()`
  (**`public ContactoInterfaz siguienteContacto()`**) este método devuelve el siguiente contacto en una referencia del tipo **`ContactoInterfaz`**, sino hay más contactos y se intenta obtener con este método se lanzará la excepción **`OperacionNoPermitidaExcepcion`**.

De esta forma todas los contactos se pueden obtener en el constructor de la clase **`Agenda`** de la siguiente forma (que no se proporciona en los ficheros entregados para la realización de la práctica final):

```
public class Agenda implements AgendaInterfaz {

    private ContactoInterfaz matriz [] = null;

    private int numeroContactosUsados  = 0;

    /**
     * Maximo numero de contactos que puede incluir la agenda.
     */
    public final static int MAX_NUM_CONTACTOS = 1000;

    public Agenda ( String archivo ) throws OperacionNoPermitidaExcepcion {
        this.matriz  = new ContactoInterfaz[MAX_NUM_CONTACTOS];
        DatosDeEntrada  obj = null;

        try {
            obj = new DatosDeEntrada( archivo );
            while(obj.haySiguienteContacto()) {
                ContactoInterfaz contacto = obj.siguienteContacto();
                this.insertarContacto ( contacto );
            }
        } catch (OperacionNoPermitidaExcepcion e) {
            throw e;
        }
    }
//la clase Agenda continúa …
```

Extracto de la clase `Agenda.java`

3.5 Resultados del programa.

Los resultados se proporcionarán por la salida estándar (**`System.out`**) mediante el método **`println`**. Los datos se generan mediante el método **`mostrarContactos`**(`public void` **`mostrarContactos`** `()`) especificado en **`AgendaInterfaz`**.

3.6 Errores de ejecución.

En caso de que en el fichero de entrada se proporcione un valor correspondiente al teléfono o al correo electrónico en un formato incorrecto el código proporcionado de **`DatosDeEntrada`** no tiene en cuenta esta entrada, y lee el siguiente contacto si lo hubiere.

3.7 Pruebas del programa.

El programa se probará mediante clases que contengan el método **`main`**. Estas clases estarán en el paquete **`fp2.poo.principal`**. Se proporciona una clase de ejemplo, que crea una agenda y realiza una operación sobre un contacto.

4 Recomendaciones finales

4.1 Organización del trabajo.

Los ficheros deben de estar organizados según el enunciado de la práctica.

4.2 Normas para la defensa.

Para evaluar el Trabajo de Curso el alumno deberá hacer una defensa personal del trabajo presentado. La defensa del trabajo de curso es un examen donde el alumno debe responder claramente a las preguntas que formulen los profesores sobre el diseño y realización del trabajo. El alumno debe saber explicar la estructura de sus programas y su funcionamiento. Como se ha indicado anteriormente, durante la defensa podrá solicitarse del alumno que realice pequeñas modificaciones sobre el código.

4.3 Documentación a entregar.

El trabajo se entregará en la plataforma virtual según las normas de la asignatura.
Se valorará positivamente la inclusión en el código de los comentarios `javadoc` y la entrega de la documentación generada mediante la herramienta `javadoc` en el directorio `./doc`.
También se valorará positivamente la generación del fichero `.jar`.