Lista de Personas. Analizando Código

Fecha entrega: 16/03/15

Autor: **Pedro J. Ramos**
Estela Muñoz

Analiza el código Solución. Lista de personas y responde a las siguientes preguntas:

1. **Enumera las clases implementadas.**

- Lista Personas
- Persona
- TestListaPersonas
- PersonaTest
- Menu
- Teclado

Pattern y Matcher son clases heredadas de java.util, que ya vienen predefinidas y son utilizadas.

2. **Indica aquellas que tienen comunicación con el usuario**

Teclado que es llamada desde la clase TestListaPersonas.

3. **Indica cuántas de ellas utilizan la clase java.util.ArrayList y para qué**

- ListaPersonas : Para crear una arrayList del tipo persona
- TestListaPersonas : para crear un arrayList de las personas de una provincia concreta

4. **Dentro de la clase Persona:**

- **Total de constructores definidos, así como su visibilidad. Justifícalo.**

Existen un total de 5 constructores:

- Hay 3 *Pattern*, utilizados para nombre, *fecha de nacimiento* y código postal respectivamente.
- Hay 2 *Persona*, uno con un único atributo *id* y otro con los atributos *nombre, primer apellido, segundo apellido, fecha de nacimiento y código postal.*

Son todos privados. En el caso de los patrones porque sólo son utilizados dentro de la misma clase Persona, y en el caso de los constructores de Persona por el mismo motivo, ya que se crea un objeto Persona dentro de los métodos instanciarPersona(), que son friendly al ser llamados desde otras clases del mismo paquete.

- **Posibilidad de crear dos personas con el mismo identificador. Por qué.**

No se puede crear dos personas con el mismo *id* ya que este campo es un campo autocalculado que se genera cuando se crea el objeto y no a través de argumentos pasados al constructor.

- **Por qué se definen los métodos equals y hashCode. Métodos que los utilizan.**

  Son definidos y reescritos para poder utilizarse para establecer criterios de búsqueda y ordenamiento. De forma directa no es utilizado ya que no es llamado desde esta clase ni desde otra externa, pero de forma indirecta sí es utilizado por el *contains()* (usa a equals()) y *remove()* (usa a hashCode()).

- **Métodos de clase definidos. Por qué.**

  - instanciarPersona (int id): Crea una persona con único campo *id.*
  - instanciarPersona (String nombre, String primerApellido, String segundoApellido, String fechaDeNacimiento, String codigoPostal) : Crea un objeto persona si todos los parámetros pasados son validos
  - esValidoCP(String codigoPostal): Comprueba que el código postal es valido
  - esValidaFecha(String fechaDeNacimiento): Comprueba que la fecha de nacimiento es válida
  - esValidoNombre(String nombre): Comprueba que el nombre es válido.
  - toString(): Muestra los atributos de objeto en formato String
  - hashCode(): Sobreescribe el hasCode() por defecto
  - equals(Object obj): Sobreescribe el equals() por defecto
  - getProvincia(): Extrae los primeros dígitos del código postal en formato numerico
  - getCodigoPostal(): Devuelve el codigoPostal del objeto

- **Atributos de clase definidos. Por qué.**
  - patternNombre: Patrón para comprobar el nombre
  - patternFecha: Patrón para comprobar la fecha de nacimiento
  - patternCodigoPostal: Patrón para comprobar código postal
  - nombre: guarda el nombre de la persona
  - primerApellido: guarda el primer apellido de la persona
  - segundoApellido: guarda el segundo apellido de la persona
  - fechaDeNacimiento: guarda la fecha de nacimiento de la persona
  - codigoPostal: guarda el código postal de la persona
  - id: campo autoincremental que guarda un identificador inequívoco de la persona

Indica también:

- **Clase donde se crean los objetos Persona.**

  Se crea en la clase persona en el método instanciarPersona que se encuentra sobrecargado; en uno de ellos se crea una persona sólo con el id y en el otro con todos los campos.

- **Requisitos para añadir una persona a la lista. Dónde se controla y por qué.**

  Siempre que sean válidos los atributos *nombre, primerApellido, segundoApellido, fechaDeNacimiento y codigoPostal*.

- **Requisitos para eliminar una persona de la lista. Dónde se controla**

  Se requiere que exista una persona con el id indicado. Se controla con el remove() que llama automáticamente al hashCode(), donde se comprueba que el objeto creado tiene el mismo id que uno que ya exista en la lista.

- **Qué sucede si la persona a eliminar no existe.**

  Que no se elimina ya que no hay una coincidencia de id.

- **Cómo se genera el número de Personas de una provincia.**

En el método getDeProvincia, de la clase TestListaDePersonas, crea una lista de las personas de una misma provincia y luego muestra el tamaño de esa lista.

En TestListaPersonas:

1. **Existe un bucle infinito. ¿Cómo se sale de él?**

   Existe un do…while(true) del que se sale cuándo se escribe un número que no está contenido en los case del switch (del 1 al 4); o dicho de otro modo, que no pertenece a una de las opciones del menú.

2. **Clases y métodos utilizados para añadir una Persona en la lista**

   Se usa el método annadirPersona() que utiliza el método annadir() de la clase ListaPersonas, en el cuál se hace una comprobación de si ya existe, para lo que usa el método contains() de la clase Persona, que a su vez llama de forma automática al método equals(). También se comprueba en annadir() si es null. En el caso de que ya exista o sea null, no crea a la persona para añadirla y, en caso contrario, lo hace.

3. Clases y métodos utilizados para eliminar una Persona de la lista

Se usa el método eliminarPersona() que utiliza el método eliminar() de la clase ListaPersonas, el cuál usa el método remove(), que a su vez llama al método instanciarPersona(id) de la clase Persona. Así mismo, el método remove() de forma automática llama al método hashCode() que compara si es el mismo objeto o no por el id.