**PROGRAMACIÓN ORIENTADA A OBJETOS**

**Excepciones**

**Octubre 2017**

**Laboratorio 4/6**

Santiago Martínez Martínez

**EQUIPOS**

EN BLUEJ

**PRACTICANDO MDD Y BDD con EXCEPCIONES**

1. En su directorio descarguen los archivos contenidos en equipo.zip, revisen el contenido y estudien el diseño estructural de la aplicación.
2. ¿De qué clase no pueden consultar las fuentes? ¿En qué formato está? ¿Por qué?

Estudien su documento en el diseño o en el código.

De la clase Persona, al abrirlo abrir el editor del codigo nos manda a la documentación de la clase, se cree que esto pasa porque en el directorio es la única clase que no tiene su archivo .java, solo su archivo .class

Esta clase tiene dos atributos:

* Datos
* Nombres

Esta clase tiene dos métodos:

* Su constructor
* ValorHora

1. Dada la especificación, diseñen, codifiquen el método “valorHora”.

Archivo .astah y .java

1. Dada la especificación, diseñen, codifiquen y prueben el método “ValorHoraEstimado”.

Archivo .astah y .java

**EN CONSOLA**

**Conociendo el proyecto SINAP**

1. En su directorio descarguen los archivos en SINAP.zip, revisen el contenido.

¿Cuántos archivos se tienen? ¿Cómo están organizados? ¿Cómo deberían estar organizados?

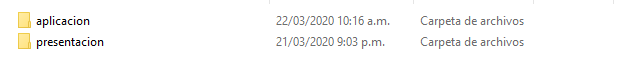
* Tenemos 5 archivos, 4 .java y un archivo .astah
* No tienen organización alguna, todos iban en el mismo archivo
* Los archivos .java deben de ir en la carpeta src y el archivo .astah en ninguna de las tres nombradas anteriormente

1. Estudien el diseño del programa: diagramas de paquetes y de clases. ¿cuántos paquetes tenemos? ¿cuántas clases tiene el sistema? ¿cómo están organizados? ¿cuál es la clase ejecutiva?

* Hay dos paquetes: aplicación y presentación.
* Cuatro clases tiene el sistema
* La clase Ejecutiva es SinapGUI

1. ****Prepare los directorios necesarios para ejecutar el proyecto. ¿qué estructura debe tener? ¿qué instrucciones debe para ejecutarlo?

Dentro del directorio src



* El comando utilizado para copilar:

**javac -d bin src/presentacion/\*.java src/aplicacion/\*.java**

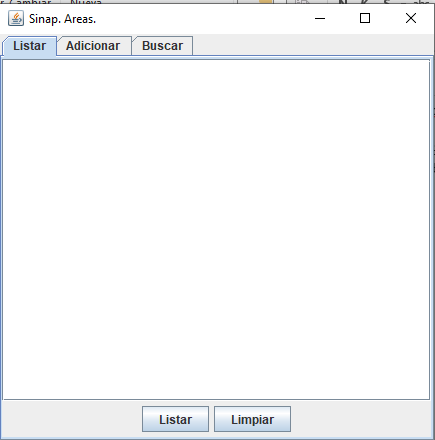
1. Ejecute el proyecto, ¿qué funcionalidades ofrece? ¿cuáles funcionan?

Realice el diagrama de casos de uso correspondientes

Comando Utilizado Para Ejecutar:

java -cp bin presentacion.SinapGUI

* Funcionalidades

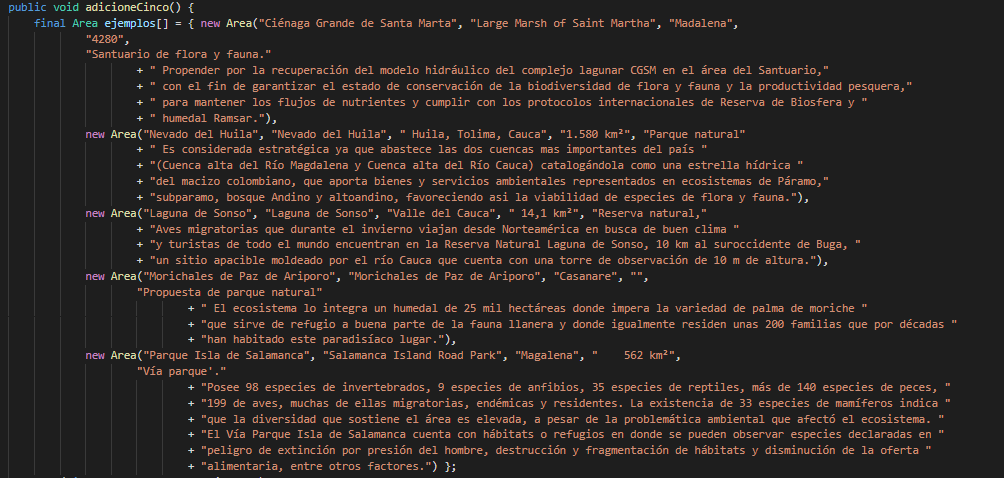


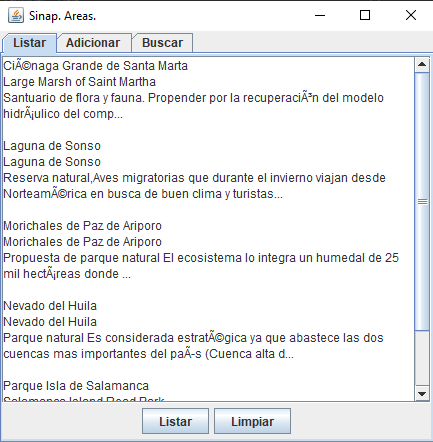
* Sirve las funcionalidades Listar, Adicionar y Buscar. El problema que encontramos es que al cerrar el programa los parques nacionales que añadimos se eliminan y quedan los definidos por defecto, esto se debe a que aún no tiene base de datos.

1. ¿De dónde sales las áreas iniciales? Revisen el código y la documentación del proyecto.

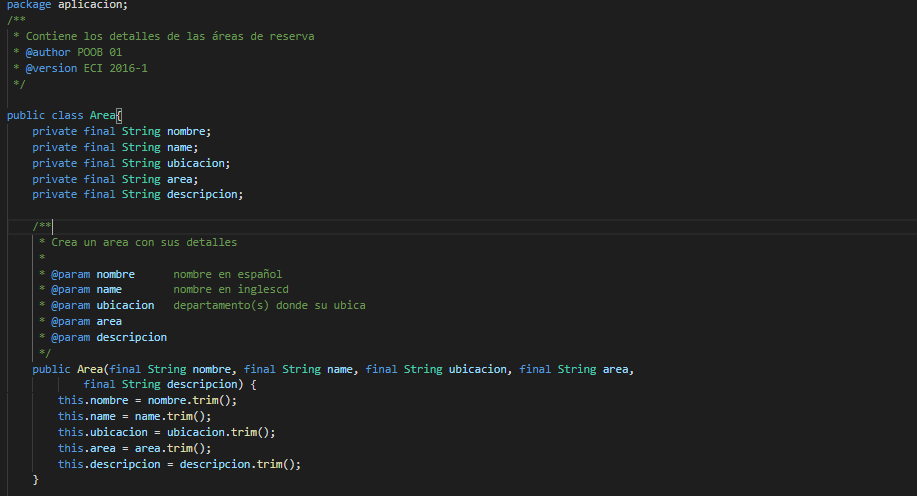
¿Qué clase pide que se adicionen? ¿Qué clase los adiciona?

* Las primeras áreas se adicionan desde un método de sinap el cual es adicioneCinco(), aquí se añaden las áreas que aparecen al ejecutar el programa.





* La clase que pide que adicionen las nuevas áreas es la clase sinap en el método mencionado anteriormente.
* La clase que los adiciona es la clase Area del paquete presentación



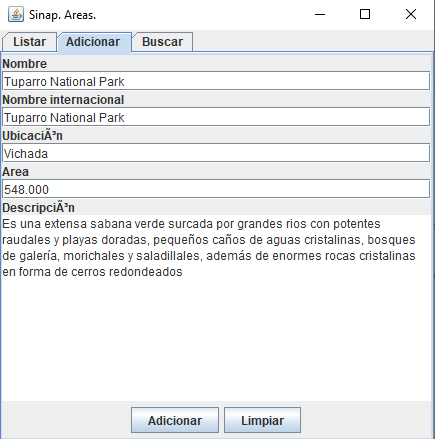
**Adicionar y listar. Todo OK.**

El objetivo es realizar ingeniería reversa a las funciones de adicionar y listar.

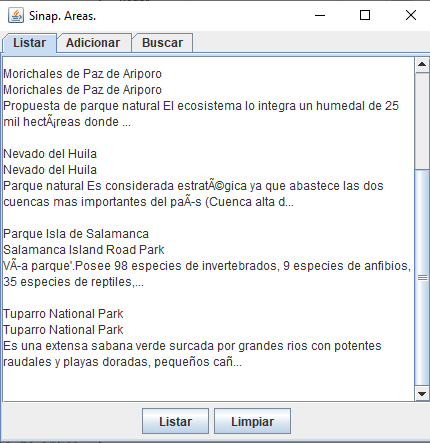
1. Adicionen una nueva área

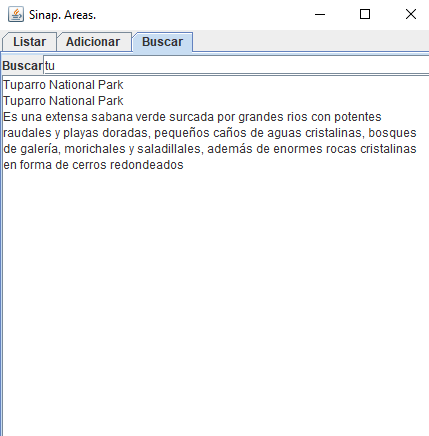
¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla ¿Es adecuado este comportamiento?

* Que ocurre: Al darle adicionar no pasa nada, no muestra mensaje alguno de que logro adicionar la nueva area



* Se comprueba en la funcionalidad de listar, al listar las áreas que hay, o en buscar con el nombre del parque

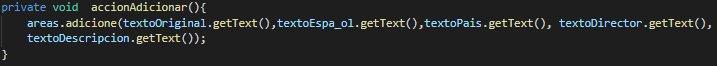




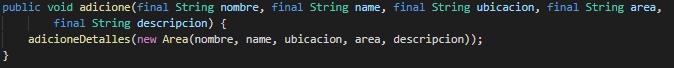
* Creemos que sí, ya que adiciono a la lista el área nueva y en el buscador esta

1. Revisen el código asociado a **adicionar** en la capa de presentación y la capa de aplicación. ¿Qué método es responsable en la capa de presentación? ¿Qué método en la capa de aplicación?

* El método encargado de crear la ventana adicionar es prepareAreaAdicionar(), en el caso de ya haber presionado el botón es el método accionAdionar()



* El método encargado de adicionar en la capa aplicación es el método adicione() de la clase Sinap()



1. Realicen ingeniería reversa para la capa de aplicación para **adicionar**.

Capturen los resultados de las pruebas de unidad.

1. Revisen el código asociado a **listar** en la capa presentación y la capa de aplicación.

¿Qué método es responsable en la capa de presentación? ¿Qué método en la capa de aplicación?

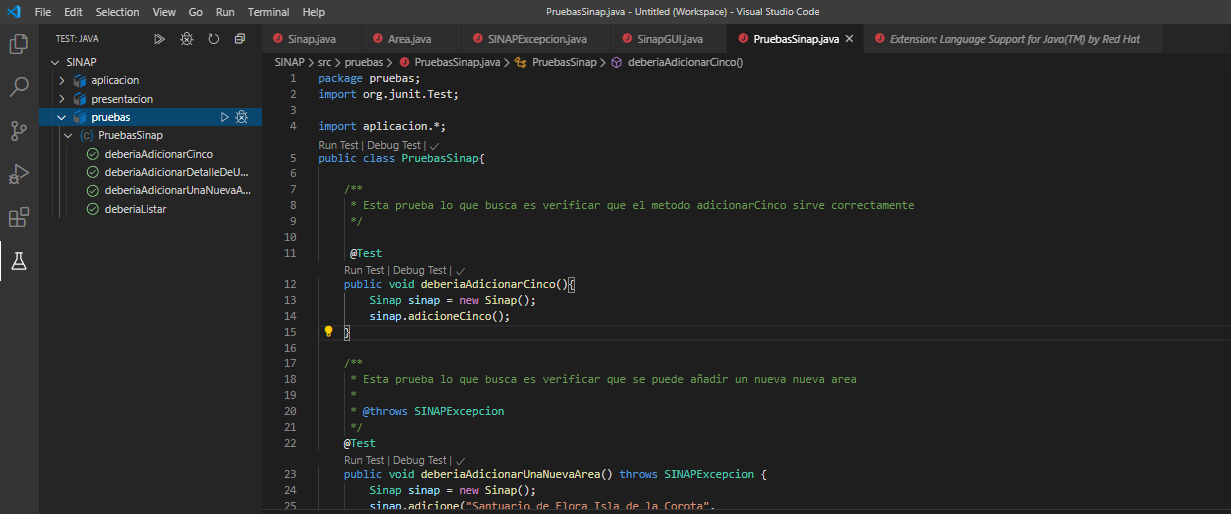
* En la capa presentación el método encargado de **listar**, es el método accionListar().
* En la capa aplicación el método encargado es toString(), este método nos consulta todas las áreas que tenemos y asi mismo las retorna.

1. Realicen ingeniería reversa para la capa de aplicación para **listar.**

Capturen los resultados de las pruebas de unidad.

Archivo .astah

1. Propongan y ejecuten una prueba de aceptación.



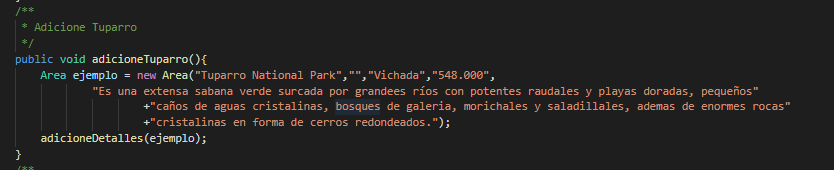
**Adicionar un área. ¡Y su no da un nombre?**

El objetivo es perfeccionar la funcionalidad de adicionar un área.

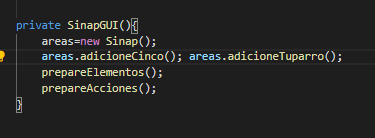
1. Adicionen el área Tuparro sin nombre internacional. ¿Qué ocurre? ¿Cómo lo comprueban?

Capturen la pantalla. ¿Es adecuado este comportamiento?

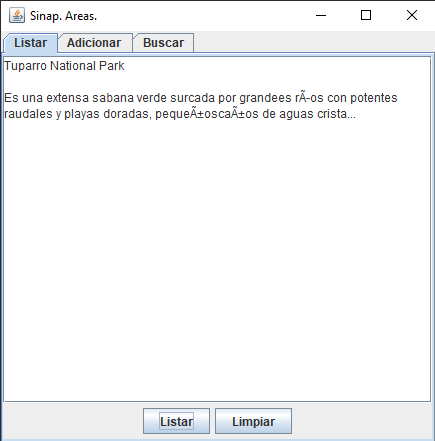
Adición de un método para adicionar Tuparro:



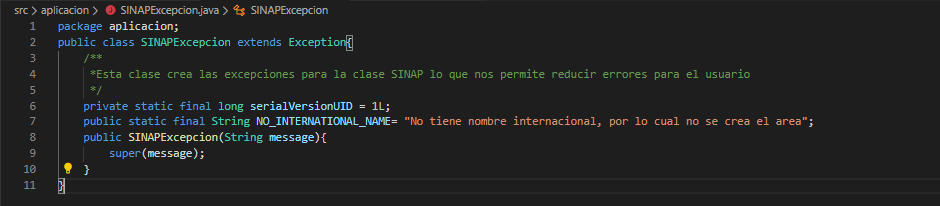
Adicion del método Tuparro en SinapGUI:



Como lo comprueban:

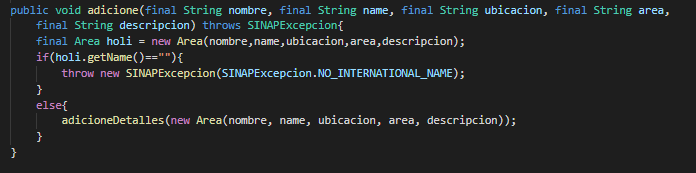


1. Vamos a evitar la creación de áreas con un nombre internacional vacío manejando una excepción **SINAPExcepcion**. Si el área no tiene nombre internacional, no la creamos y se lo comunicamos al usuario. Para esto lo primero que debemos hacer es crear la nueva clase **SINAPExcepcion** Considerando este primer mensaje.



1. Analicen el diseño realizado. ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.

El método que debería de lanzar al excepción debe de ser adicione:

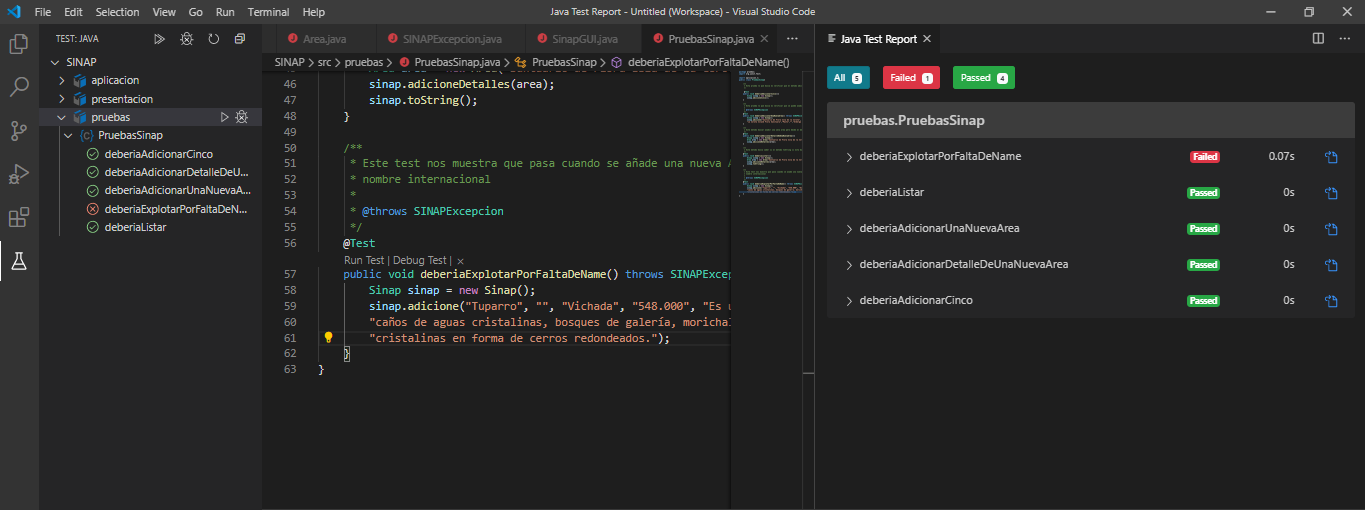


Los que deberían propagarla serían los siguientes métodos:

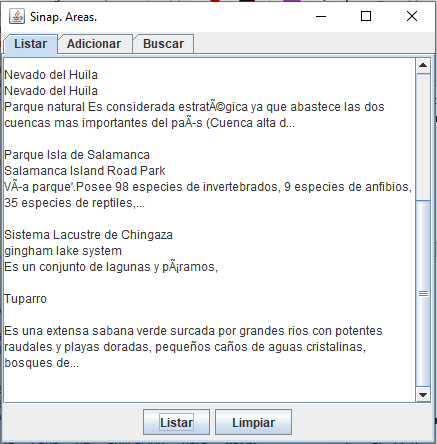
1. Clase SinapGUI:

* Main
* AccionAdicionar
* Constructor Sinap

1. Construya la solución propuesta. Capturen los resultados de las pruebas.



1. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1. ¿Qué sucede ahora? Capture la pantalla.



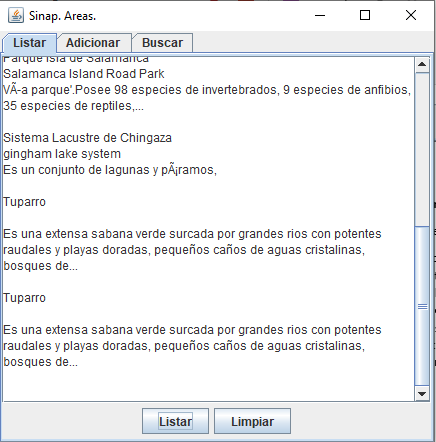
**Adicionar un área. ¿Y si ya se encuentra?**

EL objetivo es perfeccionar la funcionalidad de adicionar un área.

1. Adicionen dos veces la nueva área Tuparro ¿Qué ocurre? ¿Cómo lo comprueban?

Capturen la pantalla. ¿Es adecuado este comportamiento?

Al añadir dos veces Tuparro, en este caso nos dirigimos a listar y nos muestra dos veces la información, esto no debería pasar, si ya está, no se debería añadir.



1. Analicen el diseño realizado. ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.

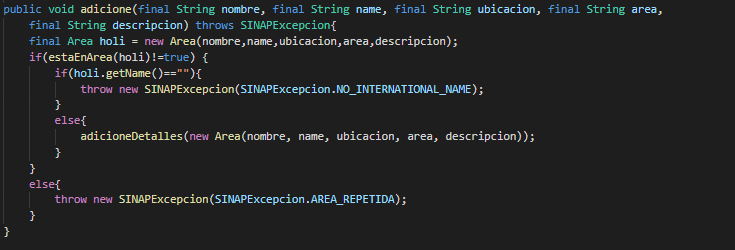
El método que debería lanzar la excepción cuando allá ya esa área debería ser adicione() de la clase Sinap.

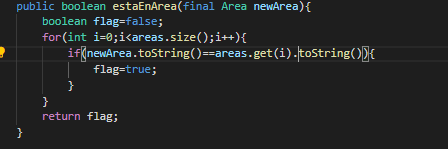
Los que deberían propagarla serían los siguientes métodos:

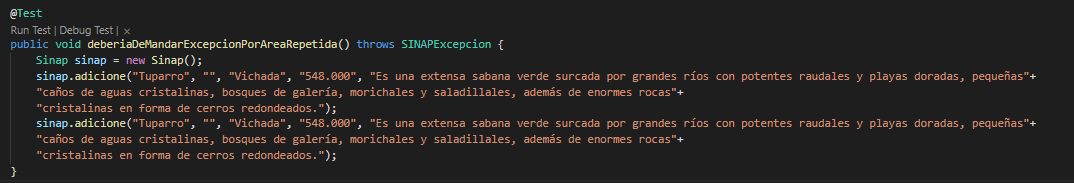
1. Clase SinapGUI:

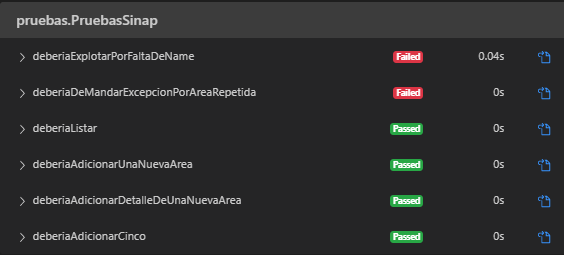
* Main
* AccionAdicionar
* Constructor Sinap

1. Construya la solución propuesta. Capturen los resultados de las pruebas.









1. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.

