**PROGRAMACIÓN ORIENTADA A OBJETOS**

**Excepciones**

**Octubre 2017**

**Laboratorio 4/6**

Santiago Martínez Martínez y Ricardo Amaya

**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?

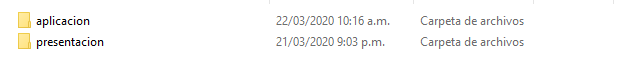
* Tenemos 5 archivos, 4 .java y un archivo .astah
* No tienen organización alguna, todos iban en el mismo archivo
* Los archivos .java deben de ir en la carpeta src y el archivo .astah en ninguna de las tres nombradas anteriormente

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**

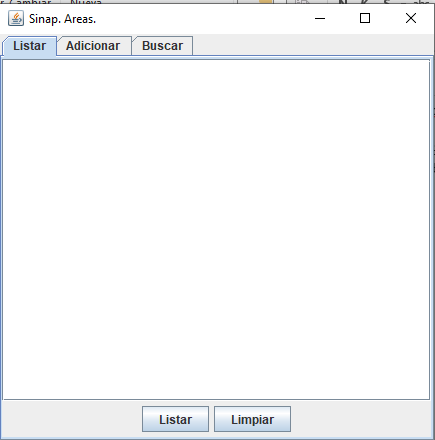
1. Ejecute el proyecto, ¿qué funcionalidades ofrece? ¿cuáles funcionan?

Realice el diagrama de casos de uso correspondientes

Comando Utilizado Para Ejecutar:

java -cp bin presentacion.SinapGUI

* Funcionalidades

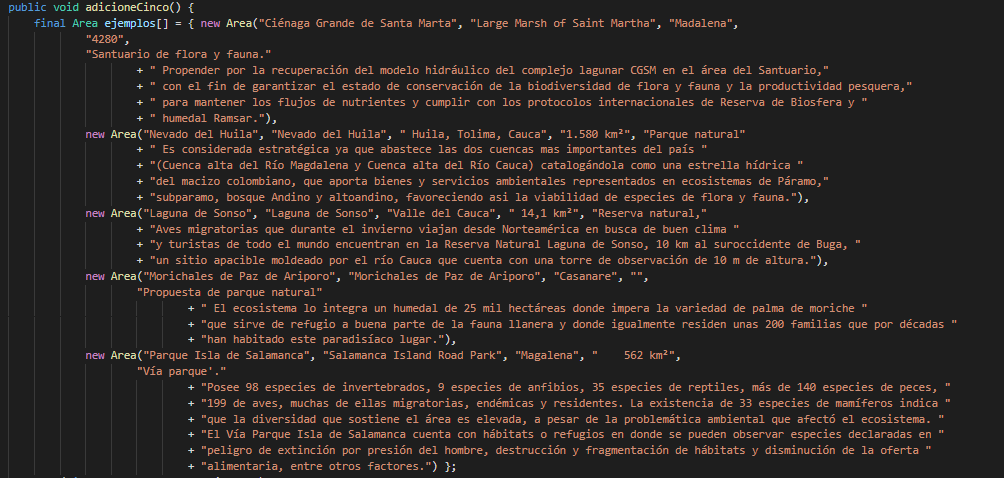


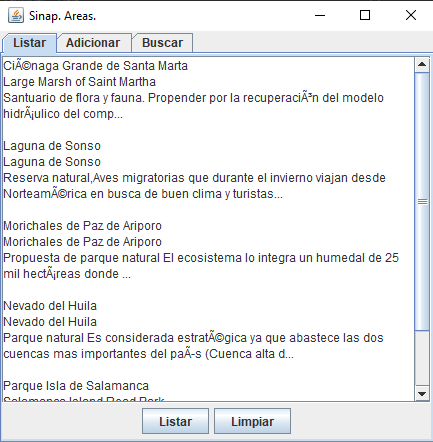
* Sirve las funcionalidades Listar, Adicionar y Buscar. El problema que encontramos es que al cerrar el programa los parques nacionales que añadimos se eliminan y quedan los definidos por defecto, esto se debe a que aún no tiene base de datos.

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

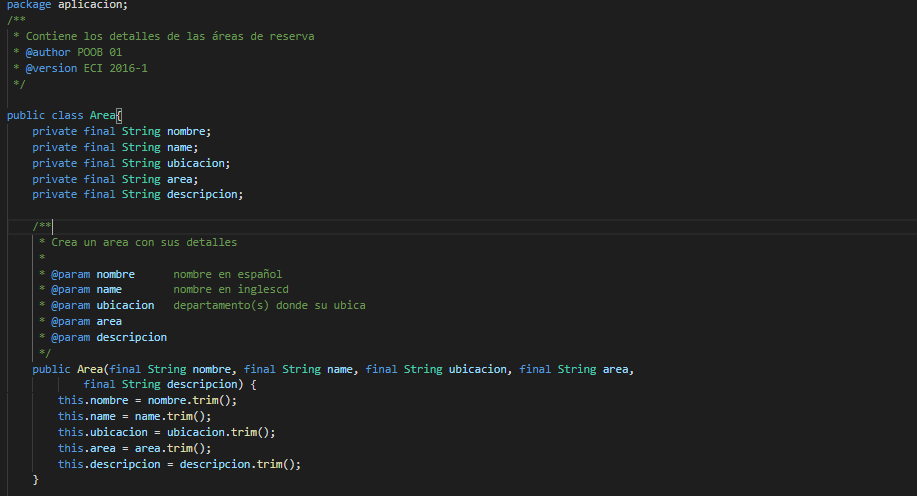
¿Qué clase pide que se adicionen? ¿Qué clase los adiciona?

* 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



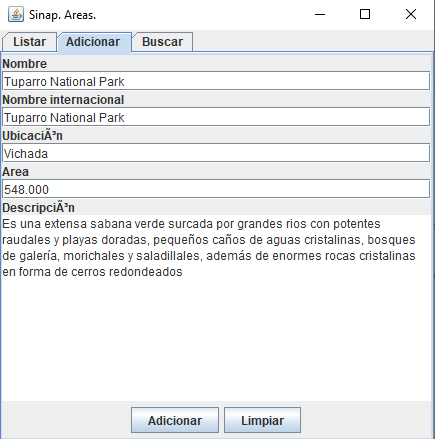
**Adicionar y listar. Todo OK.**

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

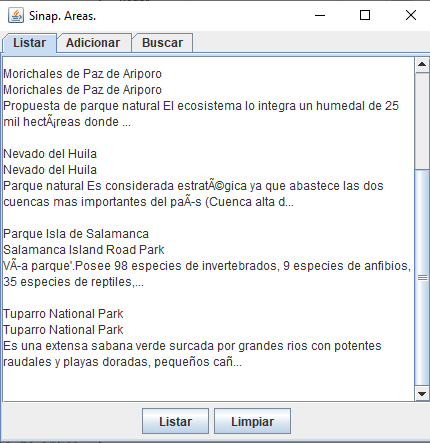
1. Adicionen una nueva área

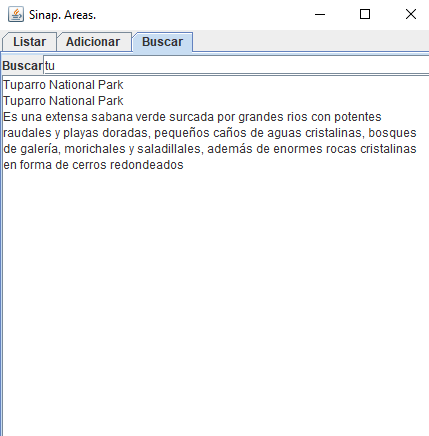
¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla ¿Es adecuado este comportamiento?

* 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

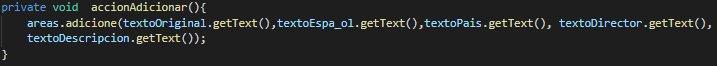




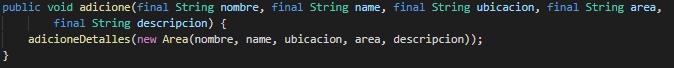
* Creemos que sí, ya que adiciono a la lista el área nueva y en el buscador esta

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?

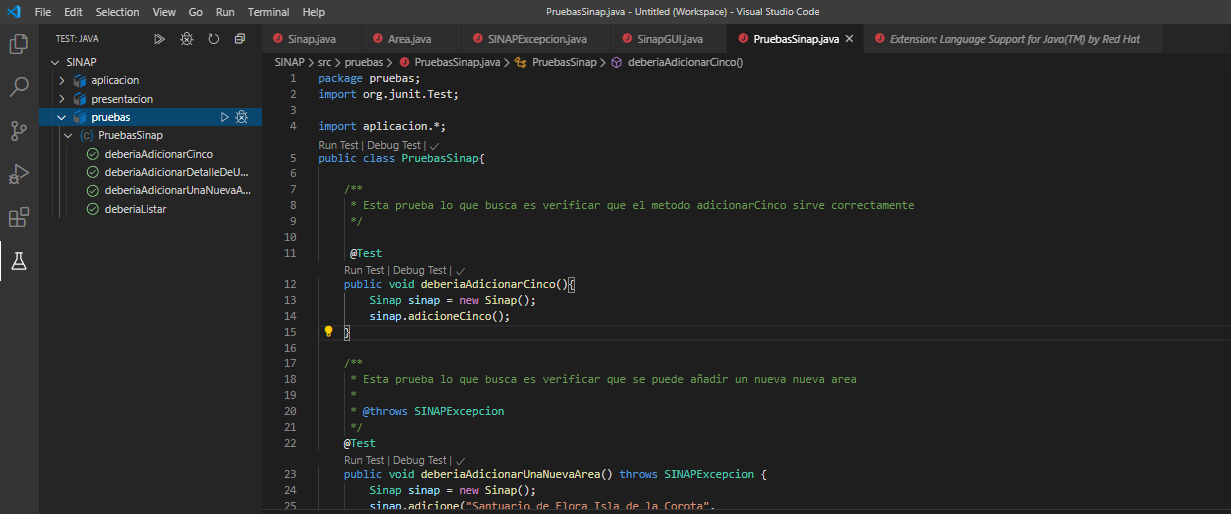
* En la capa presentación el método encargado de **listar**, es el método accionListar().
* En la capa aplicación el método encargado es toString(), este método nos consulta todas las áreas que tenemos y asi mismo las retorna.

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.



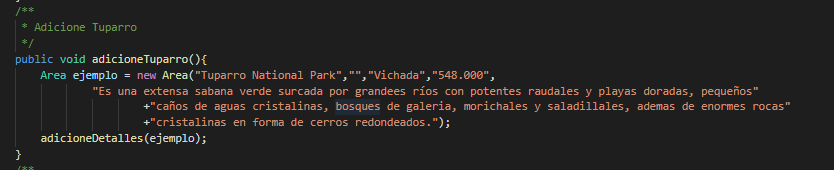
**Adicionar un área. ¡Y su no da un nombre?**

El objetivo es perfeccionar la funcionalidad de adicionar un área.

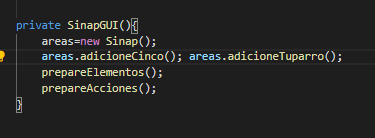
1. Adicionen el área Tuparro sin nombre internacional. ¿Qué ocurre? ¿Cómo lo comprueban?

Capturen la pantalla. ¿Es adecuado este comportamiento?

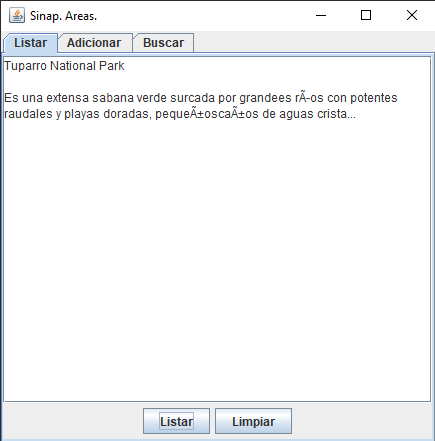
Adición de un método para adicionar Tuparro:



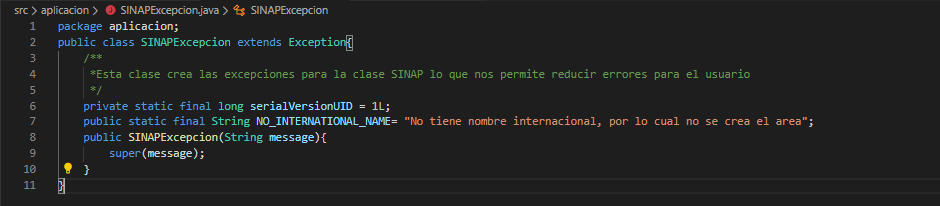
Adicion del método Tuparro en SinapGUI:



Como lo comprueban:

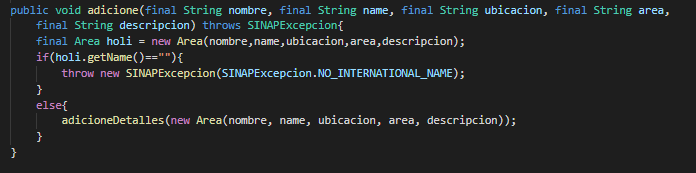


1. Vamos a evitar la creación de áreas con un nombre internacional vacío manejando una excepción **SINAPExcepcion**. Si el área no tiene nombre internacional, no la creamos y se lo comunicamos al usuario. Para esto lo primero que debemos hacer es crear la nueva clase **SINAPExcepcion** Considerando este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

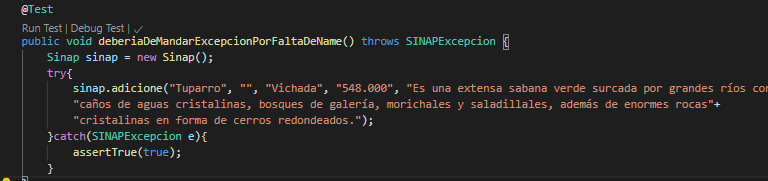


Los que deberían propagarla serían los siguientes métodos:

1. Clase SinapGUI:

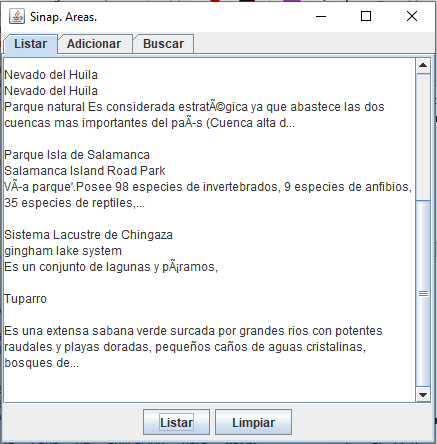
* Main
* AccionAdicionar
* Constructor Sinap

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.



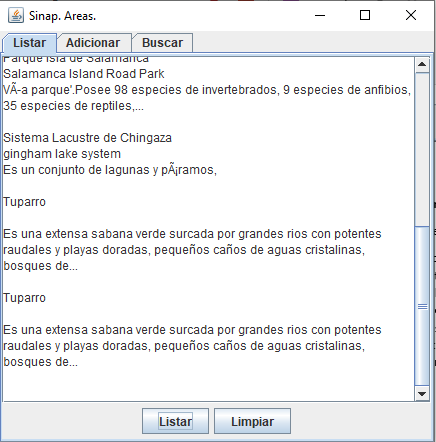
**Adicionar un área. ¿Y si ya se encuentra?**

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.

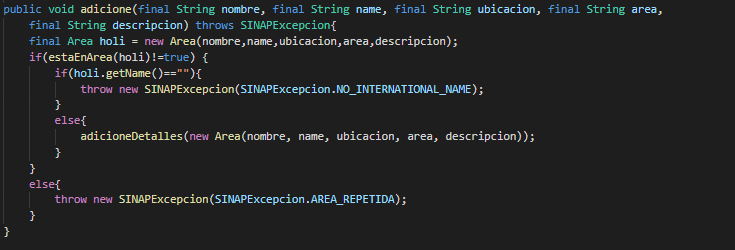
El método que debería lanzar la excepción cuando allá ya esa área debería ser adicione() de la clase Sinap.

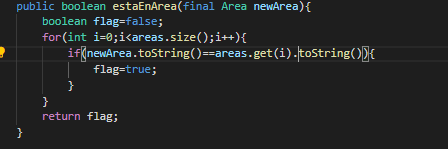
Los que deberían propagarla serían los siguientes métodos:

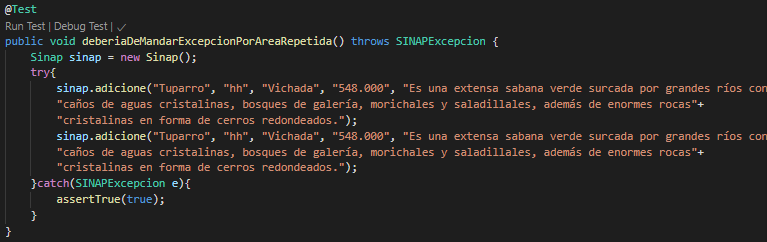
1. Clase SinapGUI:

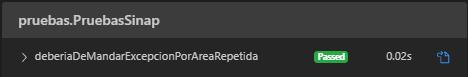
* Main
* AccionAdicionar
* Constructor Sinap

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

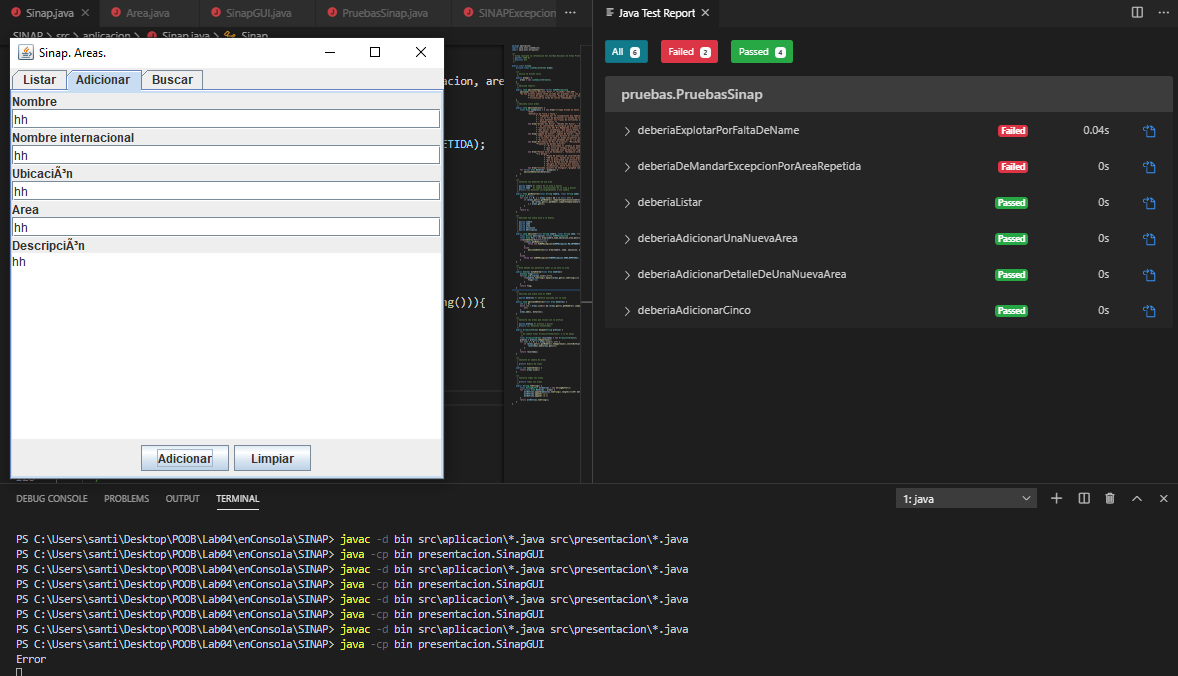








1. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.



**Adicionar un área. ¿Otras condiciones?**

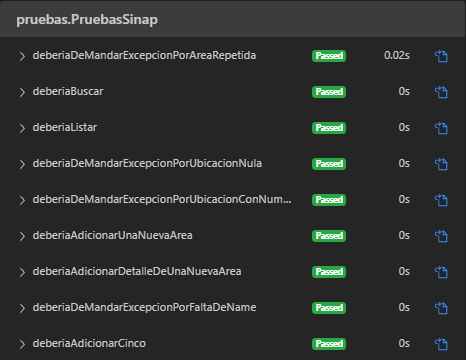
El objetivo es perfeccionar la funcionalidad de adicionar un área.

1. Propongan la solución propuesta de un área sea más robusta.

La ubicación no puede ser nula y no pueden tener números.

1. Construya la solución propuesta (diseño, prueba de unidad, codigo)

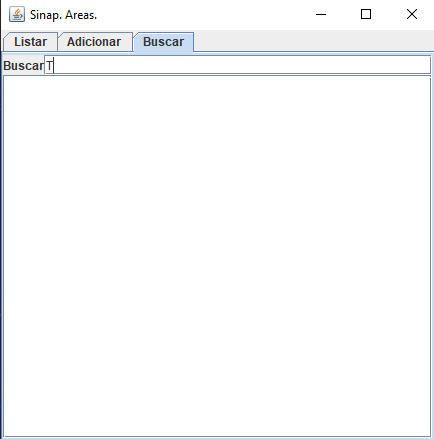
Capturen los resultados de las pruebas.



**Consultando por patrones. ¡No funciona y queda sin funcionar!**

1. Consulten un área especial que inicie con T. ¿Qué sucede? ¿Qué creen que pasó?

Capturen el resultado. ¿Quién debe conocer y quien NO debe conocer esta información?



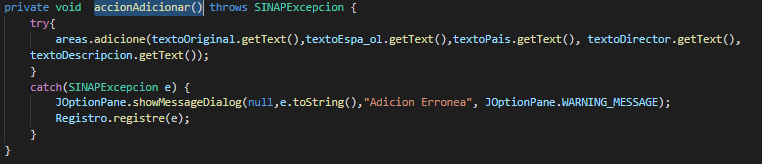
No nos muestra información alguna, esta información la deben de conocer los usuarios ya así podemos buscar las áreas que están en la lista dependiendo de su prefijo.

1. Exploren el método registre de la clase Registro ¿Qué servicio presta?

Esta clase nos ofrece el método registre el cual tiene como parámetro **“e”** el cual es de tipo exception.

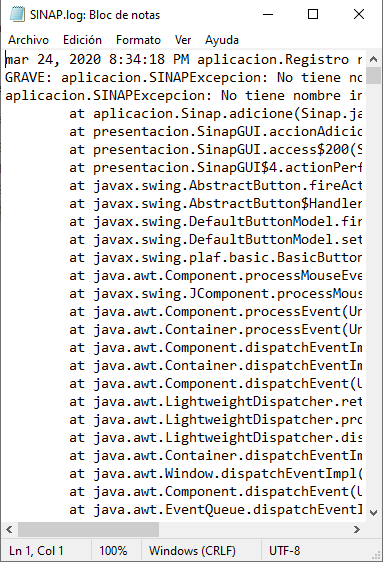
1. Analicen el punto adecuado para que SIEMPRE, al sufrir en cualquier punto el sistema un incidente como este, se presente un mensaje especial de alerta al usuario, se guarde la información del error en el registro de error y termine la ejecución. Expliquen y construyan la solución.

El punto donde debe de añadirse el registro es en el método de la clase SinapGUI el cual les accionAdicionar(), ya que este método nos recopila la información que tenemos de la aplicación y la redirige a la clase Sinap el cual manda las Excepciones. Entonces allí debemos de poner el try y el catch, para cuando tengamos una exception la registre.



1. Ejecuten nuevamente la aplicación con el caso propuesto en 1. ¿Qué mensaje salió en la pantalla? ¿La aplicación termina? ¿Qué información tiene el archivo de errores?

La aplicación no termina, en el archivo de errores el cual nos la crea un archivo dentro de **SINAP.**



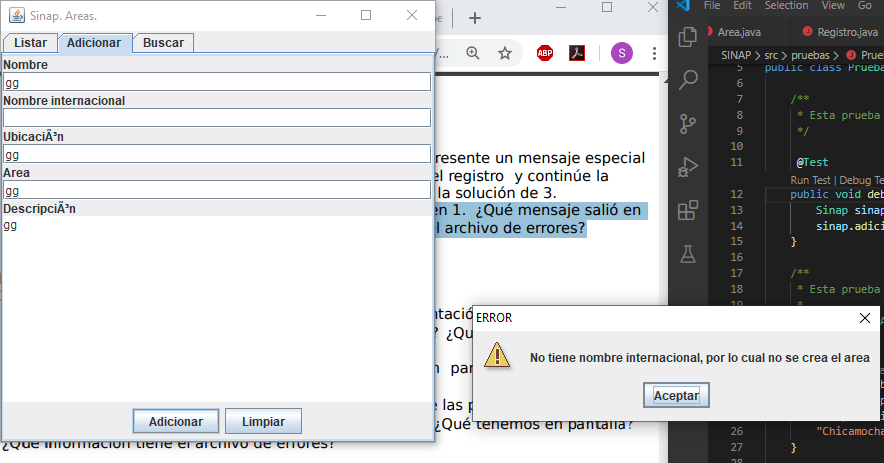
1. ¿Es adecuado que la aplicación continúe su ejecución después de sufrir un incidente como este? ¿de qué dependería continuar o parar?

Si, ya que solo debe de mandar el mensaje y no ejecutar la acción, pero no debería de parar. El parar depende de un error dentro del código, no debería ser por un error del usuario.

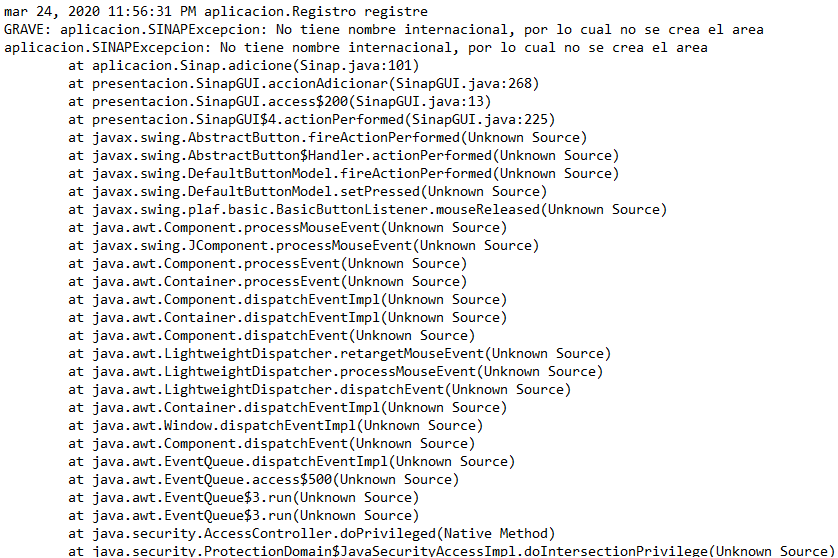
1. Analicen el punto adecuado para que EN ESTE CASO se presente un mensaje especial de alerta al usuario, se guarde la información del error en el registro y continúe la ejecución. Expliquen y construyan la solución. No eliminen la solución de 3.

Archivo SinapGUI.java

1. Ejecuten nuevamente la aplicación con el caso propuesto en 1. ¿Qué mensaje salió en pantalla? ¿La aplicación termina? ¿Qué información tiene el archivo de errores?



Al darle aceptar la aplicación no acaba.



**Consultando por patrones. ¡Ahora si funciona!**

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

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

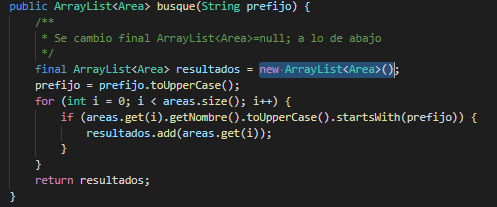
* El método accionBuscar() de la clase SinapGUI es el encargado de gestionar la búsqueda en la capa presentacion.
* El método busque(String prefijo) de la clase Sinap es el encargado de gestionar la búsqueda en la capa aplicación.

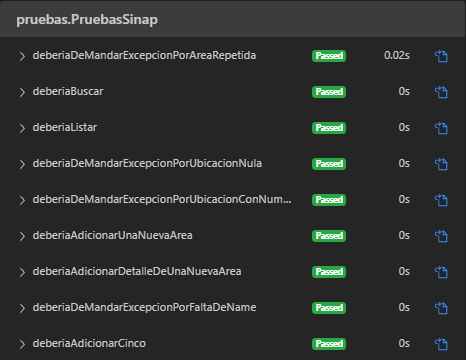
1. Realicen ingeniería reversa para la capa de aplicación para buscar. Capturen los resultados de las pruebas. Deben fallar.

Archivo .astah

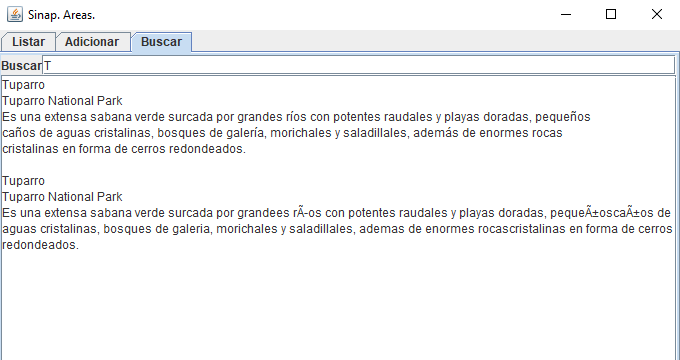
1. ¿Cuál es el error? Soluciónenlo. Capturen los resultados de las pruebas.

El error es que resultados no estaba definido, lo igualaban a null, se debía de crear un nuevo objeto de tipo ArrayList<Area>.

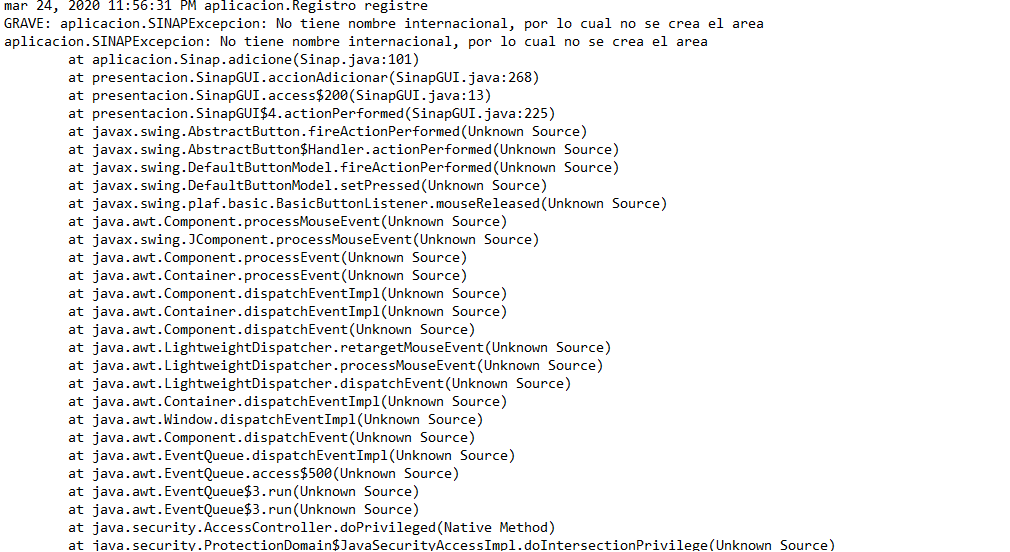




1. Ejecuten la aplicación nuevamente con el caso propuesto. ¿Qué tenemos en pantalla? ¿Qué información tiene el archivo de errores?



Archivo de errores:



No tiene ningún error de busque ya que este no tiene excepción alguna.

**RETROSPECTIVA**

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes?

Santiago Martínez – 48

Ricardo Amaya – 48

1. ¿Cuál es el estado actual del laboratorio? ¿Por qué?

Creemos que esta completo el laboratorio

1. Considerando las prácticas XP del laboratorio. ¿cuál fue la más útil?

La buena comunicación entre el equipo.

1. ¿Cuál consideran fue el mayor logro? ¿Por qué?

Poder tener la capacidad de adecuarnos a nuevos editores de java

1. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?

Poder descargar Junit y utilizarlo en Visual Studio Code

1. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

Repartir bien el trabajo, mejorar nuestras cargas de trabajo de una mejor manera.