**PROGRAMACIÓN ORIENTADA A OBJETOS**

**Excepciones**

**2025-1**

**Laboratorio 4/6**

**OBJETIVOS**

1. Perfeccionar el diseño y código de un proyecto considerando casos especiales y errores.
2. Construir clases de excepción encapsulando mensajes.
3. Manejar excepciones considerando los diferentes tipos.
4. Registrar la información de errores que debe conocer el equipo de desarrollo de una aplicación en producción.
5. Vivenciar las prácticas ***Designing*** [*Simplicity*](http://www.extremeprogramming.org/rules/simple.html)*.*

[Refactor](http://www.extremeprogramming.org/rules/refactor.html)  whenever and wherever possible.



**ENTREGA**

 Incluyan en un archivo .zip los archivos correspondientes al laboratorio. El nombre debe ser los dos apellidos de los miembros del equipo ordenados alfabéticamente.

 Deben publicar el avance al final de la sesión y la versión definitiva en la fecha indicada, en los espacios preparados para tal fin.

**Unidades**

# EN BLUEJ

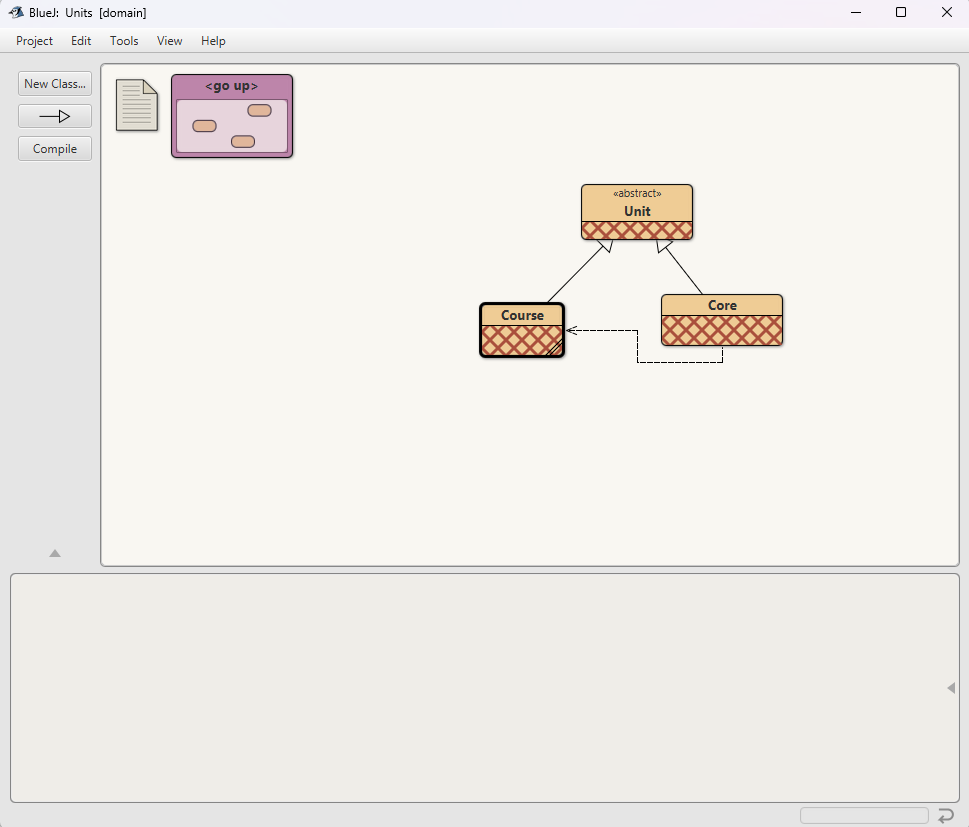
**PRACTICANDO MDD y BDD con EXCEPCIONES**

**[En lab04.doc, Plan15.asta y BlueJ units]**

En este punto vamos a aprender a diseñar, codificar y probar usando excepciones. Para esto se van a trabajar algunos métodos de la clase Core

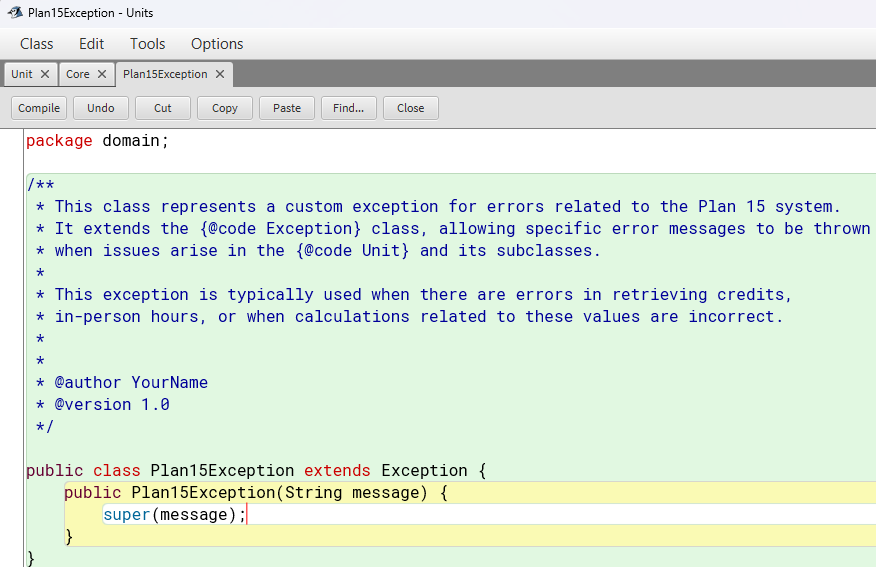
1. En su directorio descarguen los archivos contenidos en units.zip revisen el contenido y estudien el diseño estructural de la aplicación (únicamente la zona en azul).

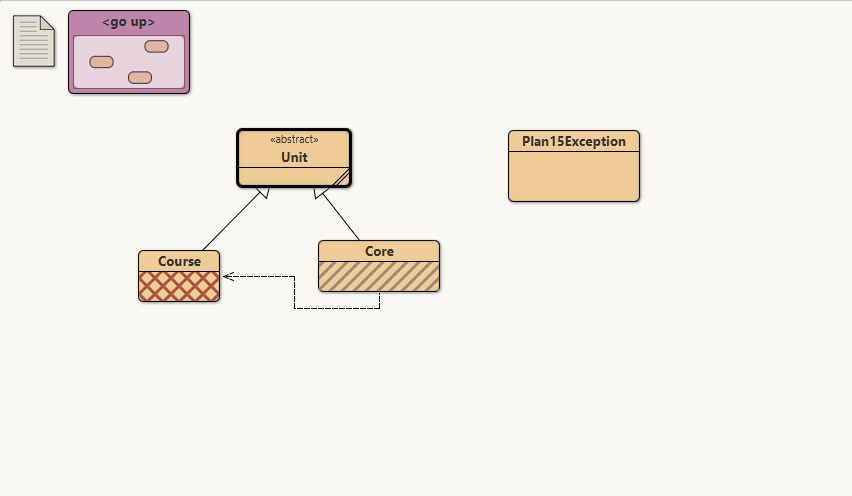
* Dentro del directorio de units.zip podemos encontrar package (BlueJ), doc y domain. Ya en sí, cuando abrimos el BlueJ podemos observar que se tiene una clase abstracta llamada Unit, y dos clases llamadas Core y Course que son heredadas de Unit, según lo que se puede ver en BlueJ.



1. Expliquen por qué el proyecto no compila. Realicen las adiciones necesarias para lograrlo.

* El proyecto no compila porque no se encuentra definida la acción de “Plan15Exception” que es lo que nos genera el error. Para esto simplemente creamos una clase llamada Plan15Exception y hacemos la acción necesaria para evitar esto.

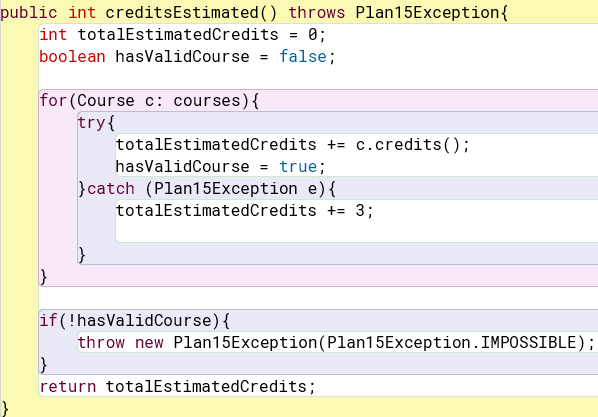




1. Dado el diseño y las pruebas documenten y codifiquen el método credits().



1. Dada la documentación y el diseño codifiquen y prueben el método creditsEstimated().



1. Documenten, diseñen, codifiquen y prueben el método inPersonEstimated().

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**Plan15**

# EN CONSOLA

El objetivo de esta aplicación es mantener un catálogo de las unidades de aprendizaje correspondientes al Plan15. En este plan se ofrecen diferentes tipos de disfraces: básicos y completos.

**Conociendo el proyecto Plan15 [En lab04.doc]**

**No olviden respetar los directorios bin docs src**

1. En su directorio descarguen los archivos contenidos en plan15.zip, revisen el contenido.

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

* Al momento de descargar el archivo de plan15.zip dentro de este se encuentran tres archivos los cuales son:

Tabla

El contenido generado por IA puede ser incorrecto.

* Estos deben estar organizados como src,bin y docs. Además de esto, se deben crear los paquetes de “presentation” y “test”.

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

* En este momento del laboratorio se tiene un total de solo un paquete nada más que se llama “domain”.
* Se tiene un total de 4 clases, Plan15, Unit, Core y course.
* En cuanto a la organización: La clase Plan15 tiene una relación de asociación con Unit, La clase Unit está relacionada con la clase Course, y la clase Core también está relacionada con la clase Course, La clase Core también tiene una relación de agregación con Course.
* Creemos que la **clase ejecutiva** es Plan15 ya que contiene métodos que sugieren la gestión y manipulación de datos del plan, como addSome(), consult(), addCourse(), addCore(), select(), data(), y search().

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

¿qué clases deben tener? ¿dónde están esas clases? ¿qué instrucciones debe dar para ejecutarlo?

* Se debe tener la estructura de src, bin, docs. Las 4 clases que mencionamos anteriormente se deberían encontrar en el paquete de domain que se encuentra en src, donde se encuentra todos los .java.

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

* Existen dos formas de ejecutar el programa, la primera es de forma manual desde BlueJ, entrando a Plan15GUI y se le da como si crearamos un objeto, con la pequeña diferencia que le damos es al void main(String[] args).
* La otra forma de poder ejecutar el programa es desde la consola, para esto se ejecuta el siguiente comando en la consola:

**java -cp bin presentation.Plan15GUI**

* Ofrece las funcionalidades de Listar, adicionar y buscar, funcionan las opciones de Listar y Limpiar.

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.

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

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

* Las unidades iniciales se crean e insertan directamente en el método addSome() de la clase Plan15.
* La clase que "pide" o, mejor dicho, la clase que contiene la lógica para agregar las unidades iniciales es la clase Plan15. Específicamente, el método addSome() dentro de Plan15 es el responsable de crear e insertar los cursos y el núcleo inicial.
* La clase que realmente "adiciona" o agrega las unidades a las estructuras de datos (ArrayList y TreeMap) es también la clase Plan15, pero a través de sus métodos addCourse() y addCore().
  + El método addCourse() crea instancias de la clase Course y las agrega tanto al ArrayList<Unit> units como al TreeMap<String, Course> courses.
  + Texto, Aplicación

    El contenido generado por IA puede ser incorrecto.El método addCore() crea instancias de la clase Core y las agrega al ArrayList<Unit> units. Adicionalmente este método agrega objetos de la clase Course a la clase Core.

**Adicionar y listar. Todo OK.**  [En lab04.doc, Plan15.astay \*.java]

(NO OLVIDEN BDD - MDD)

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

1. Adicionen un nuevo curso y un nuevo núcleo Curso

DOSW Desarrollo y Operaciones Software 4 y 4 *Core*

NFPE Nucleo de formación específica 100

Texto

El contenido generado por IA puede ser incorrecto.¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?

* Para hacer ingeniería reversa se hace lo siguiente:

Texto

El contenido generado por IA puede ser incorrecto.

* El comportamiento es adecuado como nos podemos dar cuenta ya que nos da DOSW, el número de créditos y además de esto de donde pertenece.

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

* El método responsable en la capa de presentación es el de actionAdd() de la clase Plan15GUI.
* En el caso del método de la capa de dominio encontramos dos, que son para courses y para core. Para courses es **addCourse()** y para core es **addCore().**

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

Capturen los resultados de las pruebas de unidad.

1. Revisen el código asociado a **listar** en la capa de presentación y la capa de dominio. ¿Qué método es responsable en la capa de presentación? ¿Qué método en la capa de dominio?
2. Realicen ingeniería reversa para la capa de dominio para **listar**.

Capturen los resultados de las pruebas de unidad.

1. Propongan y ejecuten una prueba de aceptación.

**Adicionar una unidad. Funcionalidad robusto**  [En lab04.doc, Plan15.astay \*.java]

(NO OLVIDEN BDD – MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un curso para hacerla más robusta.

**Para cada uno de los siguientes casos realice los pasos del 1 al 4.**

* 1. **¿Y si el nombre de la unidad no existe?**
  2. **¿Y si los valores enteros no son enteros?**
  3. **¿Y si el portage no está entre 0 y 100?**
  4. **Proponga una nueva condición**

1. Propongan una prueba de aceptación que genere el fallo.
2. Analicen el diseño realizado. Para hacer el software robusto: ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.
3. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.
4. Ejecuten nuevamente la aplicación con el caso de aceptación propuesto en 1. ¿Qué sucede ahora? Capture la pantalla.

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

[En Plan15.asta, Plan15.log, lab04.java y \*.java]

(NO OLVIDEN BDD - MDD)

1. Consulten una unidad completo que inicie con I. ¿Qué sucede? ¿Qué creen que pasó?

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

1. Exploren el método record de la clase Log ¿Qué servicio presta?
2. 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.
3. 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?
4. ¿Es adecuado que la aplicación continúe su ejecución después de sufrir un incidente como este? ¿de qué dependería continuar o parar?
5. Modifiquen la aplicación para garantizar que **SIEMPRE** que haya un error se maneje de forma adecuada. ¿Cuál fue la solución implementada?

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

[En Plan15.asta, Plan15.log, lab04.java y \*.java]

(NO OLVIDEN BDD - MDD)

1. Revisen el código asociado a **buscar** en la capa de presentación y la capa de dominio. ¿Qué método es responsable en la capa de presentación? ¿Qué método es responsable en la capa de dominio?
2. Realicen ingeniería reversa para la capa de dominio para **buscar**. Capturen los resultados de las pruebas. Deben fallar.
3. ¿Cuál es el error? Soluciónenlo. Capturen los resultados de las pruebas.
4. Ejecuten la aplicación nuevamente con el caso propuesto. ¿Qué tenemos en pantalla? ¿Qué información tiene el archivo de errores?
5. Refactorice la funcionalidad para que sea más amable con el usuario. ¿Cuál es la propuesta? ¿Cómo la implementa?

## RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre) 2. ¿Cuál es el estado actual del laboratorio? ¿Por qué?

1. Considerando las prácticas XP del laboratorio. ¿cuál fue la más útil? ¿por qué?
2. ¿Cuál consideran fue el mayor logro? ¿Por qué?
3. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?
4. ¿Qué hicieron bien como actividades? ¿Qué se comprometen a hacer para mejorar los resultados?
5. ¿Qué referencias usaron? ¿Cuál fue la más útil? Incluyan citas con estándares adecuados.