

LABORATORIO #4

JULIAN CAMILO LOPEZ BARRERO

JUAN SEBASTIAN PUENTES JULIO

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

POOB

BOGOTÁ, COLOMBIA

01 DE ABRIL DE 2025

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. Refactor 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).
2. Expliquen por qué el proyecto no compila. Realicen las adiciones necesarias para lograrlo. No compila porque le falta la clase Plan 15 Exception.

```
/**
 * Write a description of class Plan15Exception here.
 *
 * @author (Julian Lopez , Sebastian Puentes)
 * @version 1.0
 */
public class Plan15Exception extends Exception
{
    public static String CREDITS_UNKNOWN = "Se desconocen los creditos";
    public static String CREDITS_ERROR = "El numero de creditos son Menores o Iguales a 0";
    public static String PORTAGE_ERROR = "El porcentaje no esta entre 0 y 100";
    public static String IN_PERSON_UNKNOWN = "No hay persona disponible";
    public static String IN_PERSON_ERROR = "El numero De Personas Son Menores o Iguales a 0";
    public static String INVALID_NAME = "Nombre Invalido";
    public static String IMPOSSIBLE = "Imposible Calcular";
    public static String INVALID_CODE_LENGTH = "Longitud DelCodigo Invalido";
    public static String INVALID_CREDITS = "Creditos u Horas No Son Enteros";
    public static String INVALID_INPERSON = "Horas Inconsistentes";
    public static String INVALID_NAMES = "Nombres Iguales. Deben Ser Diferentes.";
    /**
     * Constructor for objects of class Plan15Exception
     */
    public Plan15Exception(String message)
    {
        super(message);
    }
}
```

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

```

/**
 * Calculates the total number of credits for all courses in this unit.
 * @return The total number of credits for this unit.
 * @throws Plan15Exception IMPOSSIBLE If the total number of credits is 0, |
 */
@Override
public int credits() throws Plan15Exception{
    int countCredits = 0;
    for(Course c : courses){
        countCredits += c.credits();
    }
    if(credits() == 0) {
        throw new Plan15Exception(Plan15Exception.IMPOSSIBLE);
    }
    return countCredits;
}

```

4. Dada la documentación y el diseño, codifiquen y prueben el método `creditsEstimated()`.

```

/**
 * Calculate the actual or estimated credits
 * If necessary (unknown or error), assumes the number of credits is 3
 * is equal to the in-person hours.
 * @return the estimated credits
 * @throws Plan15Exception IMPOSSIBLE, If there are no credits
 */
public int creditsEstimated() throws Plan15Exception{
    int estimatedCredits = 0;
    for(Course c : courses){
        try{
            estimatedCredits += c.credits();
        } catch (Plan15Exception e){
            estimatedCredits += 3;
        }
    }
    if(estimatedCredits == 0) throw new Plan15Exception(Plan15Exception.IMPOSSIBLE);
    return estimatedCredits;
}

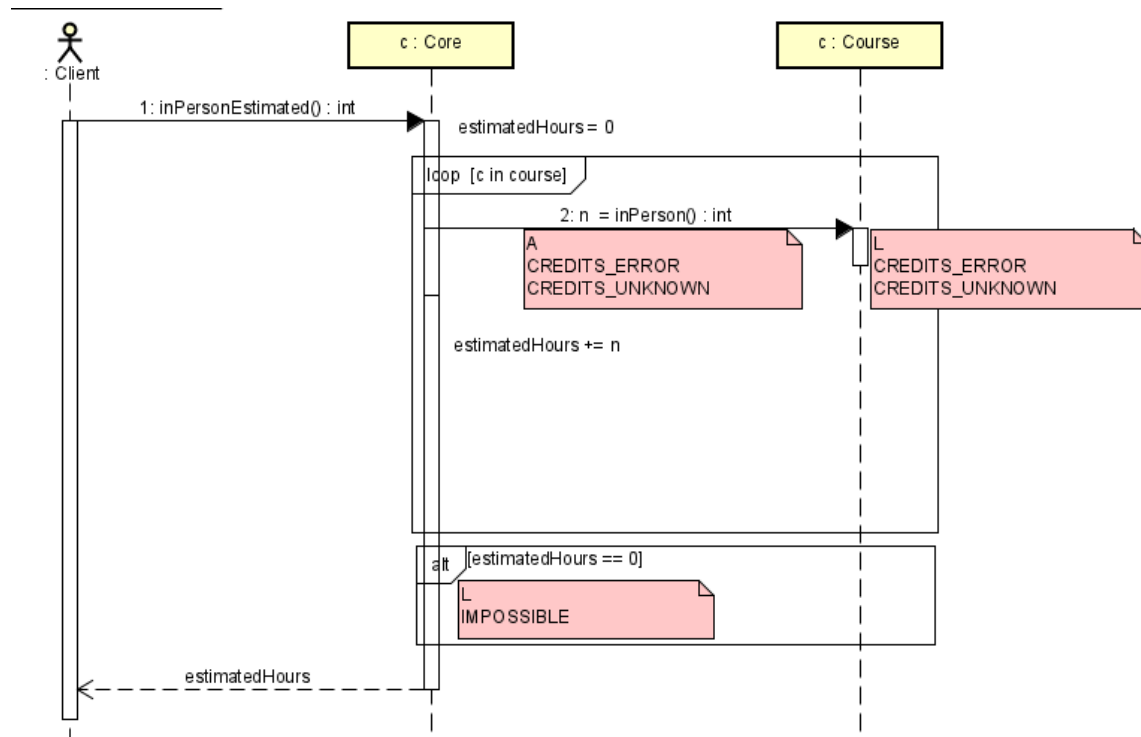
```

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

```

/**
 * Calculate the estimated in-person hours, considering courses that do not have credit issues.
 * If the hours of a course are not known, calculate the course in-person hours using the percentage suggested in the unit core.
 * If the hours of a course are incorrect, it is assumed that all the time is in person.
 * @return the estimated hours per person
 * @throws Plan15Exception IMPOSSIBLE, If there are no courses or all courses has credit issues
 */
public int inPersonEstimated() throws Plan15Exception {
    int estimatedHours = 0;
    for (Course c : courses) {
        try {
            estimatedHours += c.inPerson();
        } catch (Plan15Exception e) {
            try {
                estimatedHours += c.independent();
            } catch (Plan15Exception ex) {
                estimatedHours += 3*3;
            }
        }
    }
    if (estimatedHours == 0) {
        throw new Plan15Exception(Plan15Exception.IMPOSSIBLE);
    }
    return estimatedHours;
}

```



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?

Tiene 3 archivos.java

¿Cómo están organizados?

Están desordenados en un simple directorio.

¿Cómo deberían estar organizados?

Deberían estar organizados de tal forma que su subdirectorio los tuviera como presentation en un src.

2. Estudien el diseño del programa: diagramas de paquetes y de clases.

¿cuántos paquetes tenemos?

Tenemos el paquete de presentación y el paquete de dominio

¿Cuántas clases tiene el sistema?

Tenemos 5 clases en el sistema Unit, Core, Course, Plan15 y Plan15GUI.

¿cómo están organizadas?

En presentation se encuentra Plan15GUI.

En dominio se tiene Unit, Core, Course y Plan15.

¿Cuál es la clase ejecutiva?

La clase ejecutiva es Plan15GUI porque contiene el main.

3. Prepare los directorios necesarios para ejecutar el proyecto.

¿Qué estructura debe tener?

La estructura que debe tener es la carpeta raíz

src

 dominio

 archivos.java

 presentation

 archivos.java de presentacion

bin

 archivos.class

docs

 archivos.html

Debe tener los directorios estructurados con bin , src y docs

¿qué clases deben tener?

Debe tener las clases del paquete dominio.













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

Las clases están en bin

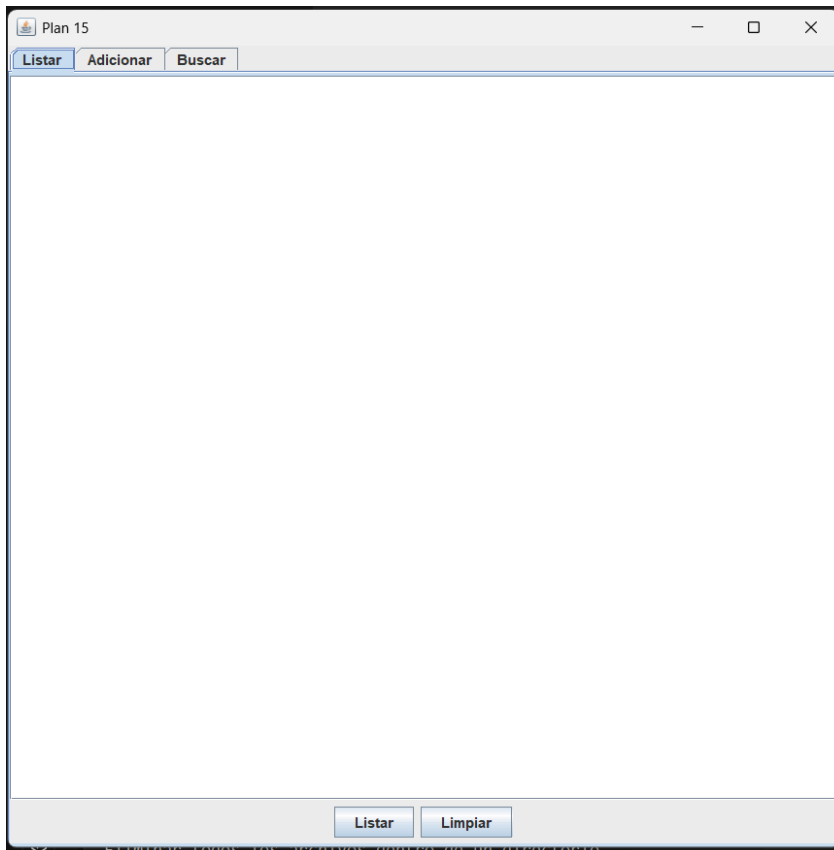
Al compilar con el comando

`javac -d bin -sourcepath src src/domain/*.java src/presentation/*.java src/*.java`

se nos crean en la carpeta de bin las respectivas clases que se necesiten.

 Core.class		2/04/2025 6:07 p.m.	Archivo CLASS	3 KB
 Course.class		2/04/2025 6:07 p.m.	Archivo CLASS	2 KB
 Log.class		2/04/2025 6:07 p.m.	Archivo CLASS	2 KB
 Plan15.class		2/04/2025 6:07 p.m.	Archivo CLASS	5 KB
 Plan15Exception.class		2/04/2025 6:07 p.m.	Archivo CLASS	1 KB
 Unit.class		2/04/2025 6:07 p.m.	Archivo CLASS	1 KB

4. Ejecute el proyecto, ¿qué funcionalidades ofrece? ¿Cuáles funcionan?



Ofrece dos funcionalidades , la de listar y la de limpiar y en la parte superior nos ofrece Adicionar, Listar y Buscar.

Funcionan listar y adicionar.

5. Revisen el código y la documentación del proyecto.

¿De dónde salen las unidades iniciales?

Las unidades salen del Metodo AddSome().

¿Qué clase pide que se adicionen?

La clase que pide que los adiciones es Plan15GUI.

¿Qué clase los adiciona?

La clase que los adiciona es Plan15 con el metodo addCourse o addCore.

Adicionar y listar. Todo OK. [En lab04.doc, Plan15.asta y *.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 Núcleo de formación específica 100 DOSW

```
private void addSome(){
    String [][] courses = {{ "PRI1", "Proyecto Integrador", "9", "3"},
                           { "DDYA", "Diseño de Datos y Algoritmos", "4", "4"},
                           { "MPIN", "Matematicas para Informatica", "3", "4"},
                           { "DOSW", "Desarrollo y Operaciones Software", "4", "4"} };
    for (String [] c: courses){
        addCourse(c[0],c[1],c[2],c[3]);
    }
    String [][] Core = {{ "FCC", "Nucleo formacion por comun por campo", "50", "PRI1\nDDYA\nMPIN"},
                       { "NFPE", "Nucleo de formacion especifica", "100", "DOSW"} };
    for (String [] c: Core){
        addCore(c[0],c[1],c[2],c[3]);
    }
}
```

¿Qué ocurre?

Se listan los cursos y los núcleos de estos.

¿Cómo lo comprueban? Capturen la pantalla.

¿Es adecuado este comportamiento?

6 unidades

```
>PRI1: Proyecto Integrador. Creditos:9[3+24]
>DDYA: Diseño de Datos y Algoritmos. Creditos:4[4+8]
>MPIN: Matematicas para Informatica. Creditos:3[4+5]
>DOSW: Desarrollo y Operaciones Software. Creditos:4[4+8]
>FCC: Nucleo formacion por comun por campo. [50%]
        PRI1: Proyecto Integrador. Creditos:9[3+24]
        DDYA: Diseño de Datos y Algoritmos. Creditos:4[4+8]
        MPIN: Matematicas para Informatica. Creditos:3[4+5]
>NFPE: Nucleo de formacion especifica. [100%]
        DOSW: Desarrollo y Operaciones Software. Creditos:4[4+8]
```

Es adecuado ya que el núcleo recibió como curso DOSW. //Revisar

2. Revisen el código asociado a **adicionar** en la capa de presentación y la capa de dominio. Capturen los resultados de las pruebas de unidad. ¿Qué método es responsable en la capa de presentación?

El método responsable de la capa de presentación es `addAction()`.

¿Qué método en la capa de dominio?

El método en la capa de dominio es `addCourse()` o `addCore()`.

3. Realicen ingeniería reversa para la capa de dominio para adicionar.

Capturen los resultados de las pruebas de unidad.

```
-- shouldAddSecondCoreAndCourseSuccessfully [OK]
-- shouldAddCoreAndCourseSuccessfully [OK]
```

4. 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?

El método responsable es `actionList()`

¿Qué método en la capa de dominio?

El método responsable en la capa de dominio es `data()`.

5. Realicen ingeniería reversa para la capa de dominio para listar.

Capturen los resultados de las pruebas de unidad.

```

-- shouldAddCourseSuccessfully [OK]
-- shouldDoToString [OK]
JUnit Platform Suite [OK]

run finished after 106 ms
  6 containers found      ]
  0 containers skipped    ]
  6 containers started    ]

```

6. Propongan y ejecuten una prueba de aceptación.

```

-- shouldDoToStringAddingTwoCourses [OK]
-- shouldSearch [OK]

```

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.

- ¿Y si el nombre de la unidad no existe?
- ¿Y si los valores enteros no son enteros?
- ¿Y si el portage no está entre 0 y 100?
- Proponga una nueva condición
 - Propongan una prueba de aceptación que genere el fallo.
 - 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.
 - Construya la solución propuesta. Capture los resultados de las pruebas de unidad.
 - Ejecuten nuevamente la aplicación con el caso de aceptación propuesto en 1. ¿Qué sucede ahora? Capture la pantalla.

a. ¿Y si el nombre de la unidad no existe?

- Prueba:

```

@Test
public void shouldFailWhenIsEmptyName() {
    try {
        Plan15 plan = new Plan15();
        try {
            plan.addCourse("CCC1", null, "3", "2");
        } catch (Plan15Exception e) {
            assertEquals(Plan15Exception.INVALID_NAME, e.getMessage());
        }
    } catch (Plan15Exception f) {
    }
}

```

Genera error debido a que aún no se ha implementado la lógica para que la prueba sea correcta.

2. El método que debería lanzar la excepción es el que se encarga de añadir los cursos el cual es, `addCourse()`. Así mismo los métodos que llamen a este deberían propagar la excepción hasta que sea capturada en el método `actionAdd()`.

3. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.

```
/**
 * Add a new course
 */
public void addCourse(String code, String name, String credits, String inPerson) throws Plan15Exception{
    if (name == null || name.trim().isEmpty()) {
        throw new Plan15Exception(Plan15Exception.INVALID_NAME);
    }

    Course nc=new Course(code,name,Integer.parseInt(credits),Integer.parseInt(inPerson));
    units.add(nc);
    courses.put(code.toUpperCase(),nc);
}
```

- ✓ Test.CoreTest.shouldThrowExceptionIfCoreHasNoCourse()
- ✓ Test.CoreTest.shouldCalculateTheCreditsOfACoreCostume()
- ✓ Test.CoreTest.shouldThrowExceptionIfCreditsIsNotKnown()
- ✓ Test.CoreTest.shouldThrowExceptionIfThereIsErrorInCredits()
- ✓ Test.Plan15Test.shouldDoToStringAddingTwoCourses()
- ✓ Test.Plan15Test.shouldSearch()
- ✓ Test.Plan15Test.shouldAddSecondCoreAndCourseSuccessfully()
- ✓ Test.Plan15Test.shouldAddCoreAndCourseSuccessfully()
- ✓ Test.Plan15Test.shouldDoToString()
- ✓ Test.Plan15Test.shouldFailWhenIsEmptyName()

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

Nombre Invalido

The screenshot shows a Java Swing window titled "Plan 15". At the top, there are three buttons: "Listar", "Adicionar", and "Buscar". Below these are several input fields with labels: "Sigla" (containing "CC1"), "Nombre" (empty), "Creditos o porcentaje" (containing "3"), "Horas presenciales (solo para cursos)" (containing "2"), and "Cursos (solo para nucleos)" (empty). At the bottom of the window, there are two buttons: "Adicionar" and "Limpiar".

En la consola se despliega un mensaje donde nos afirma que no podemos hacer la adición de este curso con un nombre vacío.

b. ¿Y si los valores enteros no son enteros?

1. Propongan una prueba de aceptación que genere el fallo.

```
@Test
public void shouldFailWhenNonIntegerCredits() {
    try {
        Plan15 plan = new Plan15();
        try {
            plan.addCourse("C1", "COURSE", "abc", "2");
        } catch (Plan15Exception e) {
            assertEquals(Plan15Exception.INVALID_CREDITS, e.getMessage());
        }
    } catch (Plan15Exception f) {}
}
```

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.

El método que debería lanzar la excepción es el que se encarga de añadir los cursos el cual es, `addCourse()`. Así mismo los métodos que llamen a este deberían propagar la excepción hasta que sea capturada en el método `actionAdd()`.

3. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.

```
/**
 * Add a new course
 */
public void addCourse(String code, String name, String credits, String inPerson) throws Plan15Exception{
    if (name == null || name.trim().isEmpty()) {
        throw new Plan15Exception(Plan15Exception.INVALID_NAME);
    }
    int parsedCredits;
    int parsedInPerson;
    try{
        parsedCredits = Integer.parseInt(credits);
        parsedInPerson = Integer.parseInt(inPerson);
        if(parsedCredits <= 0) {
            throw new Plan15Exception(Plan15Exception.CREDITS_ERROR);
        }
        if(parsedInPerson <= 0 || parsedInPerson > 3 * parsedCredits) {
            throw new Plan15Exception(Plan15Exception.INVALID_INPERSON);
        }
        Course nc=new Course(code,name,Integer.parseInt(credits),Integer.parseInt(inPerson));
        units.add(nc);
        courses.put(code.toUpperCase(),nc);
    }catch(NumberFormatException e){
        throw new Plan15Exception(Plan15Exception.INVALID_CREDITS);
    }
}
```

```

✓ Test.CoreTest.shouldThrowExceptionIfCoreHasNoCourse()
✓ Test.CoreTest.shouldCalculateTheCreditsOfACoreCourse()
✓ Test.CoreTest.shouldThrowExceptionIfCreditsIsNotKnown()
✓ Test.CoreTest.shouldThrowExceptionIfThereIsErrorInCredits()
✓ Test.Plan15Test.shouldDoToStringAddingTwoCourses()
✓ Test.Plan15Test.shouldSearch()
✓ Test.Plan15Test.shouldAddSecondCoreAndCourseSuccessfully()
✓ Test.Plan15Test.shouldAddCoreAndCourseSuccessfully()
✓ Test.Plan15Test.shouldDoToString()
✓ Test.Plan15Test.shouldFailWhenNonIntegerCredits()
✓ Test.Plan15Test.shouldFailWhenIsEmptyName()

```

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

Plan 15

Listar Adicionar Buscar

Sigla
CCC

Nombre
CCC

Credits o porcentaje
CC

Horas presenciales (solo para cursos)
1

Cursos (solo para nucleos)

Creditos u Horas No Son Enteros

En la consola se despliega un mensaje donde nos afirma que no podemos hacer la adición de este curso con un tipo de créditos u horas invalido.

c. ¿Y si el portage no está entre 0 y 100?

1. Propongan una prueba de aceptación que genere el fallo.

```

@Test
public void shouldFailWhenPortageIsNotBetween0And100() {
    try {
        Plan15 plan = new Plan15();
        try {
            plan.addCore("CCC1", "COURSE", "101", "PRYE");
        } catch (Plan15Exception e) {
            assertEquals(Plan15Exception.PORTAGE_ERROR, e.getMessage());
        }
    } catch (Plan15Exception f) {
    }
}

```

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.

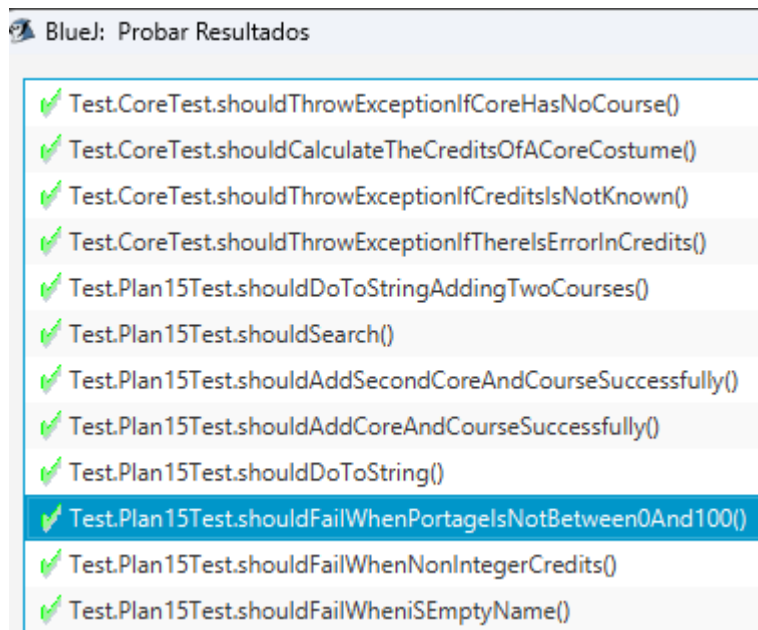
AddCore() es el método que debe lanzarla ya que es la encargada de agregar los núcleos así mismo, los métodos que llamen este deben propagar la excepción hasta que addAction() atiende el mismo.

3. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.

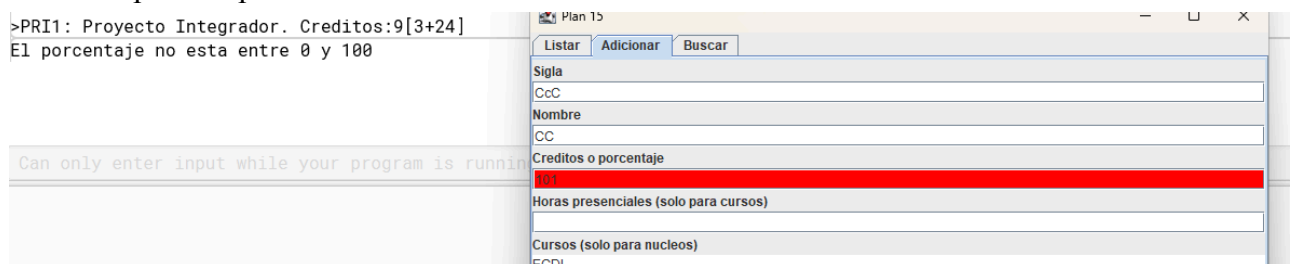
```

/**
 * Add a new core
 */
public void addCore(String code, String name, String percentage, String theCourses) throws Plan15Exception{
    int parsedPercentage;
    try{
        parsedPercentage = Integer.parseInt(percentage);
        if(parsedPercentage < 0 || parsedPercentage > 100){
            throw new Plan15Exception(Plan15Exception.PORTAGE_ERROR);
        }
        Core c = new Core(code,name,Integer.parseInt(percentage));
        String [] aCourses= theCourses.split("\n");
        for (String b : aCourses){
            c.addCourse(courses.get(b.toUpperCase()));
        }
        units.add(c);
    }catch(NumberFormatException e){
        throw new Plan15Exception(Plan15Exception.PORTAGE_ERROR);
    }
}

```



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



En la consola se despliega un mensaje donde nos afirma que no podemos hacer la adición de este núcleo con un porcentaje que no se encuentre entre 0 y 100.

d. Proponga una nueva condición

1. Propongan una prueba de aceptación que genere el fallo.

Decidimos generar una excepción si se decide adicionar un curso con la sigla y nombres iguales para evitar diferentes confusiones.

2. Analicen el diseño realizado. Para hacer el software robusto: ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarse? ¿Qué método debería atenderla? Explique claramente.

AddCore() y addCourse() son los métodos que deben lanzar la excepción ya que es la encargada de agregar los núcleos así mismo, los cursos. Los métodos que llamen estos deben propagar la excepción hasta que actionsAdd() atienda el mismo.

3. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.

```

/**
 * Add a new course
 */
public void addCourse(String code, String name, String credits, String inPerson) throws Plan15Exception{
    if (name == null || name.trim().isEmpty()) {
        throw new Plan15Exception(Plan15Exception.INVALID_NAME);
    }
    int parsedCredits;
    int parsedInPerson;
    try{
        parsedCredits = Integer.parseInt(credits);
        parsedInPerson = Integer.parseInt(inPerson);
        if(parsedCredits <= 0) {
            throw new Plan15Exception(Plan15Exception.CREDITS_ERROR);
        }
        if(parsedInPerson <= 0 || parsedInPerson > 3 * parsedCredits) {
            throw new Plan15Exception(Plan15Exception.INVALID_INPERSON);
        }
        if(code.equals(name)){
            throw new Plan15Exception(Plan15Exception.INVALID_NAMES);
        }
        Course nc=new Course(code,name,Integer.parseInt(credits),Integer.parseInt(inPerson));
        units.add(nc);
        courses.put(code.toUpperCase(),nc);
    }catch(NumberFormatException e){
        throw new Plan15Exception(Plan15Exception.INVALID_CREDITS);
    }
}

```

```

/**
 * Add a new core
 */
public void addCore(String code, String name, String percentage, String theCourses) throws Plan15Exception{
    int parsedPercentage;
    try{
        parsedPercentage = Integer.parseInt(percentage);
        if(parsedPercentage < 0 || parsedPercentage > 100){
            throw new Plan15Exception(Plan15Exception.PORTAGE_ERROR);
        }
        if(code.equals(name)){
            throw new Plan15Exception(Plan15Exception.INVALID_NAMES);
        }
        Core c = new Core(code,name,Integer.parseInt(percentage));
        String [] aCourses= theCourses.split("\n");
        for (String b : aCourses){
            c.addCourse(courses.get(b.toUpperCase()));
        }
        units.add(c);
    }catch(NumberFormatException e){
        throw new Plan15Exception(Plan15Exception.PORTAGE_ERROR);
    }
}

```

✓ Test.CoreTest.shouldCalculateTheCreditsOfACoreCostume()
✓ Test.CoreTest.shouldThrowExceptionIfCreditsIsNotKnown()
✓ Test.CoreTest.shouldThrowExceptionIfThereIsErrorInCredits()
✓ Test.Plan15Test.shouldDoToStringAddingTwoCourses()
✓ Test.Plan15Test.shouldSearch()
✓ Test.Plan15Test.shouldAddSecondCoreAndCourseSuccessfully()
✓ Test.Plan15Test.shouldAddCoreAndCourseSuccessfully()
✓ Test.Plan15Test.shouldDoToString()
✓ Test.Plan15Test.shouldFailWhenPortagelsNotBetween0And100()
✓ Test.Plan15Test.shouldFailWhenNonIntegerCredits()
✓ Test.Plan15Test.shouldFailWhenIfNameAndInitialsAreTheSame()
✓ Test.Plan15Test.shouldFailWhenIsEmptyName()

Ejecuciones: 13

✗ Errores:0

✗ Fallos:0

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

Plan 15

Listar Adicionar Buscar

Sigla

CCC1

Nombre

CCC1

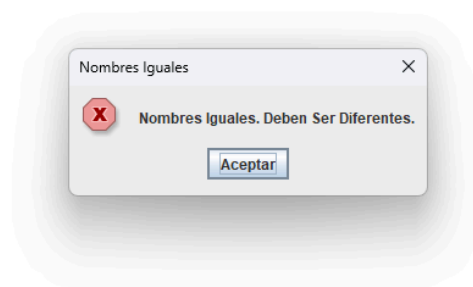
Creditos o porcentaje

3

Horas presenciales (solo para cursos)

2

Cursos (solo para nucleos)



Adicionar Limpiar

No se agrega debido a que tiene nombres iguales.

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?

Lo deberían saber los desarrolladores y el equipo de mantenimiento mas no el usuario que va a usar la aplicación porque no aporta nada al usuario , el no va a entender de que tratan estos errores.

Nos da un error que dice:

```
java.lang.IndexOutOfBoundsException:  
Index 6 out of bounds for length 6 (in jdk.internal.util.Preconditions)
```

Lo que sucede porque se intentó acceder a un índice fuera del rango del arraylist.

2. Exploren el método récord de la clase Log ¿Qué servicio presta?

Captura el error con:

```
logger.log(Level.SEVERE,e.toString(),e);
```

y luego escribe la información en un archivo de registro para análisis posterior y evita que los errores se pierdan al imprimirlos solo en consola.

3. 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.

El mejor lugar donde se podría alertar al usuario sería cuando vaya a consultar o a buscar un debido a que son los encargados de buscar Units por nombre o prefijo, si ocurre un error al buscar seria bueno mostrar ese mensaje especial.

4. Ejecuten nuevamente la aplicación con el caso propuesto en 1.

¿Qué mensaje salió en pantalla?



¿La aplicación termina?

La aplicación no termina.

¿Qué información tiene el archivo de errores?


```

abr 02, 2025 7:54:36 P. M. domain.Log record
GRAVE: java.lang.IndexOutOfBoundsException: Index 6 out of bounds for length 6
java.lang.IndexOutOfBoundsException: Index 6 out of bounds for length 6
    at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:100)
    at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:106)
    at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:302)
    at java.base/java.util.Objects.checkIndex(Objects.java:385)
    at java.base/java.util.ArrayList.get(ArrayList.java:427)
    at domain.Plan15.select(Plan15.java:87)
    at domain.Plan15.search(Plan15.java:121)
    at domain.__SHELL33.run(__SHELL33.java:10)
    at java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:103)
    at java.base/java.lang.reflect.Method.invoke(Method.java:580)
    at bluej.runtime.ExecServer$3.lambda$run$0(ExecServer.java:873)
    at bluej.runtime.ExecServer.runOnTargetThread(ExecServer.java:996)
    at bluej.runtime.ExecServer$3.run(ExecServer.java:870)

```

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

Depende de continuar en base a el grado de el error si es un error donde la consulta no encuentra resultados la aplicación debe continuar para que así el usuario pueda hacer otra consulta debe parar en dado caso que el error impide que la aplicación funcione como lo hace normalmente como por ejemplo si hubiera una inconsistencia grave en los datos.

6. Modifiquen la aplicación para garantizar que SIEMPRE que haya un error se maneje de forma adecuada. ¿Cuál fue la solución implementada?

Se implementó un Log.record() para guardar dichas excepciones. y así mismo los diferentes try- catch que atrapan los mismos para lanzar el mensaje y continuar con la ejecución del programa.

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?

El metodo responsable es actionSearch().

¿Qué método es responsable en la capa de dominio?

El metodo responsable en la capa de dominio es search(String prefix)

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

```

- Plan15Test [OK]
+-- shouldSearch [X] Index 6 out of bounds for length 6

```

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

El error es la iteración del for donde había un <= y esto no puede ser asi , lo cambiamos por un menor estricto

```

public ArrayList<Unit> select(String prefix){
    ArrayList <Unit> answers=new ArrayList<Unit>();
    prefix=prefix.toUpperCase();
    for(int i=0;i<units.size();i++){
        if(units.get(i).code().toUpperCase().startsWith(prefix)){
            answers.add(units.get(i));
        }
    }
    return answers;
}

```

```

- Plan15Test [OK]
+-- shouldSearch [OK]

```

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?

Se decide usar JOptionPane para mostrar al cliente las excepciones de manera que se entienda con facilidad y pueda ajustar sus entradas de manera correcta. Se señala con color rojo donde ocurre el error y así mismo se envía un mensaje del error.

Estas se implementaron en:

```

private void actionAdd(){
    try {
        if (basics.getText().trim().equals("")){
            plan.addCourse(code.getText(),name.getText(),credits.getText(),inPerson.getText());
        }else{
            plan.addCore(code.getText(),name.getText(),credits.getText(),basics.getText());
        }
    } catch (Plan15Exception e) {
        Log.record(e);
        if (e.getMessage().equals(Plan15Exception.INVALID_CREDITS)) {
            credits.setBackground(Color.RED);
            JOptionPane.showMessageDialog(this, e.getMessage(), "Error De Tipo", JOptionPane.ERROR_MESSAGE);
        }
        if (e.getMessage().equals(Plan15Exception.CREDITS_ERROR)) {
            credits.setBackground(Color.RED);
            JOptionPane.showMessageDialog(this, e.getMessage(), "Error De Tipo", JOptionPane.ERROR_MESSAGE);
        }
        if (e.getMessage().equals(Plan15Exception.INVALID_NAME)) {
            name.setBackground(Color.RED);
            JOptionPane.showMessageDialog(this, e.getMessage(), "Error De Nombre Nulo", JOptionPane.ERROR_MESSAGE);
        }
        if (e.getMessage().equals(Plan15Exception.PORTAGE_ERROR)) {
            credits.setBackground(Color.RED);
            JOptionPane.showMessageDialog(this, e.getMessage(), "Fuera De Rango. Porcentage entre 0 y 100.", JOptionPane.ERROR_MESSAGE);
        }
        if (e.getMessage().equals(Plan15Exception.INVALID_NAMES)) {
            name.setBackground(Color.RED);
            code.setBackground(Color.RED);
            JOptionPane.showMessageDialog(this, e.getMessage(), "Nombres Iguales", JOptionPane.ERROR_MESSAGE);
        }
    }
}

```

EJEMPLO DE VISUALIZACIÓN:

Plan 15

Listar Adicionar Buscar

Sigla

CCC1

Nombre

CCC1

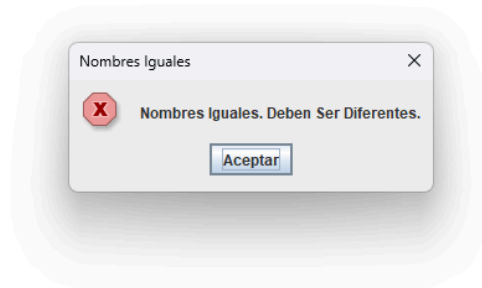
Creditos o porcentaje

3

Horas presenciales (solo para cursos)

2

Cursos (solo para nucleos)



Adicionar Limpiar

RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre)

Alrededor de 20 horas por cada uno .

2. ¿Cuál es el estado actual del laboratorio?

¿Por qué? (Para cada método incluya su estado)

El estado actual del laboratorio está terminado. todos los métodos funcionalidades requeridas fueron implementados y probados correctamente , por lo cual cumple los objetivos del laboratorio.

3. Considerando las prácticas XP del laboratorio de hoy ¿por qué consideran que son importantes?

La práctica más útil fue el desarrollo guiado por pruebas (TDD - Test-Driven Development), porque durante el desarrollo del laboratorio se realizaron diferentes test para probar la funcionalidad del sistema.

4. ¿Cuál considera fue su mayor logro? ¿Por qué? ¿Cuál consideran que fue su mayor problema? ¿Qué hicieron para resolverlo?

El Mayor problema fue realizar las pruebas y manejar el Log para las diferentes excepciones. Para resolverlo se buscaron diferentes videos de apoyo.

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

Como equipo, nos dividimos de manera equitativa el trabajo para la realización de este de manera más eficiente además hicimos uso de pair programming y así corregir los diferentes errores presentados a medida del desarrollo.

6. ¿Qué referencias usaron? ¿Cuál fue la más útil?

https://www.w3schools.com/java/ref_string_matches.asp

<https://www.makigas.es/series/records-en-java/records-de-java-que-son-y-como-usarlos>