# Disciplina de Prueba

El objetivo de esta disciplina es realizar una evaluación objetiva del producto. Esto incluye:

* Encontrar y corregir errores
* Validar que el producto opere tal como fue diseñado
* Verificar que los requerimientos hayan sido implementados

De acuerdo con RUP se distinguen 4 tipos de prueba: unitaria, de integración, de sistema y de aceptación.

Las pruebas que realizan los estudiantes en el proceso RUP-GDIS son: unitarias y de integración.

Las pruebas de sistema y de aceptación se realizan en el contexto académico con la entrega que hacen los estudiantes a los miembros del grupo docente

Esta disciplina esta conformada por un modelo y dos artefactos:

* Modelo de Prueba
  + Especificación de casos de prueba
  + Diagrama de paquete

## Modelo de Prueba

Este modelo describe las pruebas. Debe indicarse el identificador de clase, el identificador del caso de prueba, su descripción, y un reporte del resultado de la prueba.

### Especificación de casos de prueba

Este artefacto describe cuáles son los datos con los que se ejecuta el caso de prueba. Para esto se debe llenar la plantilla descrita más abajo. Al llenar la plantilla se deben considerar las pre y post condiciones especificadas en los UC y utilizar pruebas caja blanca y caja negra.

La plantilla utilizada para especificar casos de prueba es:

**Id Prueba**: # Entero de manera ascendente comenzando por 1

**Tipo de Prueba**: se indica el tipo de prueba a ser considerada. Puede ser unitarias o de integración. Las pruebas unitarias indican la unidad mínima a ser probada, por ejemplo, el método de una clase. Las pruebas de integración sirven para probar varios programas o métodos de forma unitaria.

**Descripción**: breve descripción del propósito de la prueba. Esta descripción va relacionado con el método para el cual se define la prueba.

**Clase**: nombre de la clase a ser probada

**Método**: signatura del método a ser probado. Debe especificarse los parámetros y sus tipos en formato Java, dado que vamos a implementar en java y coincidir con lo colocado en el diagrama de clases de diseño **Tipo de retorno**: tipo de retorno del método, si no tiene retorno se pone void. Debe ir en la misma línea que el método, pero alineado a la derecha.

**Pre-condición**: predicados que deben satisfacerse antes de realizar la prueba, son similares a las pre-condiciones de los UC, pero definidos para el método.

**Post-condición**: predicados que deben satisfacerse cuando se realiza la prueba, son similares a las post-condiciones de los UC, pero definidos para el método. La post-condición que se coloque aquí debe imprimirse a consola para validar que se cumpla.

**Casos de prueba**: conjunto de datos con los que se va a ejecutar el software (T). Para esto se usa el teorema fundamental de la prueba. Hay que definir quien es D y de ahí sacar el subconjunto T (la prueba ideal). T⊆D. Para llenar esto normalmente se utiliza la técnica caja negra y se busca que T consiga errores. Además, se deben definir los datos iniciales y luego datos que busquen el error. Por ejemplo, si la CI es única, deberían colocarse datos de prueba donde se repita la CI para ver el error.

**Valor esperado**: valor(es) que se debe(n) obtener después de la ejecución. Representa OUT(d, F(d)) o OK(d) del teorema fundamental de la prueba. Aquí para cada elemento t ∈ T: indicar signatura del método con los valores de t indicados en “valor esperado” y su respectivo resultado considerando la post-condición.

**Resultado**: valor(es) que se obtienen como salida una vez realizada la prueba (experimentación). Si aún no se ha implementado el código, hay que incorporar un texto que indique que se imprima a consola el resultado.

Para llenar esta planilla debemos tener en cuenta lo siguiente:

1. Se debe conocer el teorema fundamental de la prueba y las técnicas caja negra y caja blanca. Hay más detalles de esto en el documento “\_4\_Disciplina de Prueba Teoría.docx”
2. Tipo de prueba: en nuestro caso siempre son unitarias
3. Descripción: no usar la palabra usuario al describir
4. Los nombres de las clases, métodos y atributos deben corresponder con lo especificado en el diagrama de clases de diseño. Lo mismo con el tipo de retorno

Si el parámetro o atributo es un carácter se coloca entre apóstrofe para representar que es un carácter y no un string

1. Tipo de retorno: siempre va en la misma línea de método alineado a la derecha
2. Casos de prueba: Para expresar los casos de prueba hay que definir cuál es la prueba T y los elementos t ∈ T en base a los parámetros del método a probar y la pre-condición. La idea es definir el conjunto de datos con los que se va a ejecutar el método para el cual están especificando la prueba.
3. Valor esperado: se expresa en base a las post-condiciones definidas en las especificaciones de los UC del sistema. El propósito es indicar los valor(es) que se debe(n) obtener al ejecutar la aplicación, es decir son los resultados que se esperan obtener.
4. Resultado: como no hemos implementado el código, en esta parte se coloca lo que debería ocurrir cuando se ejecute la prueba. Por ejemplo:
   * Imprimir por consola los resultados del tamaño de conj\_obras para cada uno de los elementos de T cuando se ejecute la aplicación con este caso de prueba T.
   * Imprimir los resultados y después registrarlos cuando se ejecute la aplicación con este caso de prueba T
5. Otra cosa a tomar en cuenta es ver las pruebas de forma practica y global, imaginar que las vamos realizando, así por ejemplo, si vamos a realizar la prueba de un método que modifica un objeto, primer se debería hacer una prueba del método que cree el objeto y ahí agregar datos que luego se modifiquen en la siguiente prueba. Las pruebas deben llevar concordancia con lo que va quedando en las estructuras de datos.

Un ejemplo de esto sería (ver diagrama de clases puesto en diagrama de paquete abajo para los detalles de clase, método y tipo de retorno):

**Id Prueba**: 1

**Tipo de Prueba**: Unitaria

**Descripción**: Registrar obra. Se registra una obra en patrimonio con las propiedades recibidas en los parámetros de la función siempre y cuando el ID dado no coincida con el de una obra que ya esté registrada y que el ID cumplan con el formato adecuado de entrada.

**Clase**: Patrimonio

**Método**: registrar(string titulo, string ID\_Obra, string añoCreacion, string NombreA, string ubicacion, char estatus, string descripcion, image imagen, string apellidoA)

**Tipo de retorno**: Boolean

**Pre-condición**: ∃ conjuntoObras ^ CARD(conjuntoObras) ≥ 0

**Post-condición**: CARD(conjuntoObras) = CARD(conjuntoObras)’ + 1. Como esto es un atributo para comprobar que la prueba se satisface entonces hay que imprimirlo por consola.

**Casos de prueba**: Sea T⊆D y T ={(t1,t2)},{(t3)},{(t4)};

t1= (“Facultad de Odontología”,”A-A-5”,”1945-1953”,”Carlos”,”Facultad de Odontología”,’M’,””,null, “Villanueva”),

t2= (“Fachadas edificio de Odontología”,”OA-MUR-22”,”1957”,”Omar”,”Facultad de Odontología”,’D’,”Mosaicos vítreos industriales de 2x2 cm.\n Policromía”,null, “Carreño”),

t3= (“Facultad de Farmacia”,”OA-MUR-22”,”1945-1953”,”Carlos”,”Facultad de Farmacia”,’D’,””,null, “Villanueva”),

t4= (“Facultad de Farmacia”,”22-MUROA”,”1945-1953”,”Carlos”,”Facultad de Farmacia”,’D’,””,null, “Villanueva”);

donde t1 y t2 son casos de prueba para ID’s de obras válidos, t3 es el caso de un ID de obra repetido y t4 el de un intento de registro con formato inválido. Además, CARD(conjuntoObras) = 0

**Valor esperado**:

Para T =( t1,t2):

(registrar (“Facultad de Odontología”,”A-A-5”,”1945-1953”,”Carlos”,”Facultad de Odontología”,’D’,”descripción en blanco ”,null, “Villanueva”) :: CARD(conjuntoObras) = 1.

Se imprime por consola CARD(conjuntoObras)

(registrar (“Fachadas edificio de Odontología”,”OA-MUR-22”,”1957”,”Omar”,”Facultad de Odontología”,’D’,”Mosaicos vítreos industriales de 2x2 cm.\n Policromía”,null, “Carreño”) :: CARD(conjuntoObras) = 2.

Se imprime por consola CARD(conjuntoObras))

Para T =( t3):

(registrar (“Facultad de Farmacia”,”OA-MUR-22”,”1945-1953”,”Carlos”,”Facultad de Farmacia”,’D’,””,null, “Villanueva”) :: CARD(conjuntoObras) = 2.

Se imprime por consola CARD(conjuntoObras))

Para T=( t4):

(registrar (“Facultad de Farmacia”,”22-MUROA”,”1945-1953”,”Carlos”,”Facultad de Farmacia”,’D’,””,null, “Villanueva”) :: CARD(conjuntoObras) = 2.

Se imprime por consola CARD(conjuntoObras))

**Resultado**: Imprimir por consola los resultados del tamaño de conj\_obras para cada uno de los elementos de T cuando se ejecute la aplicación con este caso de prueba T

### Diagrama de Paquete

El diagrama de paquete se utiliza para organizar clases en los subdirectorios de un proyecto bajo NetBeans.

En la documentación de RUP el diagrama de paquete no pertenece a la disciplina de prueba, es un artefacto de la disciplina de implementación, pero en nuestro caso se agrega aquí, por que hacemos la disciplina de pruebas antes de la de implementación y definimos de una vez cómo vamos a organizar las clases antes de empezar a implementar.

Para hacer este diagrama hay que tomar en cuenta que:

1. Hay que tener el diagrama de diseño actualizado para poder realizar el diagrama de paquete
2. Hay que indicar cuál es el criterio que están utilizando para agrupar las clases del sistema en los paquetes
3. Si hay subpaquetes también hay que indicar el criterio utilizado
4. No utilizar la representación de clases dentro del paquete, colocar dentro de un recuadro de texto los nombres de todas las clases que están dentro de un cierto paquete
5. No se pueden duplicar clases, ellas deben estar en un solo paquete

Ejemplo:

De los siguientes diagramas de clases:

Diagrama de clases de diseño de Obras

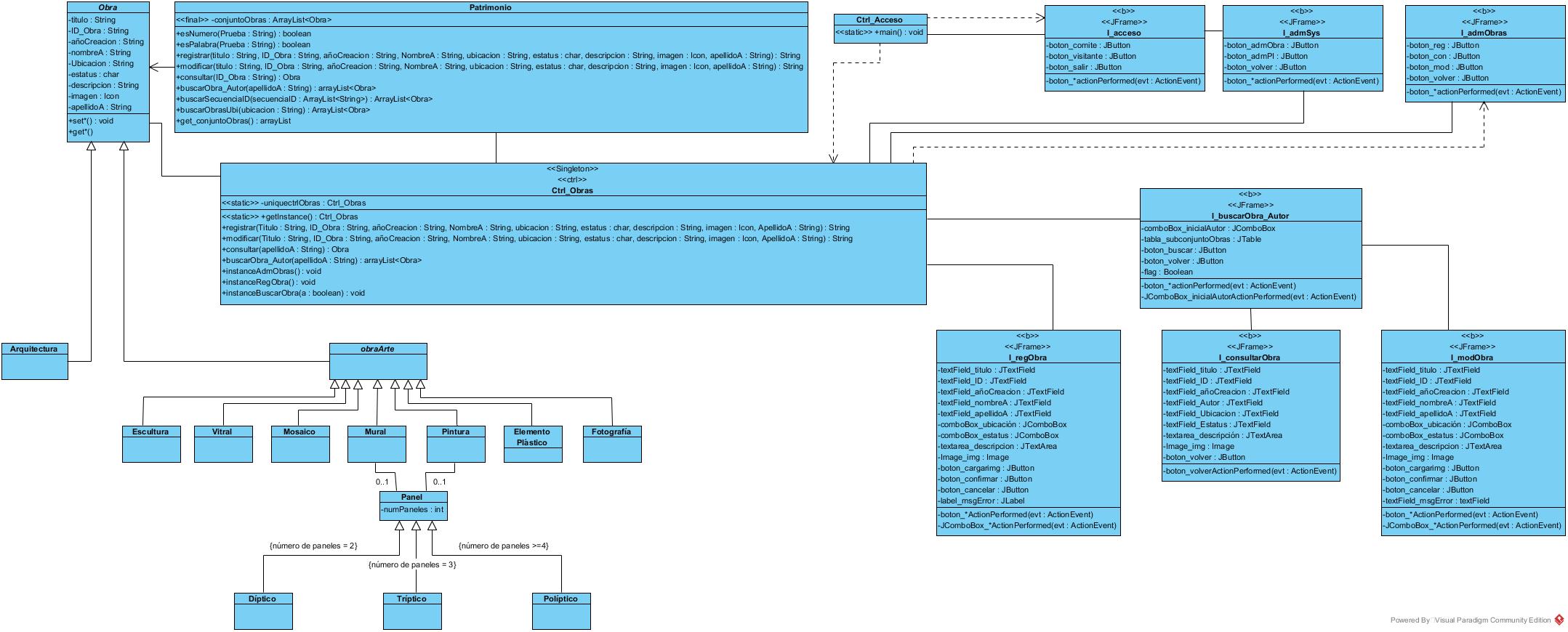
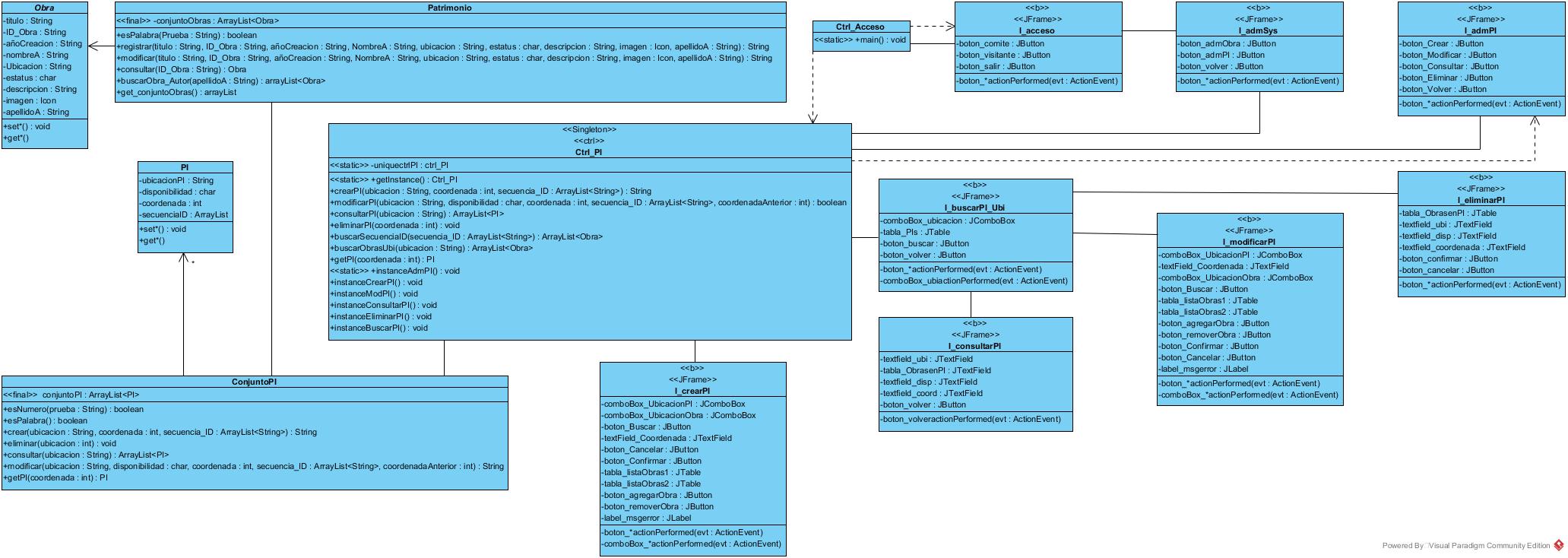


Diagrama de clases de diseño de Puntos de Interés



Tenemos el siguiente diagrama de paquete:

