# ESCUELA COLOMBIANA DE INGENIERÍA PROGRAMACIÓN ORIENTADA A OBJETOS

**Diseño y Pruebas. Interacción entre objetos. 2021-1**

# Laboratorio 2/6

### OBJETIVOS

Desarrollar competencias básicas para:

1. Desarrollar una aplicación aplicando BDD y MDD.
2. Realizar diseños (directa e inversa) utilizando una herramienta de modelado (astah)
3. Manejar pruebas de unidad usando un *framework ( junit)*
4. Apropiar nuevas clases consultando sus especificaciones (API java)
5. Experimentar las prácticas XP : **Coding** Code the [unit test first](http://www.extremeprogramming.org/rules/testfirst.html). **Testing** All code must have [unit tests](http://www.extremeprogramming.org/rules/unittests.html).



**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.
* En el foro de entrega deben indicar el estado de avance de su laboratorio y los problemas pendientes por resolver.
* Deben publicar el avance al final de la sesión y la versión definitiva en la fecha indicada en los espacios preparados para tal fin

# CONTEXTO

**Objetivo**

En este laboratorio vamos a construir una calculadora para vectores en espacios bi-dimensionales

: CalVectorial

**Conociendo el proyecto** [En lab02.doc]

1. El proyecto BlueJ “calVectorial” contiene una construcción parcial del sistema. Revisen el directorio donde se encuentra el proyecto. Describan el contenido considerando los directorios y las extensiones de los archivos.

**Rta:** El proyeto de calVectorial tiene como directorio doc y resource, el contenido son clases con extensiones .class, .txt, .ctxt y .html

1. Exploren el proyecto en BlueJ

¿Cuántas clases tiene? ¿Cuál es la relación entre ellas?

**Rta:** Tiene 4 clases, cada clase instancia una de la otra, como calVectorial, tiene un Hasmap de vector como atributo. Vector tiene a Angulo como atributo.

¿Cuál es la clase principal? ¿Cómo la reconocen?

**Rta:** CalVectorial, pues esta clase dirige a vector, y a su vez, vector dirige a Angulo.

¿Cuáles son las clases “diferentes”? ¿Cuál es su propósito?

**Rta:** AnguloTest, su propósito es asignar unas pruebas a Angulo para verificar que la construcción es correcta.

Para las siguientes dos preguntas sólo consideren las clases “**normales**”:

1. Generen y revisen la documentación del proyecto: ¿está completa la documentación de cada clase? (Detallen el estado de documentación de cada clase: encabezado y métodos)

**Rta:** No está completa, en la clase calVectorial falta completar la documentación tiene el encabezado, pero no métodos, y en Vector, falta algunas documentaciones tanto en métodos y encabezados.

1. Revisen las fuentes del proyecto, ¿en qué estado está cada clase? (Detallen el estado de las fuentes considerando: código, documentación y comentarios)

¿Qué son el código, la documentación y los comentarios?

**Rta:**

* calVectorial los métodos tienen su documentación, pero no tiene su respectivo código, además esta clase solo tiene un atributo.
* Vector los métodos están documentados (no todos) pero no tiene su respectivo código, además esta clase tiene 3 atributos.
* Angulo los métodos están documentados, pero no tiene su respectivo código, además esta clase tiene 4 atributos.

**Ingeniería reversa** [En lab02.doc CalVectorial.asta]

## MDD MODEL DRIVEN DEVELOPMENT

1. Complete el diagrama de clases correspondiente a CalVectorial. (No incluya la clase de pruebas)
2. ¿Qué tipo de contenedor está definido? Consulte la especificación y el API Java [1](#_bookmark0)¿Qué diferencias hay entre el nuevo contenedor y el ArrayList que conocemos?

**Rta:** Hashmap, la diferencia es que este nuevo contenedor necesita tanto una key(llave) como un value(valor) para cada uno de sus elementos, en cambio el ArrayList, simplemente guarda el elemento directamente.

1. https://docs.oracle.com/javase/8/docs/api/

**Conociendo Pruebas en BlueJ** [En lab02.doc \*.java]

### De TDD → BDD (TEST → BEHAVIOUR DRIVEN DEVELOPMENT)

Para poder cumplir con la prácticas XP vamos a aprender a realizar las pruebas de unidad usando las herramientas apropiadas. Para eso consideraremos implementaremos algunos métodos en la clase AnguloTest

* 1. Revisen el código de la clase AnguloTest.

¿cuáles etiquetas tiene (componentes con símbolo @)?

**Rta:** 3 etiquetas, @Before, @After, @Test

¿cuántos métodos tiene?

**Rta:** 9 métodos

¿cuántos métodos son de prueba?

**Rta:** 7 métodos son de prueba

¿cómo los reconocen?

**Rta:** Por su etiqueta @Test

* 1. Ejecuten los tests de la clase AnguloTest. (click derecho sobre la clase, Test All)

¿cuántas pruebas se ejecutan?

**Rta:**

Se ejecutan 7 pruebas.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

¿cuántos pasan las pruebas?

**Rta:** Ninguna

¿por qué?

**Rta:** Porque hay fallos en su prueba

* 1. Estudie las etiquetas encontradas en 1. Expliquen en sus palabras su significado.

**Rta:**

* @Before es aquel método que realiza los preparativos para antes de las pruebas.
* @Test es el método que realiza pruebas definiendo si es correcto o no el problema.
* @After es el método de cerrar la prueba de unidad, siempre y cuando @Before y @Test no lancen una excepción.
  1. Estudie los métodos assertTrue, assertFalse, assertEquals, assertArrayEquals, assertNull y fail de la clase assert del API JUnit [2](#_bookmark1). Explique en sus palabras que hace cada uno de ellos.

assertTrue nos permite tomar una condición como verdadera, de no ser así se ejecutará AssertionError, si su parámetro tiene de más un String, este lanzará un mensaje. Su parámetro es un boolean.

assertFalse nos permite tomar una condición como falsa, de no ser así se ejecutará AssertionError, si su parámetro tiene de más un String, este lanzará un mensaje. Su parámetro es un boolean.

assertEquals nos permite verificar que sus variables del mismo tipo son iguales ya sea objeto o número, de no ser así se ejecutará AssertionError, si su parámetro tiene de más un String, este lanzará un mensaje. Su parámetro son dos variables del mismo tipo, si son nulos, se consideran iguales.

assertArrayEquals nos permite verificar si listas del mismo tipo son iguales, de no ser así se ejecutará AssertionError, si su parámetro tiene de más un String, este lanzará un mensaje. Su parámetro son dos listas del mismo tipo.

assertNull nos permite verificar si el objeto es nulo, de no ser así se ejecutará AssertionError, si su parámetro tiene de más un String, este lanzará un mensaje. Su parámetro es un objeto.

Fail nos permite verificar si falla el test, si su parámetro tiene de más un String, este lanzará un mensaje. Este no tiene parámetro.

* 1. Investiguen la diferencia que entre un fallo y un error en Junit. Escriba código, usando los

métodos del punto 4., para lograr que los siguientes tres casos de prueba se comporten como lo prometen deberiaPasar, deberiaFallar, deberiaErrar.

**Prácticando Pruebas en BlueJ** [En lab02.doc \*.java]

### De TDD → BDD (TEST → BEHAVIOUR DRIVEN DEVELOPMENT)

Ahora vamos a escribir el código necesario para que las pruebas de AnguloTest pasen.

1. Determinen los atributos de la clase Angulo. Justifique la selección.

**Rta:**

Private double angulo;

Este es atributo de clase, lo cual va indicar el valor en el que se requiera, ya sea, grados, radianes o gradianes, y ademas con esta variable nos permite realizar distintas operaciones.

1. Implementen únicamente los métodos de Angulo necesarios para pasar todas las pruebas definidas. ¿Cuáles métodos implementaron?

**Rta:**

* El constructor, Angulo (double valor, in tipo)
* Grados(), radianes(), gradianes(), sume(Angulo ang), reste(Angulo a), multiplique(Angulo ang)

**Desarrollando CalVectorial**

### BDD - MDD

[En lab02.doc, CalVectorial.asta, \*.java]

Para desarrollar esta aplicación vamos a considerar algunos mini-ciclos. En cada mini-ciclo deben realizar los pasos definidos a continuación.

### Definir los métodos base de correspondientes al ciclo actual.

1. **Generar y programar los casos de prueba**

**(piense en los deberia y los noDeberia)**

### Diseñar los métodos

**(Use diagramas de secuencia. En astah, adicione el diagrama al método.**

**Revisen en los ejemplos el estándar y la configuración de los diagrama de secuencia.)**

### Generar y programar los casos de prueba de los métodos de la solución

**(piense en todos los debería y en todos los noDebería) [OPCIONAL]**

### Escribir el código correspondiente (no olvide la documentación)

1. **Ejecutar las pruebas de unidad (vuelva a 3 (a veces a 2). si no están en verde)**

### Completar la tabla de clases y métodos. (Al final del documento)

Ciclo 1 : Operaciones básicas de la calculadora: declarar, asignar un valor y consultar Ciclo 2 : Operaciones binarias básicas: asignar el resultado de sumas y restas

Ciclo 3 : Operaciones unarias básicas: asignar unitario, horizontal y vertical Ciclo 4 : Operaciones binarias avanzadas: producto y proyecciones

Ciclo 5 : Defina una nueva funcionalidad.

1. (<http://junit.org/javadoc/latest/>)

Completen la siguiente tabla indicando el número de ciclo y los métodos asociados de cada clase.

|  |  |  |
| --- | --- | --- |
| Ciclo | CalVectorial | CalVectorialTest |
| 1    2    3  4    5 | defina(String nombre)  asigne(String a, float longitud, float grados)  consultarEnPolares(String a)  consultarEnCartesianas(String a)  asigne(String a, String b, char op, String c)  asigne(String a, char op, String b)  asigne(String a, String b, char op, String c) | deberiaDeclarar()  noDeberiaDeclarar()  deberiaAsignarUnValor()  noDeberiaAsignarUnValor()  deberiaConsultar()  noDeberiaConsultar()  deberiaSumar()  noDeberiaSumar()  deberiaRestar()  noDeberiaRestar()  deberiaOperacionUnariaUnitario()  noDeberiaOperacionUnariaUnitario()  deberiaOperacionUnariaHorizontal()  noDeberiaOperacionUnariaHorizontal()  deberiaOperacionUnariaVertical()  noDeberiaOperacionUnariaVertical()  deberiaOperacionBinariaProducto()  noDeberiaOperacionBinariaProducto()  deberiaOperacionBinariaProyeccion()  noDeberiaOperacionBinariaProyeccion() |

## RETROSPECTIVA

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

**Rta:** 15 horas Acosta, 15 horas Olarte.

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

**Rta:**

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

**Rta:** Las pruebas, pues estas son fundamentales para ver si el código es correcto o no.

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

**Rta:** Aprender a implementar pruebas en el código, pues estas son una importante práctica XP.

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

**Rta:** A la hora de codificar las pruebas, leímos documentación para resolverlo.

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

**Rta:** Trabajar juntos y resolver los problemas en equipo, además de tener bien organizados los avances que se iban realizando.