



**División Académica de Ingeniería  
Departamento Académico de Computación**

# **Estructuras de Datos**

**Cuaderno de Ejercicios**

**Enero 2019**

# Contenido

Introducción .....	3
Conceptos básicos de la POO .....	4
Estructura de datos Pila .....	14
Estructura de datos Conjunto .....	17
Recursión.....	19
Estructura de datos Cola .....	21
Estructuras enlazadas.....	23
Estructura de datos Lista.....	25
Apéndices .....	28
<b>A - Uso del depurador (<i>debugger</i>) en NetBeans .....</b>	<b>29</b>
<b>B - Escritura/lectura de objetos en/de archivos.....</b>	<b>34</b>
<b>C - Pruebas Unitarias .....</b>	<b>37</b>
<b>D – Inclusión de clases definidas en otros proyectos.....</b>	<b>46</b>

## Introducción

En este cuaderno de ejercicios se presentan varios problemas donde se aplican los conceptos vistos en clase. Es muy importante resolver cada uno de ellos para asegurarse que los conceptos se entendieron y, por lo tanto, pudieron aplicarse en la solución de un problema.

Antes de resolver cada problema, ten en cuenta que:

- 1) Es muy importante que leas cuidadosamente todo el enunciado del problema.
- 2) Debes determinar todas las clases involucradas en el problema.
- 3) Para cada clase, debes identificar sus miembros: atributos y métodos.
- 4) Antes de codificar, asegúrate que tu solución (algoritmo) cubre todos los casos que puedan presentarse.
- 5) Revisa tu solución y asegúrate que es completa, eficiente y legible. Además, debes documentarla.
- 6) Debes probar tu solución para garantizar que la misma resuelve el problema planteado de manera **correcta** y **completa**.

Existen algunos atributos de calidad que son necesarios o deseables en los productos de software que generamos. Es importante que vayas desarrollando la disciplina necesaria para que todos los programas que escribas puedan considerarse productos de calidad. A continuación, se mencionan algunos de los atributos más importantes:

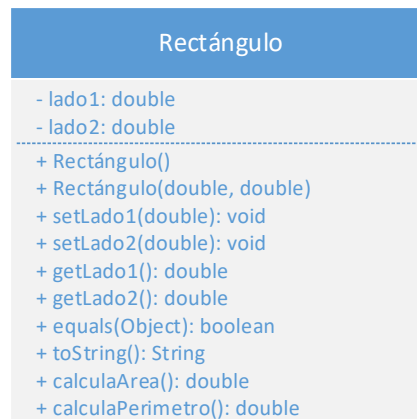
- Correctitud (Correctness)
- Robustez (Robustness)
- Performance (Eficciency)
- Amigabilidad (Friendliness)
- Verificabilidad (Verifiability)
- Mantenibilidad (Maintainability)
  - Reparabilidad (Reparability)
  - Evolucionabilidad (Evolvability)
- Reusabilidad (Reusability)
- Comprensibilidad (Understandability)
- Interoperabilidad (Interoperability)
- Oportunidad (Timeliness)

**Conceptos básicos de la POO:** herencia, polimorfismo, clases abstractas, interfaz, sobrescritura y sobrecarga de métodos.

### **PROBLEMA 1**

Escribe las clases que se mencionan más abajo. Luego compila y ejecuta tu programa. **Asegúrate que genera resultados correctos.**

Clase “Rectángulo” tal como se indica en el diagrama UML:



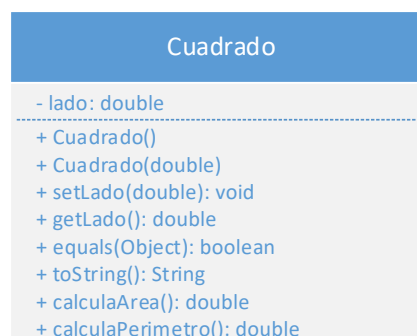
Clase “ComplejoVacacional” que tiene como atributos el nombre del complejo vacacional y los datos de sus *numAlb* albercas ( $1 \leq \text{numAlb} \leq 10$ ). Todas las albercas tienen forma rectangular. Decide qué métodos incluir en esta clase, si el comportamiento esperado de la misma es:

- 1) Calcular y regresar el total de metros de lona que se necesita para cubrir la superficie de todas ellas.
- 2) Calcular y regresar el total de metros lineales requeridos para cercar cada una de las albercas.

### **PROBLEMA 2**

Escribe las clases que se mencionan más abajo. Luego compila y ejecuta tu programa. **Asegúrate que genera resultados correctos.**

Clase “Cuadrado” tal como se indica en el diagrama UML:



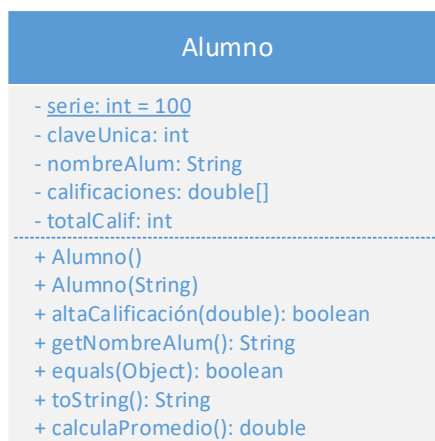
Clase “Restaurante” que tiene como atributos el nombre del restaurante y el tamaño de las *numMesas* mesas ( $1 \leq \text{numMesas} \leq 25$ ) que tiene disponible. Todas las mesas son cuadradas. Decide qué métodos incluir en esta clase, si el comportamiento esperado es el siguiente:

- 1) Calcular y regresar el total de metros de tela que se necesita para fabricar manteles para todas ellas.
- 2) Calcular y regresar el total de metros lineales requeridos de puntilla para poner en todos los extremos de los lados de los manteles.

### **PROBLEMA 3**

Escribe las clases que se mencionan más abajo. Luego compila y ejecuta tu programa. **Asegúrate que genera resultados correctos.**

Clase “Alumno” tal como se indica en el diagrama UML. Si lo necesitas puedes agregar otros atributos y/o métodos.



Clase “Escuela” que tiene un nombre, una dirección y *numAlumnos* alumnos ( $1 \leq \text{numAlumnos} \leq 50$ ). La escuela debe tener los métodos necesarios para que pueda mostrar el siguiente comportamiento:

- 1) Dada una cierta clave regresar en forma de cadena todos los datos del alumno correspondiente. Considera todos los casos que puedan presentarse.
- 2) Regresar el nombre y promedio de todos los alumnos.
- 3) Regresar el nombre del alumno que tenga el mayor número de materias aprobadas. Considera que sólo hay un alumno que cumple con esta condición.

### **PROBLEMA 4**

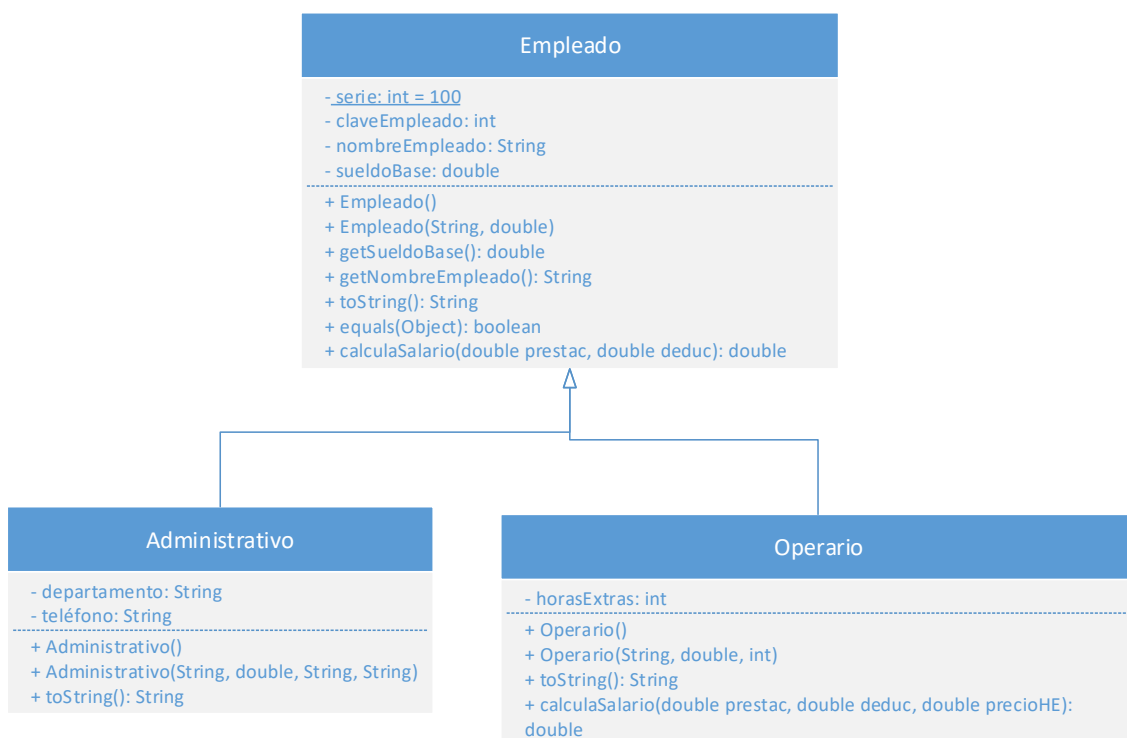
Considera el diagrama de clases que aparece más abajo (puedes agregar atributos y/o métodos si lo crees conveniente). Lee cuidadosamente toda la información que se proporciona. Analiza el problema y plantea la mejor solución. **Prueba tu programa.**

- El salario de un empleado, en general, se calcula como sueldo base + un % de prestaciones (prestac) – un % deducciones (deduc).
- El salario de un operario se calcula como sueldo base + un % de prestaciones (prestac) + las horas extras trabajadas \* el precio de cada hora – un % deducciones (deduc).

Se tiene además la clase “Empresa” que tiene como atributos el nombre de la empresa, la dirección, el nombre del dueño, los datos de todos sus empleados administrativos y los datos de

todos sus empleados operarios. De acuerdo a lo que se describe a continuación, decide qué métodos incluir en esta clase. El método main es el encargado de crear el (los) objeto(s) e invocar el(los) método(s) necesario(s) para:

- 1) Dar de alta tanto operarios como administrativos.
- 2) Generar un reporte con los nombres y los sueldos base de cada empleado administrativo.
- 3) Dada la clave de un empleado administrativo y un porcentaje de aumento, si está registrado, actualizar su sueldo.
- 4) Dada la clave de un empleado administrativo y un nombre de departamento, si está, registrar el cambio de departamento.
- 5) Dada la clave de un operario y los datos necesarios, si está, imprimir cuanto cobrará ese operario este mes.
- 6) Generar un reporte con los nombres de todos los operarios que tienen un sueldo base menor a cierta cantidad dada como parámetro. Además, debe incluir el total de empleados operarios que cumplen con esa condición.



## **PROBLEMA 5**

Lee cuidadosamente la información que se proporciona. Analiza el problema y plantea la mejor solución. En el Colegio “*Los libros al poder*” existen cuatro secciones, según el nivel de enseñanza que se imparte:

- ✓ Kinder: comprende maternal, Kinder y preescolar.
- ✓ Primaria: comprende los 6 años de primaria.
- ✓ Secundaria: comprende los 3 años de secundaria.
- ✓ Preparatoria: comprende los 3 años de preparatoria.

De cada alumno se conoce:

1. Kinder:

- ✗ Nombre completo
- ✗ CURP
- ✗ Fecha de nacimiento: mes/día/año
- ✗ Nombre completo de la madre
- ✗ Nombre completo del padre
- ✗ Nombre completo del tutor (este dato puede estar vacío si el alumno tiene a uno o a ambos padres).
- ✗ Nivel: maternal, kinder I, kinder II o preescolar

2. Primaria:

- ✗ Nombre completo
- ✗ CURP
- ✗ Fecha de nacimiento: mes/día/año
- ✗ Nombre completo de la madre
- ✗ Nombre completo del padre
- ✗ Nombre completo del tutor (este dato puede estar vacío si el alumno tiene a uno o a ambos padres).
- ✗ Escuela de procedencia
- ✗ Promedio año anterior
- ✗ Grado que cursa: primero, segundo, ..., sexto
- ✗ Calificaciones de los 5 bimestres del año que cursa (puede que no esté completo, dependiendo de la fecha)

3. Secundaria:

- ✗ Nombre completo
- ✗ CURP
- ✗ Fecha de nacimiento: mes/día/año
- ✗ Nombre completo de la madre
- ✗ Nombre completo del padre
- ✗ Nombre completo del tutor (este dato puede estar vacío si el alumno tiene a uno o a ambos padres).
- ✗ Escuela de procedencia
- ✗ Promedio año anterior
- ✗ Grado que cursa: primero, segundo, tercero
- ✗ Calificaciones de los 5 bimestres del año que cursa (puede que no esté completo, dependiendo de la fecha)
- ✗ Actividad artística/deportiva que practica como parte de su carga escolar

4. Preparatoria:

- ✗ Nombre completo
- ✗ CURP
- ✗ Fecha de nacimiento: mes/día/año
- ✗ Nombre completo de la madre
- ✗ Nombre completo del padre
- ✗ Nombre completo del tutor (este dato puede estar vacío si el alumno tiene a uno o a ambos padres).
- ✗ Escuela de procedencia
- ✗ Promedio año anterior
- ✗ Grado que cursa: primero, segundo, tercero
- ✗ Calificaciones de los 5 bimestres del año que cursa (puede que no esté completo, dependiendo de la fecha)
- ✗ Área de especialización (asumiendo que se elige al empezar el primer año de preparatoria)

### Actividades:

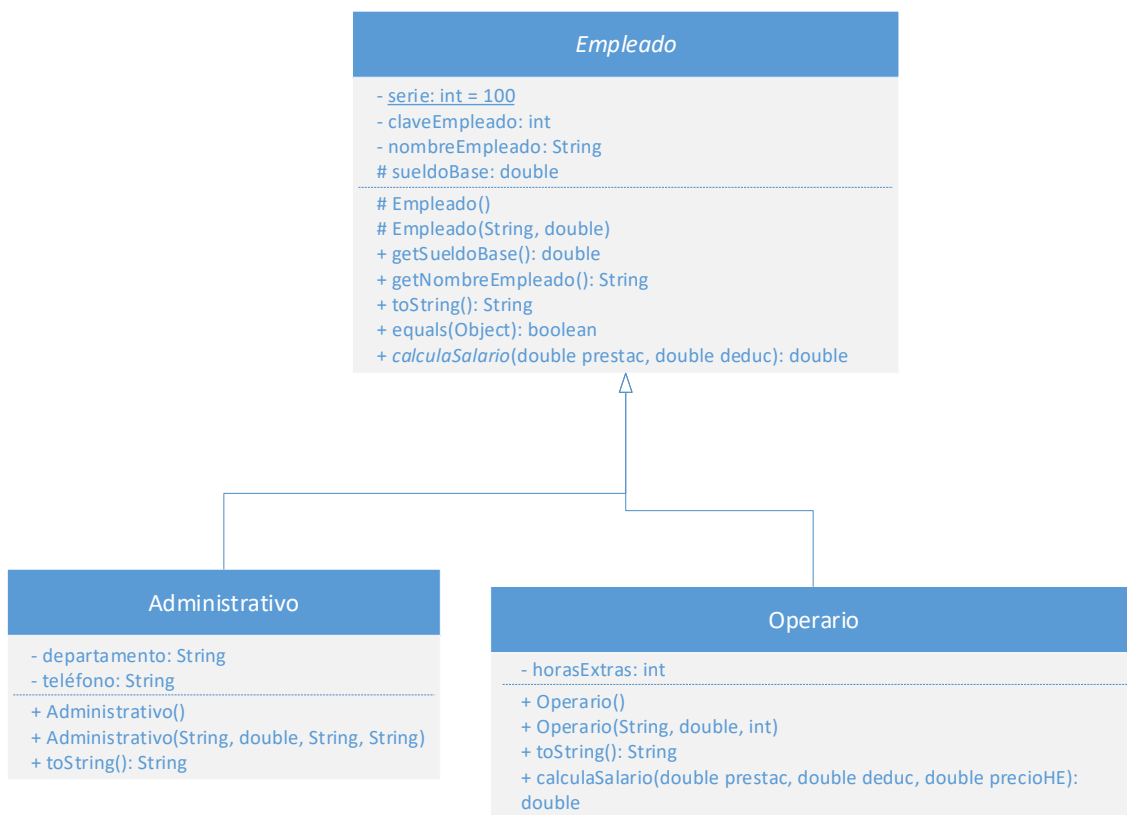
- 1) Define las clases necesarias para representar el conocimiento sobre los alumnos, disponible en la escuela.
- 2) Establece relaciones entre ellas de tal manera que representen la información de manera eficiente, **PERO** sin perder la claridad en los conceptos que representan.
- 3) Diseña y desarrolla una aplicación que pueda:
  - a) Almacenar los datos de los alumnos de cada una de las secciones de la escuela.
  - b) Generar un reporte de todos los alumnos –por sección- de la escuela ordenados alfabéticamente.
  - c) Obtener e imprimir el total de alumnos que estén cursando primaria y que tengan como escuela de procedencia la misma escuela.
  - d) Imprimir todos los datos de los alumnos de secundaria que tengan un promedio (hasta el momento) mayor a 9 y que estén a cargo de un tutor.
  - e) Imprimir todos los datos de los alumnos de preparatoria que hayan obtenido un promedio menor a 9 en el año escolar anterior y que estén en el área 1.
  - f) Dado el nombre de un alumno, si está en el colegio, imprimir todos sus datos.

**PRUEBA tu solución con pocos datos.**

**PARA la aplicación puedes simplificar las clases.**

### PROBLEMA 6

Lee cuidadosamente la información que se proporciona. Analiza el problema y plantea la mejor solución. **Prueba tu programa.**



**Nota:** El nombre de la clase y del método en *italica* indica que son **abstractos**.



- El salario de un administrativo se calcula como sueldo base + un % de prestaciones (prestac) – un % deducciones (deduc).
- El salario de un operario se calcula como sueldo base + un % de prestaciones (prestac) + las horas extras trabajadas \* el precio de cada hora – un % deducciones (deduc).

Se tiene además la clase “Empresa” que tiene como atributos el nombre de la empresa, la dirección, el nombre del dueño, los datos de todos sus empleados administrativos y los datos de todos sus empleados operarios. De acuerdo a lo que se describe a continuación, decide qué métodos incluir en esta clase. El método main es el encargado de crear el(los) objeto(s) e invocar el(los) método(s) necesario(s) para:

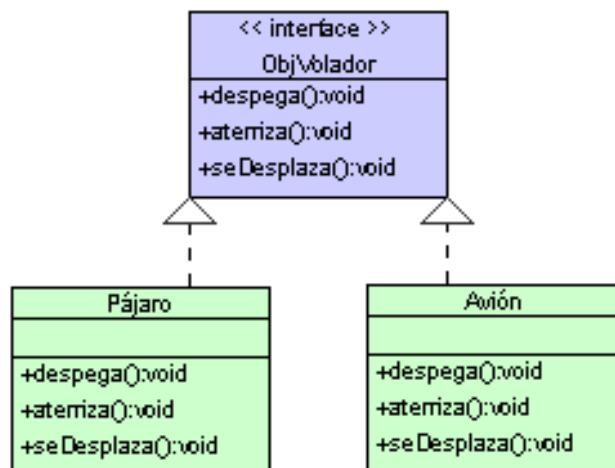
- 1) Dar de alta empleados, tanto operarios como administrativos.
- 2) Generar un reporte con los nombres y los sueldos base de cada empleado administrativo.
- 3) Generar un reporte con los nombres y los sueldos base de cada operario.
- 4) Dada la clave de un empleado administrativo y un porcentaje de aumento, si está registrado, actualizar su sueldo.
- 5) Dada la clave de un empleado administrativo y un nombre de departamento, si está, registrar el cambio de departamento.
- 6) Generar un reporte con los nombres de todos los operarios que tienen un sueldo base menor a cierta cantidad dada como parámetro. Además, debe incluir el total de empleados que cumplen con esa condición.
- 7) Suponiendo que el porcentaje de deducciones y aportaciones es el mismo para todos los administradores, calcular e imprimir el salario de cada uno de los administradores. Además imprimir el total de la nómina de la empresa (sólo con administradores).

(Se pueden omitir algunos de los incisos para simplificar el problema.)

**¿Qué tanto cambia esta solución con respecto a la del ejercicio 4? Indica ventajas/desventajas de la nueva solución. ¿Tuviste que usar alguno de los otros conceptos vistos?**

## **PROBLEMA 7**

Lee cuidadosamente la información que se presenta más abajo. Analiza el problema y plantea la mejor solución. **Prueba tu programa.** En el diagrama de clases que aparece más abajo, las líneas punteadas indican interface. Agrega los atributos y métodos que creas conveniente.

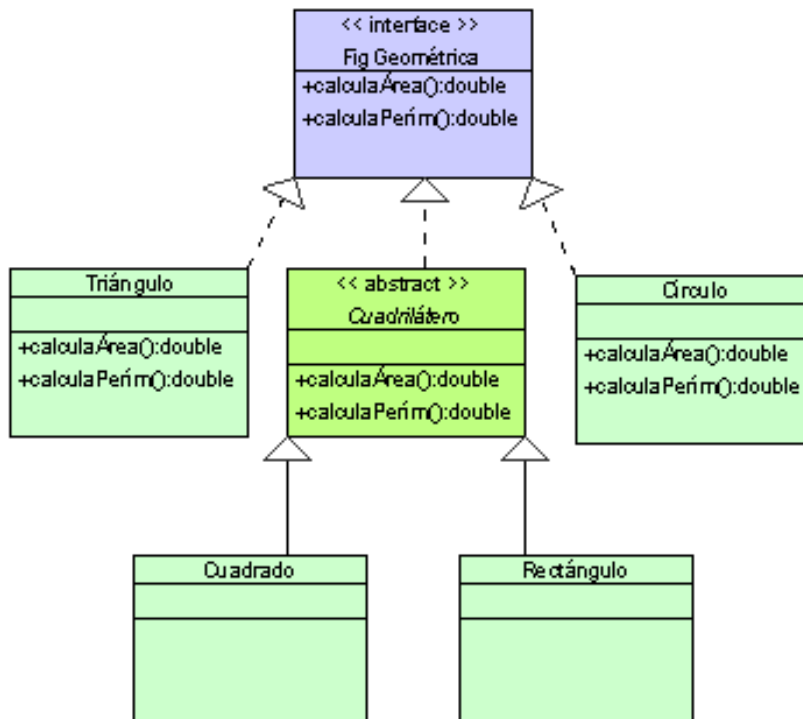


- 1) Define la interface y las clases que aparecen en el diagrama.
- 2) Decide los atributos y métodos a incluir.
- 3) Escribe una clase aplicación que pueda:
  - a) Declarar un arreglo de pájaros y otro de aviones.
  - b) Permita conocer, dada la clave de un pájaro y si fue almacenado, todos los datos de dicho pájaro. En caso contrario desplegar un mensaje adecuado.
  - c) Dada la clave de un avión, imprimir cuántos pasajeros puede transportar, siempre que el avión sea de pasajeros y se encuentre almacenado en el arreglo. En otro caso dar un mensaje.
  - d) Actualizar el hábitat de un pájaro. Decide qué datos necesitas.

## **PROBLEMA 8**

Analiza el siguiente diagrama de clases. Posteriormente:

- 1) Define todas las clases/interfaces representadas en el esquema que aparece más abajo.
- 2) Decide qué atributos y métodos deben incluirse para lograr una definición adecuada de los correspondientes conceptos.



## **PROBLEMA 9**

Retoma el problema 8. Desarrolla una aplicación que pueda, por medio de un menú, realizar las actividades que se describen a continuación.

- 1) Definir un arreglo polimórfico de FigGeométrica.
- 2) Dar de alta figuras.
- 3) Calcular e imprimir el área de todas las figuras almacenadas.
- 4) Encontrar e imprimir los datos del círculo más grande.
- 5) Calcular e imprimir el total de cuadrados.

- 6) Eliminar todos los triángulos equiláteros.

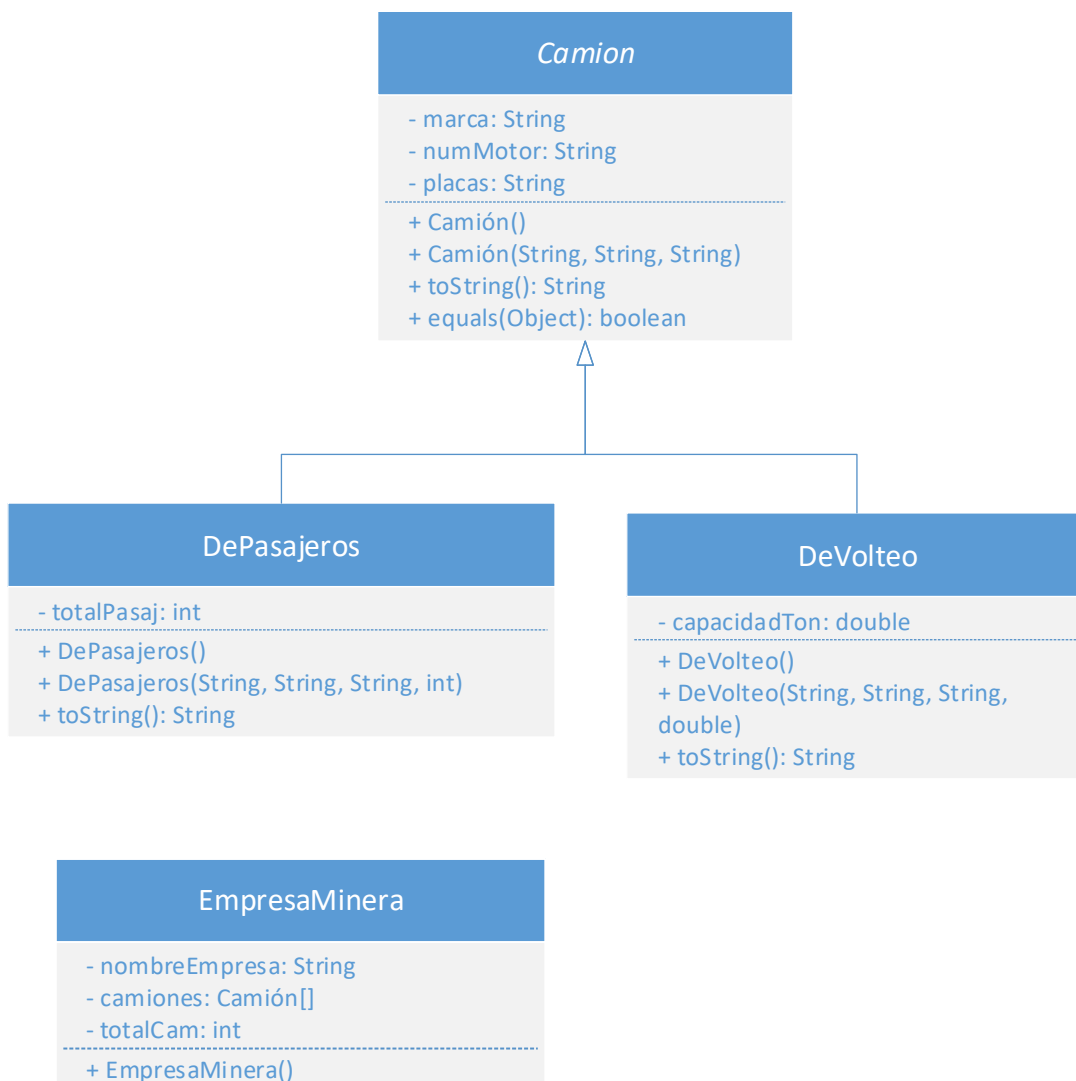
### **PROBLEMA 10**

Una empresa minera tiene camiones de volteo, que utiliza para el movimiento y traslado de tierra, y camiones de pasajeros para transportar a su personal. El esquema de clases que aparece más abajo corresponde a como se han modelado los datos. Puedes agregar atributos y/o métodos si lo crees conveniente.

En la clase EmpresaMinera puedes agregar otros atributos y debes definir los métodos necesarios para que se pueda realizar lo que se describe a continuación:

- 1) Regresar en forma de cadena todos los datos de los camiones de pasajeros.
- 2) Dada la placa de un camión de volteo, actualizar su capacidad de transporte (la nueva capacidad se recibe como parámetro).
- 3) Calcular y regresar el total de camiones de pasajeros que sean de una cierta marca (la marca se recibe como parámetro).
- 4) Calcular y regresar el total de toneladas de tierra que puede ser transportado simultáneamente.
- 5) Utiliza un archivo de objetos para guardar toda la información de la empresa minera.

**Hacer un main (o usar pruebas unitarias) para probar tu solución.**



## **PROBLEMA 11**

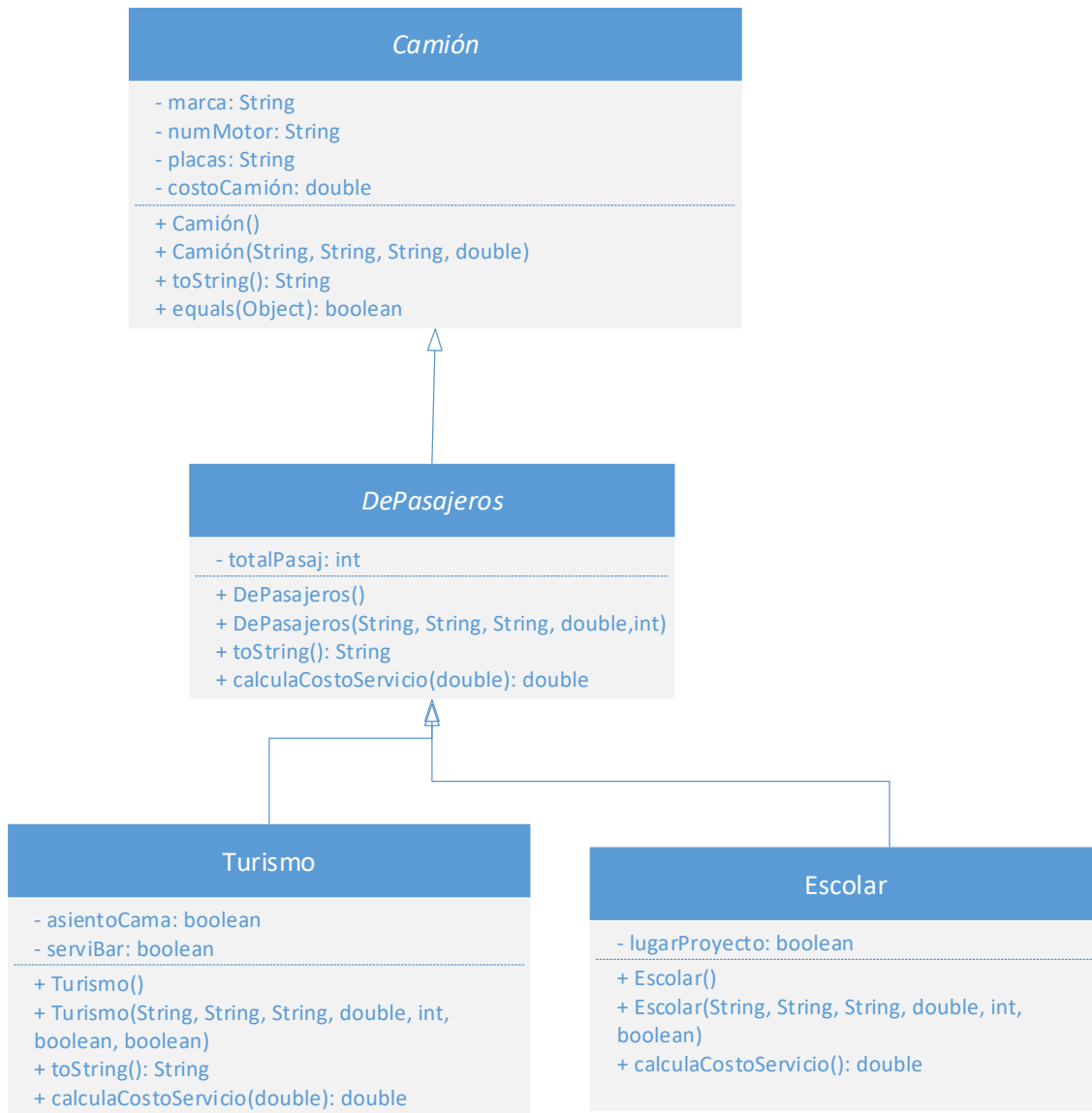
Observa el esquema de clases que aparece más abajo. Reúsa de problemas anteriores todo lo que puedas. Asimismo, considera:

- El costo del servicio para los camiones de pasajeros se calcula como el 0.01% del costo del camión dividido entre el número de pasajeros y multiplicado por la cantidad de kilómetros que recorrerá.
- El costo del servicio para los camiones de pasajeros, dedicados al transporte de escolares se calcula como el 0.01% del costo del camión dividido entre el número de pasajeros, multiplicado por \$250 y eso da el importe mensual que deberá abonar cada niño.
- El costo del servicio para los camiones de pasajeros, dedicados al transporte de turistas se calcula como el 0.01% del costo del camión dividido el número de pasajeros y multiplicado por la cantidad de kilómetros que recorrerá. Además, si el camión cuenta con servicio de bar, el costo se incrementa en un 5%, y si tiene asientos que pueden convertirse en camas, el costo se incrementa en otro 5%.

A su vez, se tiene información de una empresa de transporte, llamada “El rápido” que tiene tanto camiones escolares como de turismo. Define la clase “EmpTransp” que tiene como atributos el nombre de la empresa, el total de camiones y los datos de todos sus camiones. Los datos de éstos se encuentran almacenados en un archivo de objetos. Si quieres puedes agregar otros atributos y debes definir los métodos necesarios para que la clase tenga el siguiente comportamiento:

- 1) Llega un cliente para rentar el servicio de un camión para turistas. Dada la cantidad de pasajeros que se quiere transportar y la cantidad de kilómetros que se van a recorrer, regresar en forma de cadena las características de todos los camiones disponibles y el costo del servicio de cada uno de ellos.
- 2) Llega el director de una escuela para rentar el servicio de varios camiones escolares. Dada la cantidad máxima de niños que se desea transportar por camión y la cantidad de camiones requeridos, indicar si es posible satisfacer la demanda.
- 3) Dado el número de placas de un camión, indicar si está disponible. Si lo está, regresar qué tipo de unidad es.
- 4) Obtener y regresar en forma de cadena los números de placas de todos los camiones escolares que estén disponibles, que puedan llevar más de 20 alumnos y que cuenten con un espacio para que los estudiantes lleven sus proyectos.
- 5) Obtener y regresar el total de camiones de una cierta marca (la marca se recibe como parámetro), destinados al transporte de turistas que cuenten con servicio de bar y que el costo del servicio sea inferior a una cantidad recibida como parámetro. También se recibe como parámetro la cantidad de kilómetros a recorrer.
- 6) Utiliza un archivo de objetos para guardar toda la información de la empresa de transporte.

**Hacer un main (o usar pruebas unitarias) para probar tu solución.**



**Estructura de datos Pila:** definición de la clase Pila y su uso en la solución de problemas.

### **PROBLEMA 12**

Lee cuidadosamente la información que se presenta más abajo. Analiza el problema y plantea la mejor solución. **Prueba tu solución.**

Se requiere de un programa que sea capaz de analizar una cadena (un dato tipo String) para determinar si los paréntesis (izquierdos y derechos) están balanceados. En tu solución debes tener:

- Interface Pila: según lo estudiado en clase.
- Clase PilaA: según lo estudiado en clase.
- Clase RevisorParentesis: cuyo atributo es la cadena que se quiere analizar. Además, debe contar con un método que realice el análisis de la misma (con la ayuda de un objeto tipo Pila) y regrese verdadero si se cumple la condición establecida o falso en caso contrario.

### **PROBLEMA 13**

Lee cuidadosamente la información que se presenta más abajo. Analiza el problema y plantea la mejor solución. **Prueba tu solución.**

Se requiere de un programa que sea capaz de invertir el orden de los caracteres que forman una cadena (un dato tipo String). En tu solución debes tener:

- Interface Pila: según lo estudiado en clase.
- Clase PilaA: según lo estudiado en clase.
- Clase InvierteCad: cuyos atributos son la cadenaInicial (la cual se quiere invertir) y la cadenaInvertida (resultado de la inversión). Además, debe contar con un método que realice la inversión (con la ayuda de un objeto tipo Pila) de la cadenaInicial y la deje en cadenaInvertida.

### **PROBLEMA 14**

Escribe un método estático que reciba un objeto tipo pila genérica y que regrese el número de elementos que tiene almacenados. INDICA claramente de qué tipo son los datos que utilizas en tu solución. La pila NO puede modificarse, es decir, terminado el método la misma debe quedar como se recibió. **Prueba tu solución.**

### **PROBLEMA 15**

Escribe un método estático que reciba 2 pilas genéricas y regrese true si la primera de ellas “contiene” a la segunda. Una pila p1 contiene a una pila p2 si todos los elementos de p2 están también en p1. Las pilas NO pueden modificarse, es decir, terminado el método las mismas deben quedar como se recibieron. **Prueba tu solución.**

### **PROBLEMA 16**

Escribe un método estático que invierta el orden de los elementos de una estructura tipo Pila genérica. Para ello puedes usar cualquier estructura de datos como auxiliar. **Prueba tu solución.**

### **PROBLEMA 17**

Escribe un método estático que quite todos los elementos repetidos de una estructura tipo pila (si la pila tiene elementos repetidos, estos se encuentran en posiciones consecutivas). **Prueba tu solución.**

### **PROBLEMA 18<sup>1</sup>**

Retoma el problema 12 y agrégale un método que determine, no sólo que los paréntesis están balanceados, sino también los [] y las {}.

### **PROBLEMA 19<sup>1</sup>**

Supón que se realizan *push* y *pop* (en cualquier secuencia) sobre una pila. El push es para agregar los números del 0 al 9, en ese orden. Lo que regresa el pop se imprime. Analiza las siguientes secuencias e indica cuál de ellas NO puede obtenerse.

- (a) 4 3 2 1 0 9 8 7 6 5
- (b) 4 6 8 7 5 3 2 9 0 1
- (c) 2 5 6 7 4 8 9 3 1 0
- (d) 4 3 2 1 0 5 6 7 8 9

### **PROBLEMA 20<sup>1</sup>**

Escribe un método estático que reciba un entero y que genere y regrese su representación binaria en formato de número.

### **PROBLEMA 21<sup>1</sup>**

Modifica la interface PilaADT y la clase PilaA para incluir la siguiente funcionalidad:

multiPop(int n): void

El método no destructivo multiPop(int n) debe sacar n elementos de la pila, si es posible. Considera todos los casos que puedan presentarse.

### **PROBLEMA 22<sup>2</sup>**

Es posible usar un arreglo unidimensional para almacenar dos pilas: una a partir de la posición 0 y la otra a partir de la última posición. Modifica la interface PilaADT y la

---

<sup>1</sup> Tomado de: <http://www.cs.princeton.edu/courses/archive/spr01/cs126/exercises/adt.html>

<sup>2</sup> Tomado de:

[ftp://www.cc.uah.es/pub/Alumnos/G\\_Ing\\_Informatica/Data\\_Structures/Exercises/ExT1\\_StacksQueues.pdf](ftp://www.cc.uah.es/pub/Alumnos/G_Ing_Informatica/Data_Structures/Exercises/ExT1_StacksQueues.pdf)

clase PilaA para ajustarlas a las nuevas especificaciones. Decide de qué manera se indicará a qué pila afectarán las operaciones. Considera todos los casos que puedan presentarse.



**Estructura de datos Conjunto:** definición de la clase Conjunto y su uso en la solución de problemas.

**PROBLEMA 23**

Según lo visto en clase, define una interface `ConjuntoADT<T>` en la que establezcas el comportamiento esperado de un conjunto. Posteriormente la clase `Conjunto` con los atributos que lo caracterizan y los métodos que implementen las operaciones establecidas en la interface (agregar un elemento, quitar un elemento, conjunto vacío, cardinalidad, etc.).

**Prueba tu clase desarrollando una pequeña aplicación o usando pruebas unitarias.**

**PROBLEMA 24**

En la clase hecha en el ejercicio anterior agrega métodos para implementar las siguientes operaciones:

- 1) Unión de conjuntos.
- 2) Intersección de conjuntos.
- 3) Diferencia de conjuntos.

**PROBLEMA 25**

Define la clase `Escuela` con los atributos: nombre, dirección, y dos conjuntos de alumnos: uno formado por los alumnos que estudian alguna ingeniería y otro con los alumnos que estudian alguna licenciatura. Debes definir la clase `Alumno` con los atributos y métodos que creas conveniente. A la clase `Escuela` le puedes agregar otros atributos si lo consideras necesario. Además, le debes agregar los métodos requeridos para la siguiente funcionalidad:

- 1) Agregar alumnos a cualquiera de los conjuntos.
- 2) Quitar un alumno de cualquiera de los conjuntos.
- 3) Generar y regresar un conjunto formado con todos los alumnos de la escuela. El resultado debe darse en forma de cadena con todos los datos de los alumnos.
- 4) Generar y regresar un conjunto formado con todos los alumnos que están cursando una ingeniería y una licenciatura. El resultado debe darse en forma de cadena con todos los datos de los alumnos.
- 5) Generar y regresar un conjunto formado con todos los alumnos que están cursando solamente una ingeniería o una licenciatura, pero no ambas. El resultado debe darse en forma de cadena con todos los datos de los alumnos.
- 6) Calcular y regresar el promedio de todos los alumnos de Ingeniería.
- 7) Obtener y regresar el total de alumnos de licenciatura que son mayores que una edad (la cual se recibe como parámetro).

**NOTA:** Para este problema supón que no se pueden cursar dos ingenierías o dos licenciaturas.

### **PROBLEMA 26**

Se realizó una encuesta entre alumnos del ITAM para conocer qué idiomas extranjeros dominan. De acuerdo a las respuestas dadas se agruparon a los alumnos en los que dominan: inglés, francés y/u otro (alemán, chino, italiano, etc.), guardando sus nombres en 3 conjuntos. Escribe métodos estáticos para: 1) Generar una cadena con los nombres de todos los alumnos que dominan **inglés y francés**, 2) Obtener el total de alumnos que **sólo** dominan inglés, y 3) Generar una cadena con los nombres de los alumnos que dominan, al menos, **3 idiomas**.

**Recursión:** desarrollo de algoritmos recursivos. Recuerda que algunos de los problemas que debes resolver usando recursión pueden ser resueltos iterativamente de manera más eficiente, considerando el costo de recursos computacionales y la complejidad del código. Sin embargo, por su simplicidad son útiles para reforzar este nuevo concepto.

### **PROBLEMA 27**

Escribe un método estático que reciba un arreglo de enteros y regrese la suma de sus componentes. Debes usar recursión. **Prueba tu solución.**

### **PROBLEMA 28**

Escribe un método estático que reciba un arreglo de enteros e imprima cada uno de sus componentes: a) de izquierda a derecha y b) de derecha a izquierda. Debes usar recursión. **Prueba tu solución.**

### **PROBLEMA 29**

Define la clase ArregloGenérico en la cual escribas los métodos que se mencionan más abajo de manera recursiva. Analiza tus soluciones con respecto a las correspondientes implementaciones iterativas.

- 1) Búsqueda secuencial: recibe un dato tipo T y regresa la posición en la que está o un -1 si no está.
- 2) Búsqueda binaria: recibe un dato tipo T y regresa la posición en la que está o el negativo de la posición + 1 en la que debería estar.
- 3) toString(): regresa el contenido del arreglo en forma de cadena.
- 4) Método que encuentre y regrese la posición del mayor de los elementos del arreglo.
- 5) Método que elimine todas las ocurrencias de un cierto objeto en el arreglo. Decide qué parámetros se necesitan y qué tipo de resultado da el método.
- 6) Método de ordenación por selección directa.
- 7) Escribe una pequeña aplicación para probar tus métodos.

### **PROBLEMA 30**

Retoma el problema 24 (Conjunto) y programa los métodos solicitados de manera recursiva.

### **PROBLEMA 31**

Define la clase OperacionesArregloBidimensional en la cual escribas los métodos que se mencionan más abajo de manera recursiva. Analiza tus soluciones con respecto a las correspondientes implementaciones iterativas.

- 1) sumaPorRenglón: suma por renglón todos los elementos del arreglo bidimensional, regresando la suma obtenida.
- 2) sumaPorColumna: suma por columna todos los elementos del arreglo bidimensional, regresando la suma obtenida.
- 3) toString(): regresa el contenido del arreglo en forma de cadena.
- 4) sumaDiagonalPrincipal: suma y regresa los elementos de la diagonal principal.
- 5) sumaMatrices: suma dos matrices y regresa la matriz resultado.
- 6) multiplicaMatrices: multiplica dos matrices y regresa la matriz resultado

### **PROBLEMA 32**

Escribe un método estático que reciba una cadena que almacena el nombre de un archivo de texto y cuente y regrese el total de palabras almacenadas en dicho archivo. **Prueba tu solución.**

- Debes usar recursión.
- Puedes suponer que las palabras están separadas por un espacio en blanco.

**Estructura de datos Cola:** definición de la clase Cola y su uso en la solución de problemas.

### **PROBLEMA 33**

Se requiere de un método estático que invierta el orden de los elementos de una estructura tipo cola genérica. Para ello puedes usar cualquier estructura de datos como auxiliar. **Prueba tu solución.**

### **PROBLEMA 34**

Escribe un método estático que quite todos los elementos repetidos de una estructura tipo cola (si la cola tiene elementos repetidos, estos se encuentran en posiciones consecutivas). **Prueba tu solución.**

### **PROBLEMA 35**

Escribe un método estático que quite todas las ocurrencias de un cierto objeto de una estructura tipo cola. Es decir, dado un objeto, si se encuentra en la cola, se deben eliminar todas sus ocurrencias. **Prueba tu solución.**

### **PROBLEMA 36**

Modifica la interface ColaADT y la clase ColaA de tal manera que se agregue la siguiente funcionalidad, por medio de métodos no destructivos:

- a) `cuentaElementos()`: regresa el total de elementos almacenados en la cola.
- b) `consultaUltimo()`: regresa el último elemento almacenado en la cola, sin quitarlo.
- c) `multiQuita(int n)`: regresa un ArrayList almacenando los n elementos quitados de la cola.
  - Considera todos los casos que puedan presentarse.
  - Prueba tu solución.

### **PROBLEMA 37**

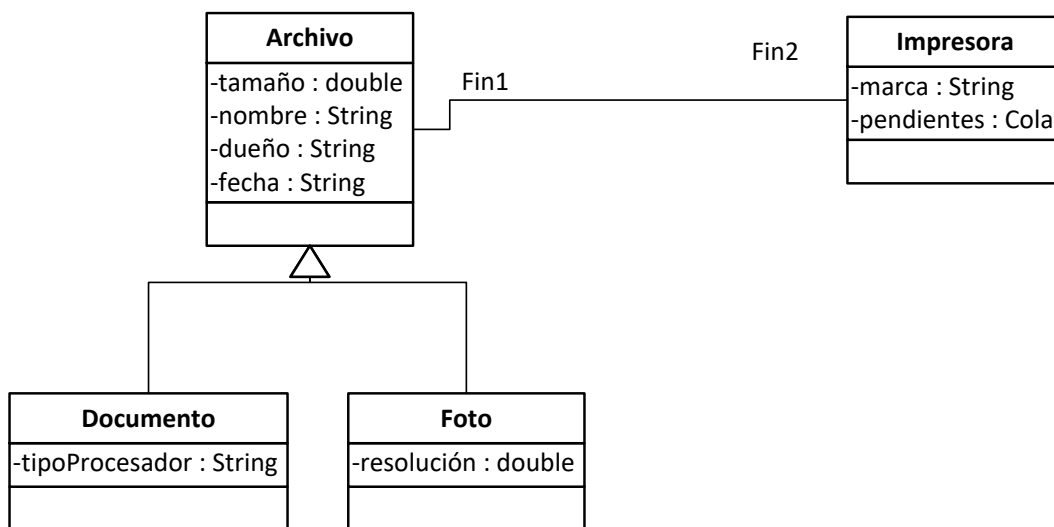
Suponiendo que se tiene una cola de personas que están esperando subirse a un avión. Las personas se fueron encolando por orden de llegada. Sin embargo, por disposiciones del personal de la aerolínea, se les pide reubicarse en la cola de tal manera que el orden de embarque quede determinado por la edad del pasajero: primero los de más edad. Escribe una clase con un método estático que reciba una cola de tipo Persona y asegúrate que los elementos de la misma queden ordenados usando el criterio dado más arriba. Se considerarán años cumplidos al 31 de diciembre del año en curso. Debes asegurarte que la clase Persona tenga un método que, a partir de la fecha de nacimiento del pasajero, calcule y regrese los años que tiene o que tendrá al 31 de diciembre.

### **PROBLEMA 38**

Observa el diagrama de clases que aparece más abajo. Debes hacer una aplicación que permita administrar una *impresora*, según lo especificado en el diagrama. La impresora tiene, entre sus atributos, los archivos que debe imprimir, los mismos están encolados. Es decir, se deben imprimir en el orden en el cual llegaron. Si requieres atributos adicionales, puedes agregarlos. En **Impresora** debes agregar los métodos necesarios para poder tener la siguiente funcionalidad. Decide si se necesitan parámetros y el tipo de resultado de los métodos.

- 1) Inicialmente se leen de un archivo de objetos varios *archivos* que deben ser impresos. Pueden ser documentos o fotos. Los archivos se encolan en el orden en el cual se leen.
- 2) Llega un nuevo archivo para ser impreso. Debe agregarse a la cola.
- 3) La impresora se desocupó, debe imprimir el siguiente archivo.
- 4) Eliminar todas las fotos de la cola de impresión.
- 5) Eliminar todos los documentos cuyo tamaño sea mayor a 500 (KB).

**Prueba tus clases desarrollando una pequeña aplicación.**



**Estructuras enlazadas:** definición de estructuras de datos enlazadas y su uso en la solución de problemas.

### **PROBLEMA 39**

Según lo visto en clase, define las clases que representen a un nodo (Nodo) y a una estructura enlazada (EE). Agrega todos los métodos necesarios para implementar el comportamiento esperado.

### **PROBLEMA 40**

En la clase EE escribe el método: **public boolean eliminaAnteriorA(T info)** que elimine el nodo anterior al que ocupa la “info”. Regresa true si lo pudo eliminar y false en caso contrario. CONSIDERA TODOS LOS CASOS QUE PUEDEN PRESENTARSE. **Prueba tu solución.**

### **PROBLEMA 41**

En la clase EE escribe el método: **public boolean eliminaSiguienteDe(T info)** que elimine el nodo siguiente al que ocupa la “info”. Regresa true si lo pudo eliminar y false en caso contrario. CONSIDERA TODOS LOS CASOS QUE PUEDEN PRESENTARSE. **Prueba tu solución.**

### **PROBLEMA 42**

En la clase EE escribe el método: **public boolean insertaAntesQue(T refer, T nuevo)** que inserta un nodo (con nuevo como información) antes que el nodo dado como referencia (refer). Regresa true si lo pudo insertar y false en caso contrario. CONSIDERA TODOS LOS CASOS QUE PUEDEN PRESENTARSE **Prueba tu solución.**

### **PROBLEMA 43**

En la clase EE escribe el método: **public int eliminaTodosRepetidosOrdenado()** que elimine todos los elementos repetidos de una EE, dejando sólo una ocurrencia de cada dato. Además, regresa como resultado el total de elementos eliminados. Supón que los elementos de la EE están ordenados y por lo tanto los repetidos ocupan posiciones consecutivas. **Prueba tu solución.**

### **PROBLEMA 44**

En la clase EE escribe el método: **public int eliminaTodosRepetidosDesordenado()** que elimine todos los elementos repetidos de una EE, dejando sólo una ocurrencia de cada dato. Además, regresa como resultado el total de elementos eliminados. Supón que los elementos de la EE están desordenados y por lo tanto los repetidos pueden estar en cualquier posición de la EE. **Prueba tu solución.**

#### **PROBLEMA 45**

Escribe un método en la clase EE que reciba como parámetro un objeto de tipo EE (objEE) y modifique a la EE que llama al método (*this*), *mezclando sus nodos con los de objEE* de acuerdo a lo descrito a continuación. A *this* se le deberá intercalar un nodo de *objEE* entre cada par de ella. Es decir, quedará el primer nodo de *this* seguido del primero de *objEE*, luego el segundo de *this* seguido del segundo de *objEE*, y así sucesivamente. Si tienen diferente cantidad de nodos, al final quedarán los nodos de la EE más larga. SE DEBEN USAR LOS NODOS DE LAS EE, NO SÓLO LAS INFORMACIONES QUE ALMACENAN. **Prueba tu solución.** Una vez ejecutado el método, ¿en qué estado quedan las dos EE involucradas?

#### **PROBLEMA 46**

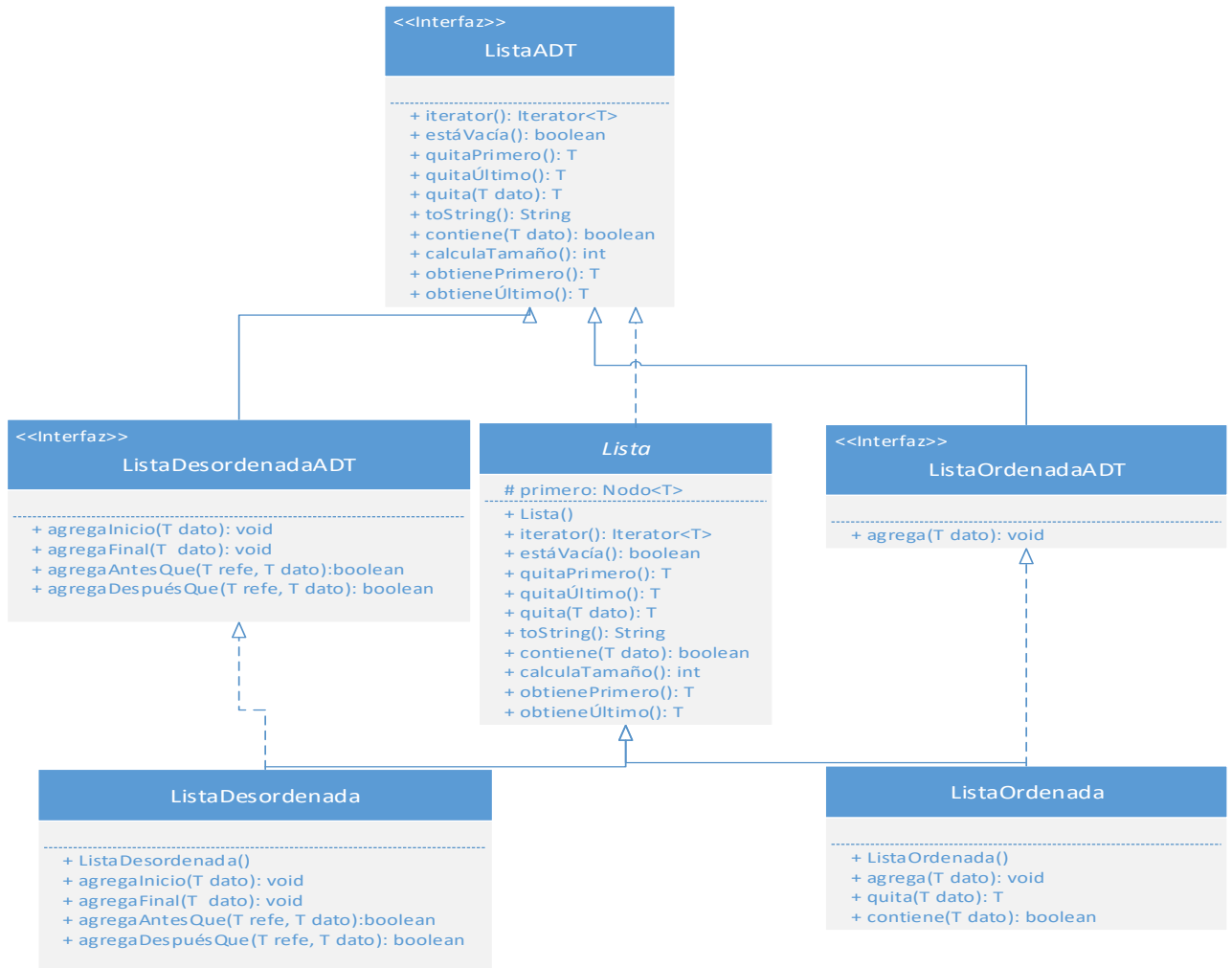
Rescribe las clases Pila y Cola usando una EE como base para su implementación (en vez de un arreglo genérico).

#### **PROBLEMA 47**

Rescribe la clase Conjunto utilizando nodos enlazados, en lugar de un arreglo. La nueva clase también debe implementar la interface **ConjuntoADT** ya definida.



**Estructura de datos Lista:** definición de la clase Lista (y sus variantes) y su uso en la solución de problemas.



#### **PROBLEMA 48**

En la clase **ListaOrdenada<T>** vista en clase rescribe de manera recursiva los métodos: **toString()** y el de búsqueda considerando que los elementos están ordenados. **Prueba tu solución.**

#### **PROBLEMA 49**

Debes usar las clases **NodoSimple<T>** y **ListaOrdenada<T>** para almacenar los datos de un grupo de personas. En la clase **Persona** incluye los atributos y métodos que creas conveniente. Define la clase **Club** cuyos atributos son *el nombre del club* y *una lista de socios* (personas) ordenados por nombre. Debe tener métodos que permitan dar de alta un socio, quitar un socio y consultar los datos de un socio.

**Escribe una pequeña aplicación para probar tu solución.**

### **PROBLEMA 50**

Agrega a la clase ListaOrdenada<T> un método equals (Object otra): boolean que permita comprobar si dos listas ordenadas son iguales. Recibe como parámetro una lista y regresa un valor booleano (true si son iguales, false en caso contrario). Dos listas son iguales si tienen **exactamente el mismo** contenido y en el mismo orden. Asegúrate que tu clase ListaOrdenada<T> sigue siendo general. No se deben alterar los contenidos de las listas dadas. Implementa el método solicitado de manera iterativa y de manera recursiva. **Prueba tu solución.**

### **PROBLEMA 51**

Escribe un programa que, usando las clases NodoSimple<T> y ListaOrdenada<T> sea capaz de almacenar (ordenados alfabéticamente) los nombres de los países de América, según la ubicación: Norte, Centro y Sur, en tres listas distintas. Posteriormente, deberá formar una lista que contenga los nombres de todos los países del continente, también ordenados alfabéticamente. Las 3 listas iniciales no deben alterarse. Para formar la nueva lista se deben ir “mezclando” el contenido de las otras tres manteniendo el orden. Modulariza y **prueba tu solución.** Los nombres de los países se deben leer de un archivo de texto.

### **PROBLEMA 52**

Define la clase MateriaSemestre con los siguientes atributos: nombre de la materia, nombre del salón en el cual se imparte, nombre del semestre en el cual está siendo impartida, nombre del maestro que la imparte, lista de alumnos inscritos y lista de libros usados como fuente para dicha materia. Esta clase tiene, además de los métodos esperados (por ejemplo constructores, toString(), ...) los que se describen más abajo. Define las clases Alumno y Libro con los atributos y métodos que creas conveniente. En el main debes incluir las instrucciones necesarias para probar tu solución.

- 1) Un método que permita dar de baja un alumno de dicha materia.
- 2) Un método que permita dar de alta un alumno de dicha materia.
- 3) Un método que permita cambiar el salón en el cual se imparte la materia.
- 4) Un método que permita agregar un libro como referencia de dicha materia.
- 5) Un método que permite quitar uno de los libros dados como referencia de dicha materia.
- 6) Un método que calcula y regresa el promedio de todos los alumnos inscritos en la materia.
- 7) Un método que, dada la clave de un alumno, regrese en forma de cadena los datos del alumno. Si el alumno no se encuentra registrado, deberá regresar null.
- 8) Un método que reciba un entero (n>0) y un *promedio* y que regrese true si hay, al menos, n alumnos con un promedio mayor al dado. En caso contrario deberá regresar false.
- 9) Un método que, dado el nombre de un autor, obtenga y regrese el total de libros que se tienen de ese autor.

- ✓ Si corresponde decide qué dato (o datos) debe suministrar el usuario.
- ✓ Decide qué tipo de listas resulta más adecuada para almacenar los alumnos y los libros.
- ✓ Considera todos los posibles casos de errores.
- ✓ Trata de generalizar tu solución.
- ✓ Cuida la eficiencia de tu solución.

### **PROBLEMA 53**

Define una clase (NodoDoble) para representar un nodo con doble dirección, una de ellas para guardar la dirección del nodo que lo precede y la otra para guardar la dirección del nodo que le sigue. Luego escribe la clase ListaDoble<T> de tal manera que utilice objetos de tipo NodoDoble para formar una lista. Incluye los métodos necesarios para:

- 1) Buscar un elemento en una lista desordenada. Escribe el método de manera recursiva.
- 2) Generar una cadena (toString(): String) con toda la información de la lista. Escribe el método de manera recursiva.
- 3) Determinar si una lista está contenida en otra (estáContenida(ListaDoble<T> otra): boolean). Implementa el método de manera iterativa y recursiva. El método recibe como parámetro una lista y regresa un valor booleano igual a true si la lista recibida como parámetro está contenida en la lista que invoca al método, y false en caso contrario. **Una lista está contenida en otra si todos sus elementos (sin importar el orden) están en la segunda.**
- 4) Determinar si una lista tiene sus elementos ordenados de manera creciente (estáOrdenadaCreciente(): boolean). Realiza una solución iterativa y una recursiva.
- 5) Determinar si una lista NO tiene sus elementos ordenados de manera creciente (noEstáOrdenadaCreciente(): boolean).

### **PROBLEMA 54**

Define la clase Empresa que tiene, entre sus atributos, dos listas, una de **clientes** y otra de **deudores**. Escribe un método que, usando alguno de los ya programados, determine si todos los deudores de la empresa son sus clientes. El método regresa un boolean. Puedes usar una interfaz gráfica para hacer más amigable tu sistema. A la clase Empresa puedes agregarle los atributos y métodos que creas necesarios. Para definir a los clientes y deudores puedes usar la clase Persona de ejercicios anteriores.

## Apéndices

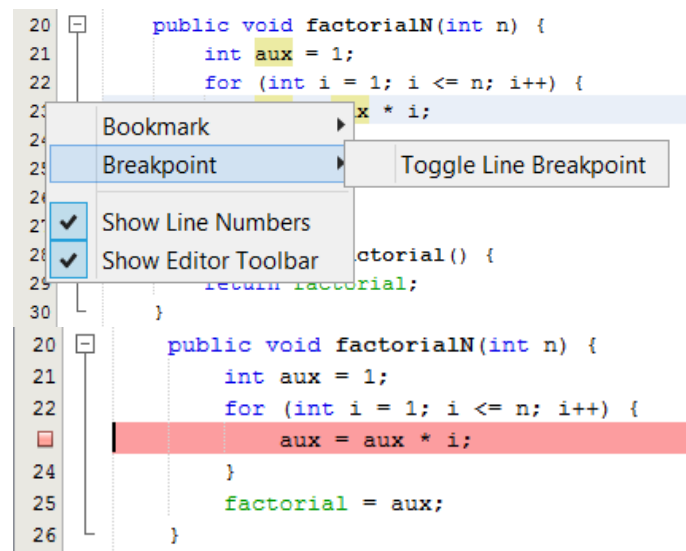
## A - Uso del depurador (*debugger*) en NetBeans

Depurar un programa es analizar su código y sus variables interactivamente durante la ejecución del mismo. Generalmente se usa para identificar fallas o faltas en el código.

Se utilizan **breakpoints** en el código para indicar los puntos en los cuales se quiere revisar, ya que estos provocan la detención de la ejecución. Una vez que el programa es detenido se pueden analizar las variables, cambiar su contenido, etc.

### ¿Cómo colocar los breakpoints?

Para colocar un breakpoint en la línea de código elegida, se debe dar clic derecho en el pequeño margen izquierdo y seleccionar **Breakpoint** → **Toggle Line Breakpoint** en el menú desplegable. Otra manera es dar un clic sobre el número de la línea.



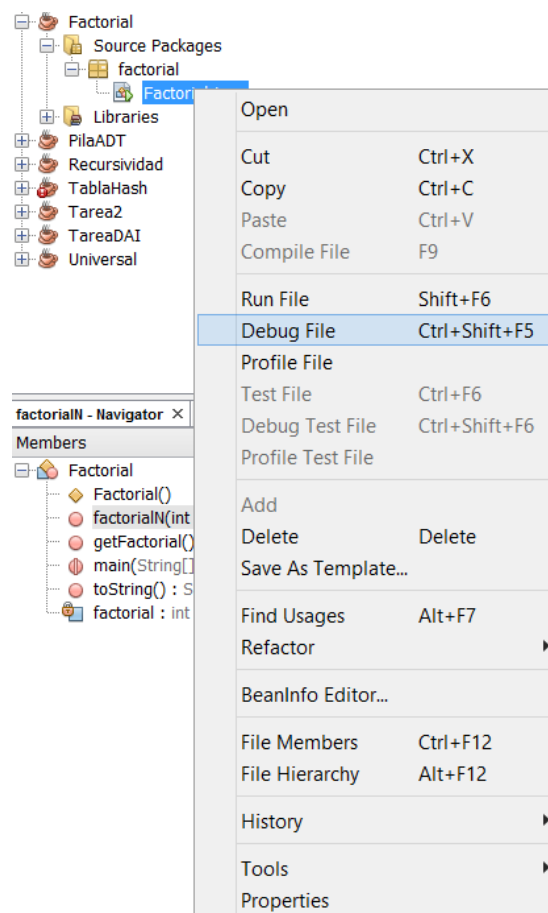
De inmediato aparecerá un cuadro rojo y la línea seleccionada será resaltada en el mismo color.

### ¿Cómo quitar los breakpoints?

Para quitar un breakpoint se requiere dar un clic sobre el cuadro rojo.

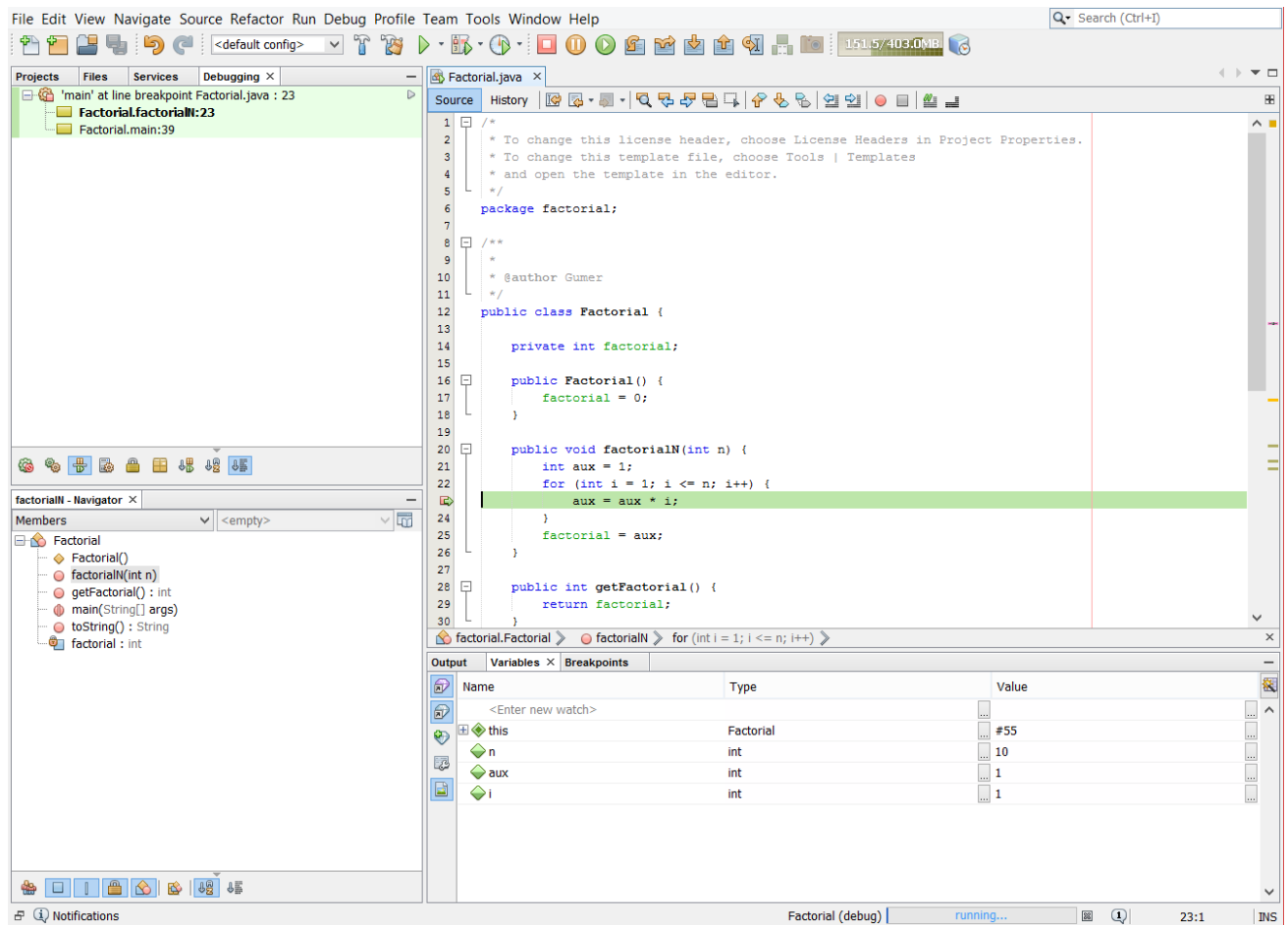
### ¿Cómo iniciar el depurador?

Para iniciar el depurador se puede proceder de diversas maneras. Algunas de ellas son: 1) seleccionar el archivo de Java que será ejecutado, dar clic derecho en él y seleccionar **Debug File**, 2) presionar las teclas Ctrl + Shift + F5 simultáneamente, 3) elegir **Debug** del menú principal (barra superior) y luego **Debug File**. A continuación aparece la imagen correspondiente a la primera opción.



Si no se ha definido ningún breakpoint, el programa correrá normal. Para utilizar el depurador se requiere definir primero los breakpoints.

Una vez iniciado el depurador, NetBeans abrirá una vista similar a la siguiente:



El programa se ejecuta inmediatamente y el depurador está listo para usarse. Automáticamente se detiene en el primer breakpoint para que el usuario pueda empezar su análisis. La línea de código con el breakpoint en turno será marcada de color verde.

### ¿Cómo se analiza el programa?

Junto a la consola (Output) aparecerán las ventanas *Variables* y *Breakpoints*, en cada una se podrán visualizar sus correspondientes elementos.

La pestaña *Breakpoints* permite eliminar o desactivar breakpoints.









La pestaña *Variables* muestra tipos y contenidos de las variables a medida que las mismas se van usando durante la ejecución. En esta ventana también se puede cambiar el valor asignado a una variable en tiempo de ejecución.

En la parte superior derecha de la ventana de NetBeans, se muestra el uso de memoria actual de tu programa.



### ¿Cómo controlar la ejecución del programa mientras se usa el depurador?

En la barra de herramientas aparecen botones para controlar la ejecución del programa, aunque también se pueden usar las teclas correspondientes.

Tecla	Descripción
F5 	Ejecuta la línea actualmente seleccionada y avanza hasta el siguiente breakpoint o hasta el final del programa según sea el caso.
Shift + F5 	Finaliza el depurador y con él la ejecución del programa.
F7 	Hace que el depurador "entre en" (step into) el código. Muestra el código de métodos invocados y profundizará en él cuando se realizan muchas llamadas a métodos anidados.
Ctrl + F7 	Hace que el depurador salga de la función en la que se encuentra y avanza a la siguiente línea del programa.
F8 	Provoca que el código "pase a lo siguiente (step over)". Ignora el funcionamiento interno de las llamadas a funciones y sólo se enfoca en el valor que retorna.
Shift + F8 	Evalúa la expresión que consiste en llamadas a múltiples métodos, si no hay llamadas a métodos actúa como un F8 (step over).
F4 	Continúa la ejecución del programa hasta que se tope con el cursor.
	Pausa la ejecución.



## Ejemplo

```
public class Factorial {  
  
    private int factorial;  
  
    public Factorial() {  
        factorial = 0;  
    }  
  
    public void factorialN(int n) {  
        int aux = 1;  
        for (int i = 1; i <= n; i++) {  
            aux = aux * i;  
        }  
        factorial = aux;  
    }  
  
    public int getFactorial() {  
        return factorial;  
    }  
  
    public static void main(String[] args) {  
        Factorial prueba = new Factorial();  
        int n = 10;  
        prueba.factorialN(n);  
        System.out.println("El factorial de " + n + " es igual a " + prueba.getFactorial());  
    }  
}
```

1. Captura el código anterior.
2. Coloca un breakpoint dentro del método factorialN () de la clase Factorial. Utiliza el depurador para correr el programa y sigue la ejecución del método.
3. Podrás analizar cómo va aumentando la variable i y cómo va cambiando la variable aux. Cuando acabe de ejecutarse el método, el depurador regresará al main para seguir con la ejecución e imprimir en la consola.

## B - Escritura/lectura de objetos en/de archivos

La *serialización* de objetos (en Java) permite que los mismos se puedan convertir en una secuencia de bytes para ser guardados en un archivo y posteriormente obtenerse la información, recuperando asimismo la “forma” del objeto. Es decir, se pueden guardar y recuperar objetos completos (con todos sus miembros), con sólo una operación de escritura o lectura respectivamente.

La clase cuyos objetos van a ser *serializables* debe implementar la interfaz de java **Serializable**. Por ejemplo:

```
public class Alumno implements java.io.Serializable {  
    ...  
}
```

o bien:

```
import java.io.Serializable;
```

```
public class Alumno implements Serializable {  
    ...  
}
```

- **No requiere implementar ningún método.**
- En caso de herencia entre clases, sólo la clase base debe implementar la interfaz **Serializable**. Cuando se almacena en archivo un objeto de una clase derivada (no importa el nivel de herencia) se conservan todos sus miembros, los propios y los heredados.
- En caso de uso de interfaces, sólo la interfaz extiende la interfaz **Serializable**. En el archivo se pueden guardar objetos de cualquiera de las clases que implementen a la interfaz.

### Escritura en archivo

Se utiliza el flujo de datos *ObjectOutputStream* y el flujo de salida a archivo *FileOutputStream* para escribir en un archivo. Para definir los flujos se hace:

```
FileOutputStream fileSal = new FileOutputStream("Alumnos.obj");  
ObjectOutputStream salida = new ObjectOutputStream(fileSal);
```

O bien:

```
ObjectOutputStream salida = new ObjectOutputStream(new  
FileOutputStream("Alumnos.obj"));
```

Para escribir los objetos en el archivo se hace:

```
// Suponiendo que al1 es un objeto tipo Alumno previamente instanciado  
salida.writeObject(al1);
```

Finalmente se cierra el archivo:

```
salida.close();
```

**NOTA:** el archivo se guarda en la carpeta del proyecto en el cual fue creado. Si se desea otra dirección, se debe especificar.

## Lectura de archivo

Se utiliza el flujo de datos *ObjectInputStream* y el flujo de entrada de archivo *FileInputStream* para leer objetos de un archivo.

Para definir los flujos se hace:

```
FileInputStream fileEn = new FileInputStream("Alumnos.obj");  
ObjectInputStream entrada = new ObjectInputStream(fileEn);
```

O bien:

```
ObjectInputStream entrada = new ObjectInputStream(new  
FileInputStream("Alumnos.obj"));
```

Para leer los objetos del archivo se hace:

```
// Suponiendo que al1 es una variable tipo Alumno previamente declarada  
al1 = (Alumno) entrada.readObject();
```

La conversión explícita al tipo de dato se puede hacer directamente si se tiene certeza del tipo de objeto almacenado. En caso contrario, se deberá confirmar el tipo antes de hacer la conversión (o usar un try catch).

Finalmente se cierra el archivo:

```
entrada.close();
```

**NOTA:** la apertura del archivo, tanto para escribir como para leer de él, debe hacerse en un *try-catch* o en un *try-with-resources*.

## C - Pruebas Unitarias

Se utilizan para probar **el correcto funcionamiento de un módulo de código** (método, clase, subprograma, programa, etc.).

Existen varios métodos para el diseño de las pruebas y la elección de los datos a usar durante las mismas. El tema de pruebas de software (testing) es uno de los temas clave en el desarrollo de software con calidad. El mismo se trata en la materia Ingeniería de Software.

A los efectos de este curso, es importante que el alumno entienda la importancia de diseñar y aplicar pruebas a todos los productos de software que desarrolla. Como consecuencia, que aplique pruebas unitarias, al menos, a los componentes críticos.

### **Pruebas Unitarias usando NetBeans**

NetBeans facilita el trabajo del ingeniero proveyendo una herramienta para crear pruebas unitarias. El funcionamiento de la misma se describe más abajo.

La ventaja de esta herramienta es que genera un método de prueba para cada uno de los métodos de la clase que se quiere probar. De esta manera, no se depende de que el programador se acuerde de generar una prueba para cada uno de ellos.

El programador queda a cargo de la implementación de cada una de las pruebas, ya que la herramienta sólo genera la estructura del método de prueba. Además, el programador debe elegir el conjunto de datos de prueba. Esta tarea es fundamental para asegurar que los métodos tienen el comportamiento esperado. Algunos criterios que se pueden aplicar para la elección de los datos de pruebas son: a) Valores dentro del rango esperado, b) Valores en los extremos del rango esperado, c) Valores fuera del rango esperado, etc.

### **Pasos para generar las pruebas**

Para generar (y ejecutar) un JUnit Test (prueba unitaria) sobre los métodos de la clase X utilizando NetBeans:

0) Una vez abierto el proyecto que incluye la clase X, resaltar el nombre de la clase X en el árbol que muestra la estructura del proyecto (en la parte superior izquierda de la pantalla de NetBeans), bajo los “Source Packages” del proyecto.

1) Apretar el botón derecho del mouse y escoger Tools → Create/Update Tests (aceptando o no las opciones por omisión que nos presenta NetBeans, en una ventana que va a aparecer).

2) Como resultado NetBeans va a crear una clase llamada XTest que va a colocar bajo los “Test Packages” del proyecto (se va a ver reflejado esto en el árbol que muestra la estructura del proyecto).

3) Dentro de la clase XTest habrá métodos vacíos llamados “setUpClass”, “tearDownClass”, “setUp” y “tearDown”. El método “setUpClass” se ejecuta una sola vez, antes de ejecutar cualquier otro método de la clase XTest. Se usa para crear las condiciones necesarias para poder ejecutar las pruebas (por ejemplo, crear una conexión con una base de datos). El método “tearDownClass” se ejecuta una sola vez luego de todas las pruebas y sirve para finalizar la ejecución de las mismas. El método “setUp” se ejecuta antes de cada método y se usa cuando se requiere inicializar variables necesarias en el método a probar. Por último, el método “tearDown” se ejecuta luego de cada método y sirve para limpiar variables. Ninguno de estos cuatro métodos es obligatorio, y sólo se tienen que incluir si existe la necesidad. NetBeans

permite, al momento de crear la clase XTest, especificar que no se desean incluir. En los ejemplos 1 y 2 (abajo) se han eliminado los cuatro. En el ejemplo 3 (abajo) se incluyen los cuatro. En el ejemplo 4 (abajo) se muestra la forma en que se pueden omitir estos cuatro métodos.

4) Aparte de los métodos mencionados en el punto anterior, también se habrá generado un método constructor para la clase XTest y un método de prueba dentro de la clase XTest para cada uno de los métodos de la clase X, con excepción de los métodos constructores de X. Con excepción del método constructor de XTest, que va a estar vacío, los otros métodos de prueba tendrán algo de código en ellos que habrá generado automáticamente NetBeans. Si un método de la clase X se llama “haceAlgo”, el método de prueba correspondiente en la clase XTest se va a llamar “testHaceAlgo”. Si el método “haceAlgo” de la clase X regresa un valor real (*double*), el contenido del método de prueba “testHaceAlgo” va a ser como el siguiente:

```
public void testHaceAlgo() {
    System.out.println("haceAlgo");
    X instance = new X();
    double expResult = 0.0;
    double result = instance.haceAlgo();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}
```

La idea es que este método cree una instancia de la clase X (almacenada en la variable “instance”), defina (en la variable “expResult”) qué valor se esperaría que regresara el método “haceAlgo” si se aplica el método a la instancia almacenada en “instance”, ejecute dicho método sobre dicha instancia, y compare el valor obtenido de la ejecución con el valor esperado (utilizando “assertEquals”). La llamada al método “fail” va a hacer que la prueba falle automáticamente, así es que hay que quitarla por completo antes de ejecutar la prueba (o ponerla como comentario).

Suponiendo que un constructor de la clase X reciba tres parámetros (dos números enteros y un número real), que decidamos crear una instancia de X cuyos tres atributos tomen los valores 2, 6 y 7.0, y que el valor que debería regresar el método “haceAlgo” para dicha combinación de valores en los atributos sea  $\pi(7.0)^2$ , tendríamos que realizar algunos cambios al método “testHaceAlgo” de tal forma que quede así:

```
public void testHaceAlgo() {
    double r = 7.0;
    System.out.println("Evalúa el método haceAlgo");
    X instance = new X(2,6,r);
    double expResult = Math.PI * Math.pow(r,2);
    double result = instance.haceAlgo();
    assertEquals(expResult, result,0.001);
    // fail("The test case is a prototype.");
}
```

El método “assertEquals”, si se utiliza para comparar valores reales, como en este caso, necesita un tercer argumento que represente el margen de error tolerado.

Después de desarrollar todos los métodos de prueba hay que ejecutarlos. Para ello se debe colocar en el árbol que muestra la estructura del proyecto (en la parte superior izquierda de la pantalla de NetBeans), bajo los “Test Packages” del proyecto, sobre el nombre de XTest y, apretando el botón derecho del mouse, escoger la opción “Run File” (lo cual se puede hacer a

pesar de que no exista un método “main” en la clase XTest). En la ventana de salida (normalmente en la parte inferior derecha de la ventana de NetBeans) debe aparecer el resultado de ejecutar las pruebas (un resultado para cada uno de los métodos de prueba que contengan código, y un resultado global que es la conjunción de los resultados individuales).

**Variantes del assert:** assertEquals, assertFalse/True, assertNotNull, assertEquals, assertEquals...<sup>3</sup>

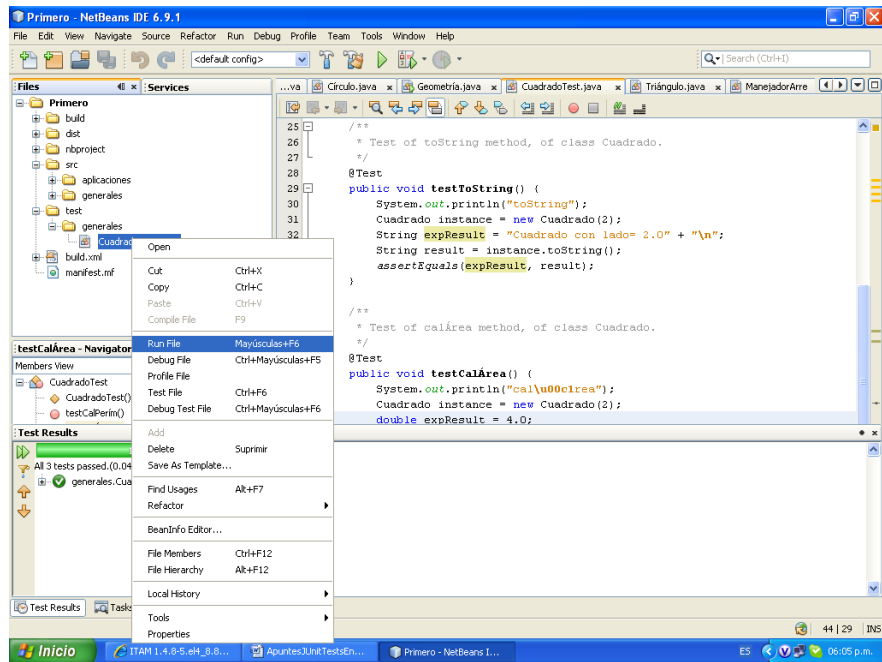
**Ejemplo 1:** Para mayor claridad se quitaron todos los métodos que no se usarán. Se crea una instancia de la clase Cuadrado para cada uno de los métodos que se desea probar.

Clase a probar	Prueba para la clase Cuadrado
<pre> public class Cuadrado {     private double lado;      public Cuadrado() {     }     public Cuadrado(double lado) {         this.lado = lado;     }      @Override     public String toString() {         return "Cuadrado con lado= " + lado + "\n";     }      public double calArea(){         return lado * lado;     }      public double calPerim(){         return lado * 4;     } } </pre>	<pre> public class CuadradoTest {     public CuadradoTest() {     }      /**      * Test of toString method, of class Cuadrado.      */     @Test     public void testToString() {         System.out.println("toString");         Cuadrado instance = new Cuadrado(2);         String expectedResult = "Cuadrado con lado= 2.0" + "\n";         String result = instance.toString();         assertEquals(expResult, result);     }      /**      * Test of calArea method, of class Cuadrado.      */     @Test     public void testCalArea() {         System.out.println("calArea");         Cuadrado instance = new Cuadrado(2);         double expectedResult = 4.0;         double result = instance.calArea();         assertEquals(expResult, result, 0.0);     }      /**      * Test of calPerim method, of class Cuadrado.      */     @Test     public void testCalPerim() {         System.out.println("calPerim");         Cuadrado instance = new Cuadrado(2);         double expectedResult = 8.0;         double result = instance.calPerim();     } } </pre>

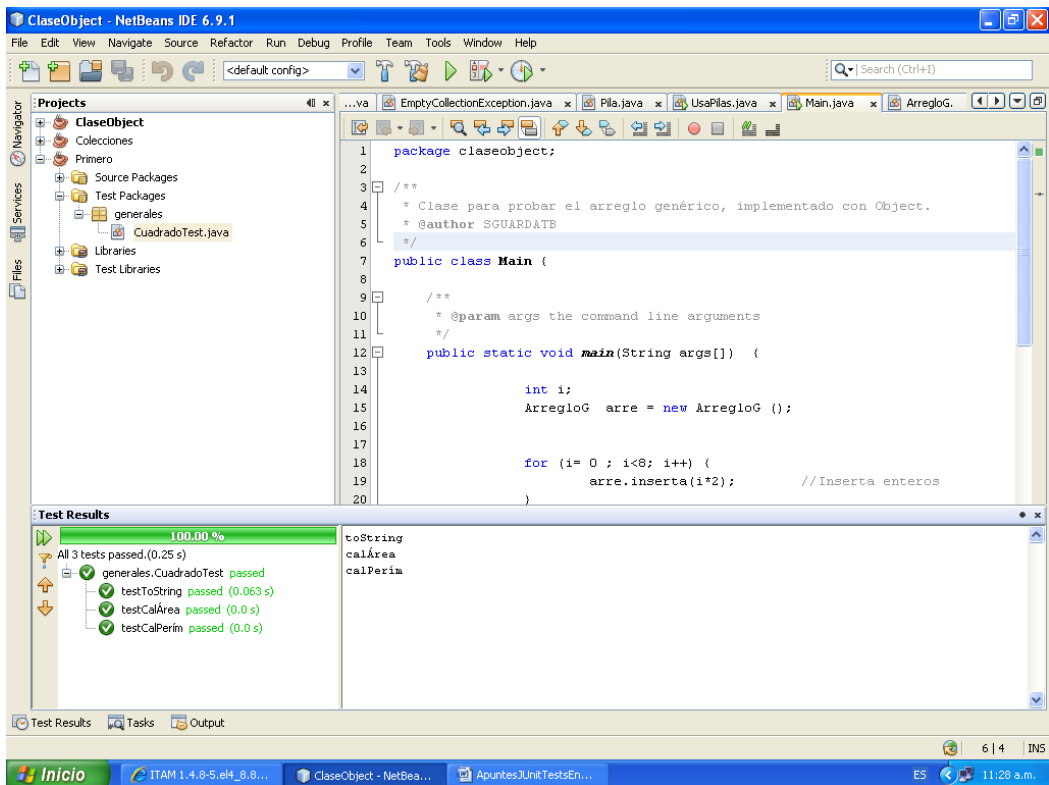
<sup>3</sup> Se sugiere revisar la documentación del assert en: <http://junit.org/junit4/javadoc/latest/org/junit/Assert.html>

	<pre> assertEquals(expResult, result, 0.0);     } } </pre>
--	--

1) Se selecciona Run File sobre el archivo CuadradoTest



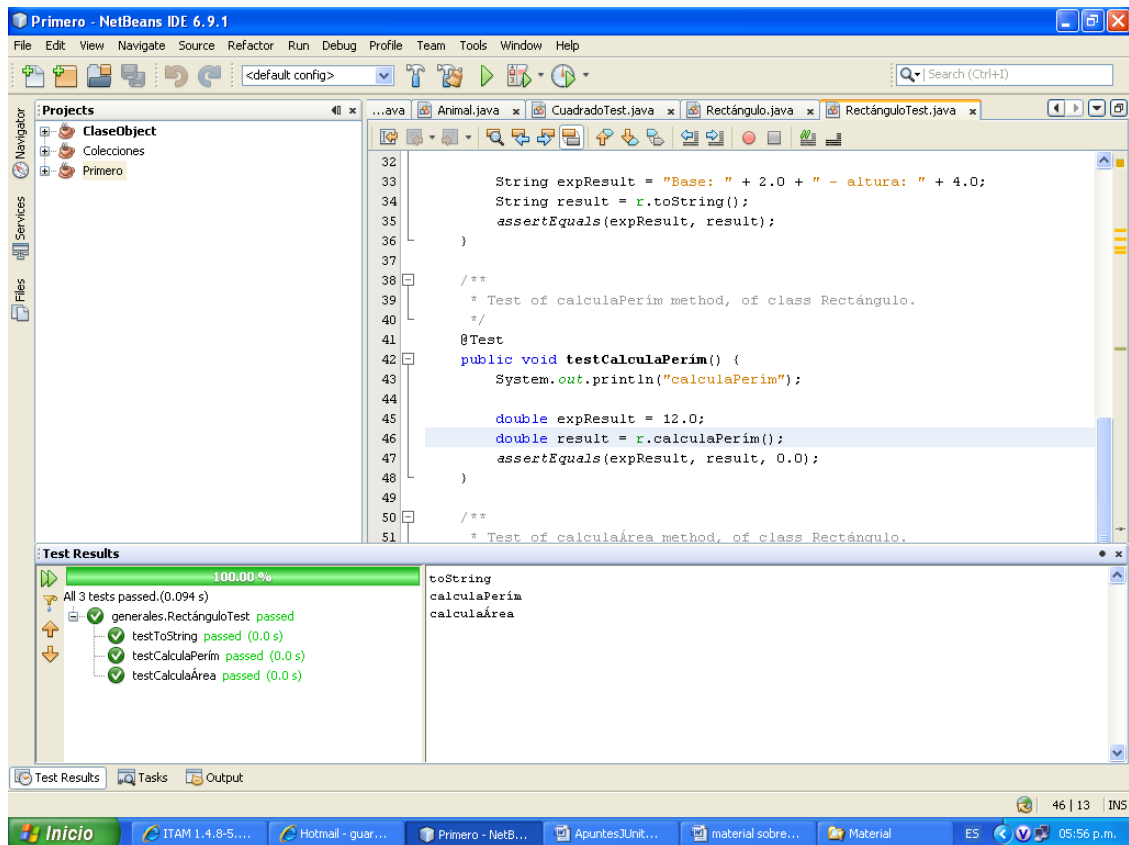
2) Luego de ejecutarse se despliega la siguiente pantalla:





**Ejemplo 2:** Para mayor claridad se quitaron todos los métodos que no se usarán. Se define un atributo de tipo Rectángulo y se crea una instancia en el constructor. Todos los métodos prueban sobre este objeto.

Clase a probar	Prueba para la clase Rectángulo
<pre> public class Rectángulo {     private double l1, l2;      public Rectángulo() { }      public Rectángulo(double b, double b) {         l1 = b;         l2 = a;     }      public String toString(){         return "Base: " + l1 + " - altura: " + l2;     }      public double calculaPerím() {         return 2 * (l1 + l2);     }      public double calculaÁrea() {         return l1 * l2;     } } </pre>	<pre> public class RectánguloTest {     private Rectángulo r;      public RectánguloTest() {         r = new Rectángulo(2,4);     }     /**      * Test of toString method, of class      Rectángulo.      */     @Test     public void testToString() {         System.out.println("toString");          String expectedResult = "Base: " + 2.0 + " -         altura: " + 4.0;         String result = r.toString();         assertEquals(expectedResult, result);     }     /**      * Test of calculaPerím method, of class      Rectángulo.      */     @Test     public void testCalculaPerím() {         System.out.println("calculaPerím");          double expectedResult = 12.0;         double result = r.calculaPerím();         assertEquals(expectedResult, result, 0.0);     }     /**      * Test of calculaÁrea method, of class      Rectángulo.      */     @Test     public void testCalculaÁrea() {         System.out.println("calculaÁrea");          double expectedResult = 8.0;         double result = r.calculaÁrea();         assertEquals(expectedResult, result, 0.0);     } } </pre>



**Ejemplo 3:** Se selecciona crear un test para la clase *Círculo*. Se agregan impresiones en los métodos inicializadores y finalizadores para mostrar cuando se ejecuta cada uno de ellos.

### Clase generada:

```
public class CírculoTest {

    public CírculoTest() {
    }

    @BeforeClass
    public static void setUpClass() throws Exception {
        System.out.println("--Antes de testear la clase Círculo");
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
        System.out.println("--Después de testear la clase Círculo");
    }

    @Before
    public void setUp() {
        System.out.println("\n**Antes de testear un método");
    }

    @After
    public void tearDown() {
    }
}
```

```

        System.out.println("**Después de testear un método");
    }

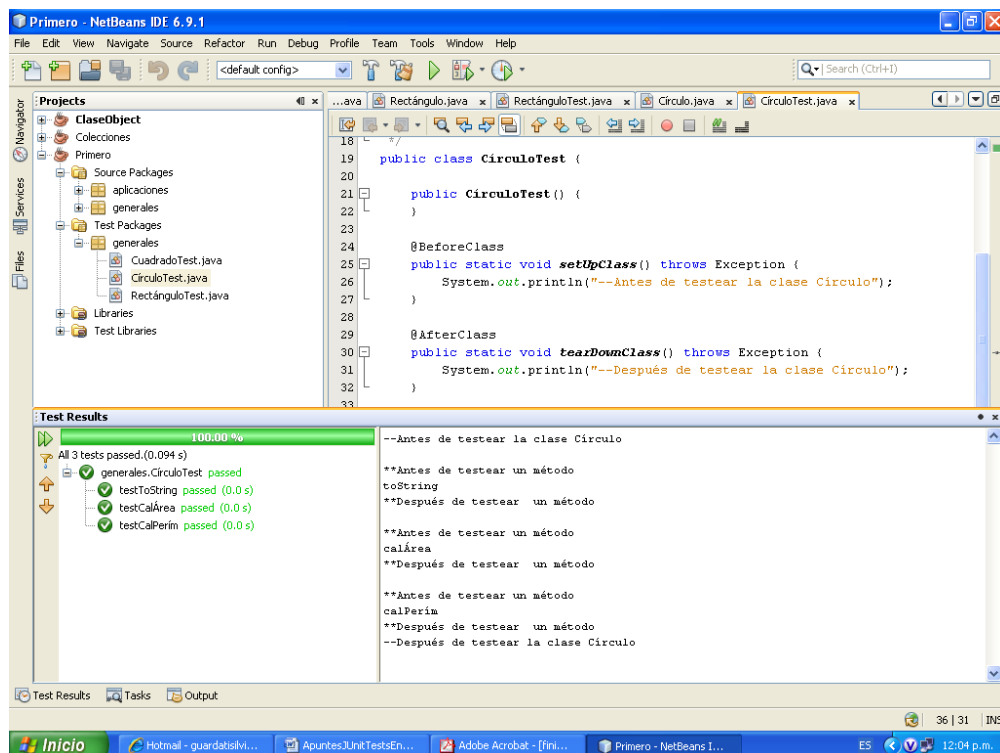
    /**
     * Test of toString method, of class Círculo.
     */
    @Test
    public void testToString() {
        System.out.println("toString");
        Círculo instance = new Círculo(8.5);
        String expectedResult = "Círculo de radio= " + 8.5 + "\n";
        String result = instance.toString();
        assertEquals(expResult, result);
    }

    /**
     * Test of calÁrea method, of class Círculo.
     */
    @Test
    public void testCalÁrea() {
        System.out.println("calArea");
        Círculo instance = new Círculo(5.3);
        double expectedResult = Math.PI * Math.pow(5.3, 2);
        double result = instance.calÁrea();
        assertEquals(expResult, result, 0.0);
    }

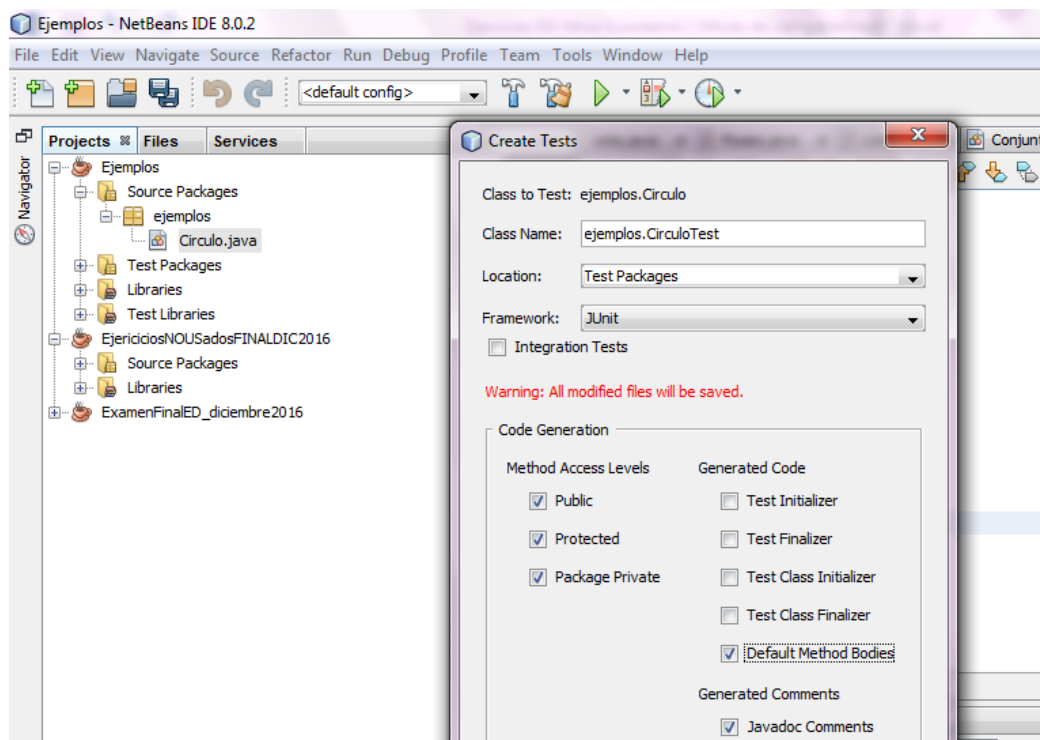
    /**
     * Test of calPerím method, of class Círculo.
     */
    @Test
    public void testCalPerím() {
        System.out.println("calPerím");
        Círculo instance = new Círculo(5.3);
        double expectedResult = 2 * 5.3 * Math.PI;
        double result = instance.calPerím();
        assertEquals(expResult, result, 0.0);
    }
}

```

**Luego de ejecutarla:**



**Ejemplo 4:** Se selecciona crear una prueba para la clase `Circulo` en la cual se omiten los cuatro métodos que se incluyen por omisión (`setUpClass()`, `tearDownClass()`, `setUp()` y `tearDown()`). Para lograrlo se deselecciona en la ventana correspondiente las marcas junto a los nombres de los métodos, tal como se muestra en la siguiente figura.



**La clase resultante es:**

```
public class CírculoTest {

    public CírculoTest() {
    }

    /**
     * Test of toString method, of class Círculo.
     */
    @Test
    public void testToString() {
        System.out.println("toString");
        Círculo instance = new Círculo(8.5);
        String expectedResult = "Círculo de radio= " + 8.5 + "\n";
        String result = instance.toString();
        assertEquals(expResult, result);
    }

    /**
     * Test of calÁrea method, of class Círculo.
     */
    @Test
    public void testCalÁrea() {
        System.out.println("calArea");
        Círculo instance = new Círculo(5.3);
        double expectedResult = Math.PI * Math.pow(5.3, 2);
        double result = instance.calÁrea();
        assertEquals(expResult, result, 0.0);
    }

    /**
     * Test of calPerím method, of class Círculo.
     */
    @Test
    public void testCalPerím() {
        System.out.println("calPerim");
        Círculo instance = new Círculo(5.3);
        double expectedResult = 2 * 5.3 * Math.PI;
        double result = instance.calPerím();
        assertEquals(expResult, result, 0.0);
    }
}
```

## D – Inclusión de clases definidas en otros proyectos

Cuando en un proyecto se necesita usar clases definidas previamente en otro proyecto, estas se pueden incluir de diversas maneras. A continuación, se presenta una de ellas.

- 1) dar click en el botón derecho del mouse sobre el proyecto (lado izquierdo de la ventana) en el cual se quiere hacer la inclusión
- 2) Elegir **Properties**
- 3) En la ventana de **Categories** elegir **Libraries**
- 4) Dar click en el botón **Add Project**
- 5) Elegir el proyecto al cual pertenecen las clases que interesan
- 6) Dar click en el botón **Add Project JAR Files**
- 7) Dar click en el botón **OK**

Una vez ejecutados estos pasos, se podrán incluir aquellas clases que deseen usarse en el proyecto actual.