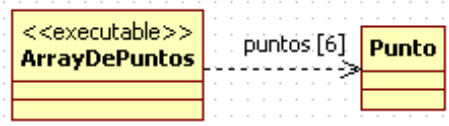
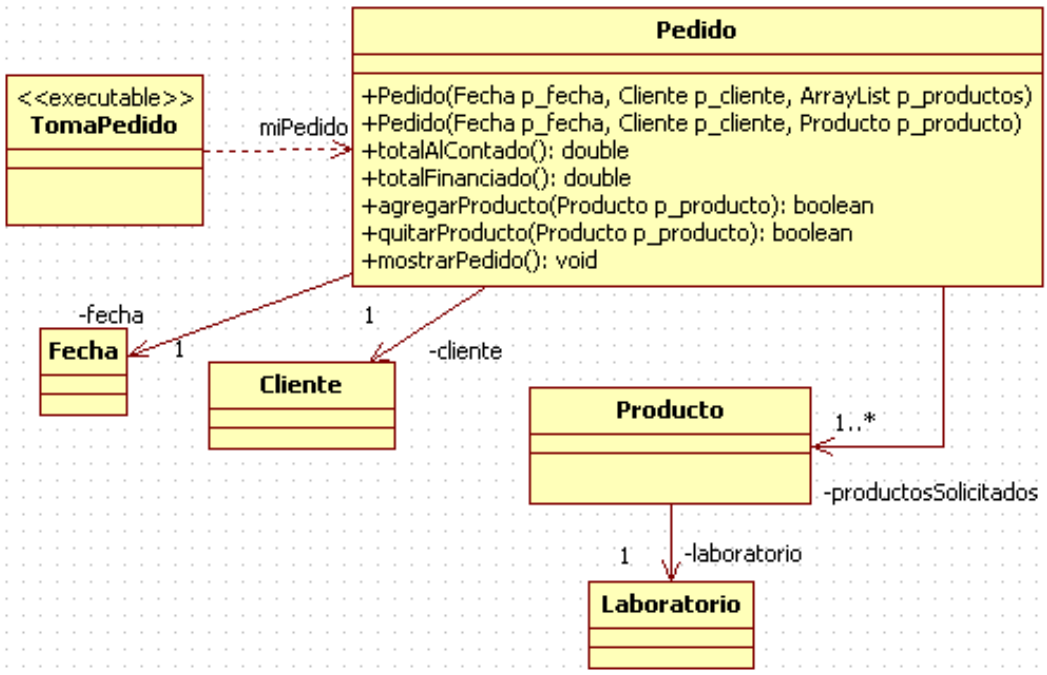
Objetivo: Que el alumno:
- Aprecie la ventaja de la reutilización del código, a través de la instanciación de **clases predefinidas de Java** (colecciones)
- Aprenda a manejar (agregar, eliminar y recuperar un elemento) colecciones de objetos de una clase
  - Colecciones estáticas/dinámicas
  - Colecciones homogéneas/heterogéneas
  - Colecciones indexadas

Nota:Resolver algunos ejercicios con genéricos y otros utilizando cast al recuperar elementos
En todos los casos, reutilizar las clases existentes

1. Crear una clase ejecutable llamada **ArrayDePuntos**, donde se haga lo siguiente:
   1.1. Crear un contenedor estático de seis elementos de tipo Punto, llamado **puntos**.
   1.2. Agregar como elementos instancias de la clase **Punto**. Ingresar los datos por teclado (Scanner ó Buffered Reader).
   1.3. Recorrer el array e imprimir las coordenadas de cada elemento, utilizando el método previsto en la clase Punto.
   1.4. Imprimir en pantalla la distancia cada 2 elementos consecutivos del contenedor (Ej: distancia pto1-pto2, distancia pto2-pto3, etc)



2. Una empresa desea implementar una aplicación para automatizar la toma de pedidos de sus clientes. El modelo de la aplicación es el siguiente:



   2.1. La clase Pedido es la responsable de mantener la lista de productos solicitados (productosSolicitados) por lo tanto debe proveer una interfaz que contemple el agregar o quitar un producto de la lista. Tenga en cuenta que esta colección presenta la característica de que cada elemento se agrega al final (no se agregan productos en el medio de la lista) y mantienen el orden en que se van agregando, por tal motivo se sugiere usar una clase ArrayList (que implementa la interface List y reúne el comportamiento de una lista). En este caso, si un producto se pide en más de una unidad, se lo agrega la cantidad de veces solicitada a la lista. Se trabaja por unidad.

   2.2. El método mostrarPedido() emite un detalle de los productos solicitados, que incluye precio de lista y de contado, y al final, un total para cada precio, con el siguiente formato:

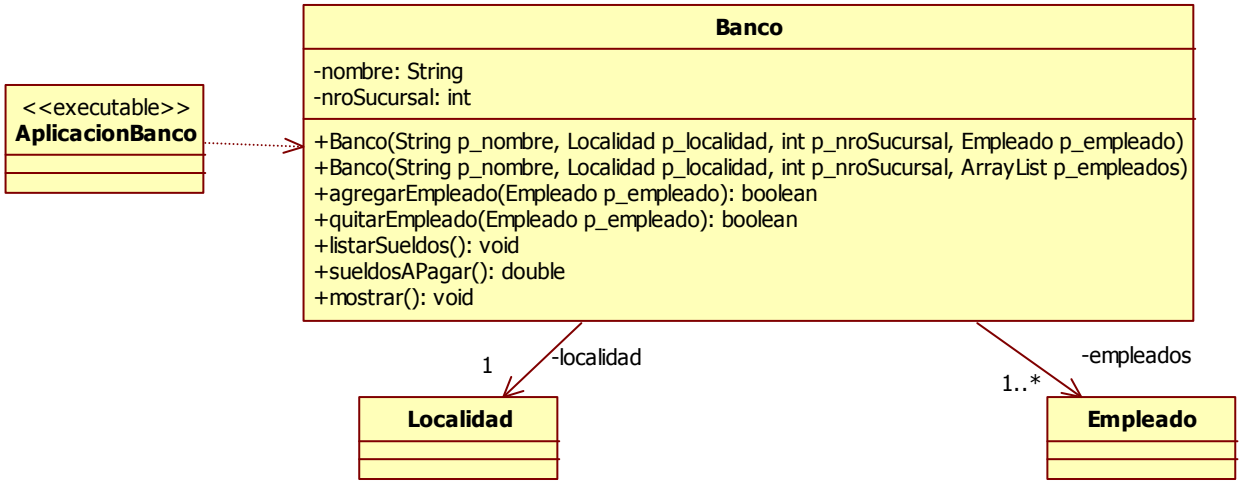
```
****** Detalle del pedido ****** Fecha: 14 de agosto de 2012.

  Producto        Precio Lista   Precio Contado
-----------------------------------------------------

CD-musica variada     35.28        33.516
Libro-POO            126.00       119.7
Revista-user           7.28         6.916
-----------------------------------------------------
****  Total ---------- 168.56       160.132
```

2.3. El método totalAlContado devuelve el resultado de calcular la suma del precio de contado de cada elemento de la lista de productos solicitados.

2.4. El método totalFinanciado devuelve el resultado de calcular la suma del precio de lista de cada elemento de la lista de productos solicitados.

2.5. En la clase TomaPedido:
- 2.5.1. Crear un cliente
- 2.5.2. Crear un producto
- 2.5.3. Crear una instancia de Pedido, que representa el pedido para el cliente creado anteriormente, y que se inicia con el producto instanciado
- 2.5.4. Crear varios productos y agregarlos al pedido del cliente (lista de productos solicitados)
- 2.5.5. Imprimir el total al contado y el total financiado del pedido
- 2.5.6. Quitar un producto de la lista
- 2.5.7. Imprimir nuevamente los totales
- 2.5.8. Emitir un detalle de los productos solicitados, con totales financiado y al contado

3. Una entidad bancaria modela una aplicación para liquidar los sueldos de sus empleados, según se observa en el diagrama de clases. Se observa que el banco debe contar con **al menos 1** empleado al momento de crearse **(1..*).** La clase Banco debe implementar los siguientes métodos:
   - 3.1. mostrar() imprime el nombre, el número de sucursal y la localidad del banco.
   - 3.2. listarSueldos(), imprime los empleados con sus respectivos sueldos (reutilizar método apropiado)
   - 3.3. sueldosAPagar(), devuelve el monto total a abonar en concepto de sueldos
   - 3.4. métodos necesarios para agregar y quitar empleados de la colección



Se desea obtener la siguiente salida impresa:

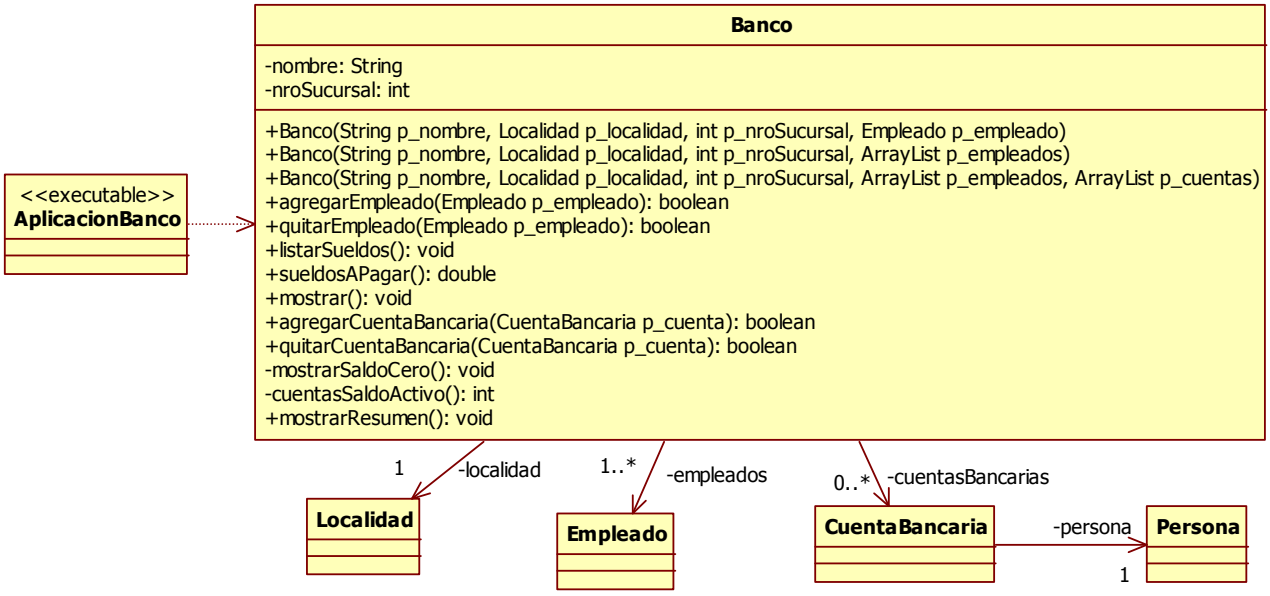
Banco**: RIO** Sucursal**: 3**
Localidad**: Corrientes** Provincia**: Corrientes**

27267504235 Perez, Lorena, ----------$1212.0
20159462638 Dominguez, Pedro ----------$2650.4

Total a Pagar----------$3862.4

En la clase AplicacionBanco crear los objetos necesarios para realizar una prueba de la implementación.

4. La misma entidad bancaria desea agregar funcionalidades a su aplicación. En este caso necesita implementar el comportamiento adecuado para manipular sus cuentas bancarias, reutilizando clases definidas previamente, y agregando métodos necesarios.
   - 4.1. Se asume que el banco realiza variadas operaciones, por lo que **no necesariamente debe poseer** una cuenta bancaria al momento de crearse **(0..*),** mientras que la cuenta bancaria requiere de un titular para su existencia **(1).**
   - 4.2. El método mostrarSaldoCero() lista los titulares de todas las cuentas cuyos saldos estén en 0 (cero).
   - 4.3. El método cuentasSaldoActivo() especifica la cantidad de cuentas cuyo saldo sea mayor a 0.
   - 4.4. Definir los métodos para agregar y quitar elementos a la colección de cuentas.



4.5. El método mostrarResumen() imprime la siguiente salida:

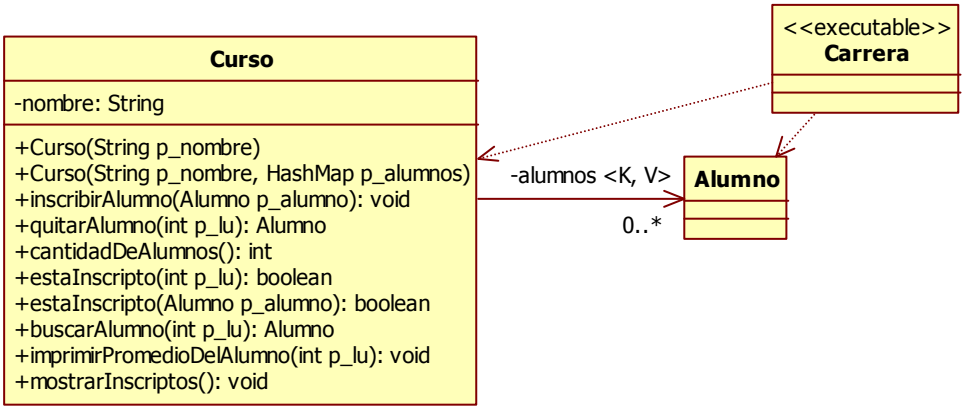
Banco: Rio - Sucursal: 3
Localidad: Saladas Provincia: Corrientes
****
RESUMEN DE CUENTAS BANCARIAS
****
Número total de Cuentas Bancarias: 2510
Cuentas Activas: 2352
Cuentas Saldo Cero: 158.

Titulares con Cuenta en Saldo Cero

14526387 Gomez, Marisa Esther
23145698 Villalba, Martín

Nota: En la clase AplicacionBanco agregar las instrucciones necesarias a fin de probar las nuevas funcionalidades. Si lo desea, puede adicionar un menú que permita seleccionar la manipulación de empleados ó de las cuentas.

5. Una institución educativa desea administrar información de los cursos que dicta. El siguiente diagrama de clases modela el diseño de la situación



Al ejecutar la aplicación (clase Carrera), se desea obtener la siguiente salida:

****-- Cantidad de inscriptos: 4
32555 Pedro Gomez
23564 Maria Vasquez
30123 Juan Perez
32655 Marcela Martinez

****-- Se da de baja a Pedro porque abandona el curso --****

Está Pedro Gomez inscripto ?? --> false

****-- Alumnos inscriptos actualmente: 3
23564 Maria Vasquez
30123 Juan Perez
32655 Marcela Martinez

****-- Busca y muestra el alumno con numero de libreta 30123 --****
Alumno -
Apellido y nombre: Juan Perez
LU: 30123 Notas: 7.0, 9.0
Promedio: 8.0 Aprobado

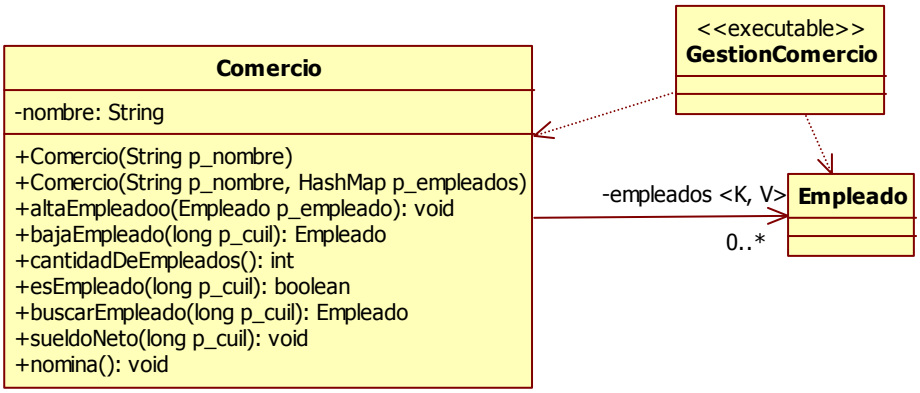
****-- Mostrar promedio del alumno 23564 --****
Promedio: 5.5

5.1. La clase Curso es la responsable de mantener el registro de los alumnos inscriptos en el curso (colección alumnos), por lo tanto debe proveer una interface que contemple el inscribir y el quitar un alumno del curso. Tenga en cuenta que en este caso la colección debe presenta la característica de que cada alumno (valor) de la misma puede ser recuperado mediante el número de LU (clave), por tal motivo se sugiere usar una clase HashMap (implementa la interface Map, reúne el comportamiento de un dictionary, trabajando con pares de clave-valor)
5.2. El método cantidadDeAlumnos() devuelve el valor que representa el total de alumnos que están inscriptos en el curso
5.3. El método estaInscripto() está sobrecargado, dado que puede recibir como parámetro la LU del alumno, o el objeto alumno. En ambos casos retorna **true** si el alumno pasado como parámetro se encuentra anotado en el curso y **false** en caso contrario.
5.4. El método buscarAlumno() retorna el alumno cuya libreta coincida con el parámetro introducido
5.5. El método imprimirPromedioDelAlumno() muestra por pantalla el promedio de notas del alumno cuya libreta coincida con el parámetro introducido
5.6. El método mostrarInscriptos() recorre la colección de inscriptos y muestra libreta y nombre.
5.7. En la clase ejecutable se debe:
  5.7.1. Crear una instancia de Curso y varias de la clase Alumno
  5.7.2. Asignarles notas de parciales a los alumnos
  5.7.3. Inscribir los alumnos al curso creado anteriormente.
  5.7.4. Imprimir la cantidad y la lista de alumnos inscriptos al curso
  5.7.5. Dar de baja un alumno del curso, y luego verificar que no esté inscripto
  5.7.6. Imprimir nuevamente la lista de alumnos para ver como que queda definitivamente y la cantidad total de alumnos inscriptos en el curso
  5.7.7. Buscar un alumno por su libreta. Una vez encontrado, mostrarlo con el método provisto por la clase Alumno.
  5.7.8. Mostrar el promedio del alumno solicitado, según libreta

6 Un Comercio desea automatizar el registro de sus empleados (colección empleados), y para ello se debe proveer una interface que contemple el alta y la baja de los mismos. Tenga en cuenta que en este caso la colección presenta la característica de que cada empleado (valor) puede ser recuperado mediante el número de cuil (clave), por lo que resulta conveniente usar una clase HashMap. Para gestionar los RRHH, es necesario contar con métodos que permitan conocer la cantidad de empleados, consultar por medio del cuil si un empleado es parte de la empresa, buscar un empleado determinado, también mediante el cuil, visualizar por pantalla el sueldo neto del empleado cuyo cuil coincida con el parámetro introducido, y por último, emitir una nómina de los empleados, que debe ser presentado ante la AFIP, con el siguiente formato:

**** Nomina de empleados de Impulso ****
30100623 Gonzalez, Juan---------- $1027.58
37045987 Martinez, Mercedes------ $1007.19
32550096 Gomez, Virginia---------- $1501.20

Implementar una clase ejecutable GestionComercio, donde se instancie un comercio y varios empleados, que serán contratados por el comercio. Emitir una nómina, y verificar el correcto funcionamiento de las demás funcionalidades implementadas.


Comercio
-nombre: String
+Comercio(String p_nombre)
+Comercio(String p_nombre, HashMap p_empleados)
+altaEmpleadoo(Empleado p_empleado): void
+bajaEmpleado(long p_cuil): Empleado
+cantidadDeEmpleados(): int
+esEmpleado(long p_cuil): boolean
+buscarEmpleado(long p_cuil): Empleado
+sueldoNeto(long p_cuil): void
+nomina(): void
<<executable>>
GestionComercio
-empleados <K, V>
0..*
Empleado