Práctica 1

Guía de TDA

1.1. TDAs simples

Ejercicio 1.1. Implementar la clase Nota para cumplir con la siguiente interfaz:

```
class Nota {
    /* pre : valorInicial está comprendido entre 1 y 10.
    * post: inicializa la Nota con el valor indicado.
    */
    Nota(int valorInicial);

    /* post: devuelve el valor numérico de la Nota,
    */
    int obtenerValor();

    /* post: indica si la Nota permite o no la aprobación.
    */
    boolean aprobado();

    /* post: indica si la Nota implica la desaprobación. */
    boolean desaprobado();
};
```

Ejercicio 1.2. Agregar a la clase Nota el método:

```
    /* pre : nuevoValor está comprendido entre 1 y 10.
    * post: modifica el valor numérico de la Nota, cambiándolo
    *   por nuevoValor, si nuevoValor es superior al
    *   valor numérico actual de la Nota.
    */
    void recuperar(int nuevoValor);
```

Ejercicio 1.3. Implementar la clase Punto. Un Punto en el plano posee coordenada X y coordenada Y. Proporcionar métodos para:

- consultar las coordenadas
- cambiar las coordenadas
- saber si el punto está sobre el eje de las X
- saber si el punto está sobre el eje de las Y
- saber si el punto es el origen de coordenadas.
- Indicar su distancia al origen

Ejercicio 1.4. Implementar la clase Rectangulo, con atributos base y altura y los métodos para modificar y obtener sus valores. Además, deben implementar métodos para obtener el perímetro y el área.

Ejercicio 1.5. Implementar la clase Cubo a partir de la siguiente interfaz:

```
class Cubo {
    /* pre: lado es un flotante positivo
    * post: inicializa el cubo a partir de la medida de lado dada
    */
    Cubo (float lado);

    /* post: devuelve la longitud de todos los lados del cubo
    */
    int obtenerLongitudLado();

    /* pre: lado es un valor mayor a 0.
    * post: cambia la longitud de todos los lados del cubo
    */
    void cambiarLongitudLado(float lado);

    /* post: devuelve la superficie ocupada por las caras del cubo
    */
    int obtenerSuperficieCubo();

    /* pre: superficieCubo es un valor mayor a 0.
    * post: cambia la superficie de las caras del cubo
    */
    void cambiarSuperficieCubo(float superficieCubo);

    /* post: devuelve el volumen que encierra el cubo
    */
    float obtenerVolumen();

    /* pre: volumen es un valor mayor a 0.
    * post: cambia el volumen del cubo
    */
```

```
    void cambiarVolumen(float volumen);
};
```

Ejercicio 1.6. Implementar la clase TarjetaBaja a partir de la siguiente declaración:

```
class TarjetaBaja {
    /* pre: saldoInicial es positivo.
    * post: saldo de la Tarjeta en saldoInicial.
    */
    TarjetaBaja(double saldoInicial);

    double obtenerSaldo()

    /* pre: monto es positivo
    * post: agrega el monto al saldo de la Tarjeta.
    */
    void cargar(double monto);

    /* pre : saldo suficiente, seccion es 1, 2 o 3.
    * post: utiliza del saldo $12, $13 o $13.75 segun la seccion
    * para un viaje en colectivo.
    */
    void pagarViajeEnColectivo(int seccion);

    /* pre : saldo suficiente.
    * post: utiliza $12.50 del saldo para un viaje en subte.
    */
    void pagarViajeEnSubte();

    /* post: devuelve la cantidad total de viajes realizados.
    */
    int contarViajes();

    /* post: devuelve la cantidad de viajes en colectivo.
    */
    int contarViajesEnColectivo();

    /* post: devuelve la cantidad de viajes en subte.
    */
    int contarViajesEnSubte();
};
```

Ejercicio 1.7. Implementar una clase que modele un Circulo, del que se desa conocer: radio, diámetro, perímetro y superficie.

Ejercicio 1.8. Implementar una clase que modele un Disco.

Gráfico 1.1.1: disco

Se desea conocer:

- radio interior
- radio exterior
- perímetro interior
- perímetro exterior
- superficie.

Debe tener operaciones para cambiar

- el radio interior y
- el radio exterior.

Ejercicio 1.9. Implementar la clase Ticket a partir de la siguiente interfaz

```
class Ticket {
    /* post: el Ticket se inicializa con importe 0.
    */
    Ticket();

    /* pre : cantidad y precio son mayores a cero. El ticket está abierto.
    * post: suma al Ticket un item a partir de la cantidad de
    * de productos y su precio unitario.
    */
    void agregarItem(int cantidad, double precioUnitario);

    /* pre : el Ticket está abierto y no se ha aplicado un descuento previamente.
    * post: aplica un descuento sobre el total del importe.
    */
    void aplicarDescuento(double porcentaje);

    /* post: devuelve el importe acumulado hasta el momento sin cerrar el Ticket.
    */
    double calcularSubtotal();

    /* post: cierra el Ticket y devuelve el importe total.
    */
    double calcularTotal();

    /* post: devuelve la cantidad total de productos.
    */
    int contarProductos();
};
```

Ejercicio 1.10. Implementar la clase CajaDeAhorro con la siguiente interfaz:

```
class CajaDeAhorro {
    /* pre: titularDeLaCuenta no es vacio
    * post: la instancia queda asignada al titular indicado
    * y con saldo igual a 0.
    */
    CajaDeAhorro(string titularDeLaCuenta);

    /* post: devuelve el nombre del titular de la Caja de Ahorro.
    */
    string obtenerTitular();

    /* post: devuelve el saldo de la Caja de Ahorro.
    */
    double consultarSaldo();

    /* pre : monto es un valor mayor a 0.
    * post: aumenta el saldo de la Caja de Ahorro según el monto depositado.
    */
    void depositar(double monto);

    /* pre : monto es un valor mayor a 0 y menor o igual que el saldo de la
    * Caja de Ahorro.
    * post: disminuye el saldo de la Caja de Ahorro según el monto
    * extraido.
    */
    void extraer(double monto);
};
```

1.2. TDAs compuestos

Ejercicio 1.11. Diseñar e implementar la clase Remis. Un Remis tiene

- Marca
- Patente
- Km inicial
- Km final
- Chofer
- A lo sumo 3 pasajeros

Tanto el chofer como los pasajeros tienen:

- DNI

- Nombre

Además de conocer los datos como marca, patente, etc. Debe

- Conocer la cantidad de pasajeros que viajan
- Indicar el nombre del pasajero más grande (menor DNI como aproximación)
- Indicar los Km recorridos

Ejercicio 1.12. Escribir un programa principal en donde se utilice un vector de 5 objetos Remis, implementado en el ejercicio anterior.

Ejercicio 1.13. Un Tragamonedas está compuesto por tres tambores. Cada Tambor, cuando gira, se detiene en un número entre 1 y 8 inclusive. El Tragamonedas otorga un premio cuando los tres tambores se detienen en la misma posición. Modelar ambas clases:

- Interfaz de Tambor
  - girar (combinando la función rand, de la librería cstdlib, con módulo se consigue un número aleatorio en el rango que se necesite)
  - obtenerPosicion
- Interfaz de Tragamonedas
  - determinarPremio (un entero positivo)
  - sortear (gira los tres tambores)
  - otorgarPremio (booleana)

Ejercicio 1.14. Implementar la clase Alarma y la clase Sensor con la siguiente interfaz.

Una Alarma cuenta con un Sensor de movimiento, un Sensor de contacto y un Sensor de sonido.

```
class Sensor {
    /* * post: sensor apagado.
    */
    Sensor();

    /* * post: enciende el sensor.
    */
    void encender();
```

```
    /* * post: apaga el sensor.
    */
    void apagar();

    /* * post: devuelve si el sensor ha sido activado a
        causa de algun evento.
    */
    boolean activado();

    /* * post: activa el sensor.
    */
    void activar();
};

class Alarma {

    /*  pre: codigoSeguridad es un entero positivo
        * post: alarma apagada con el código de seguridad indicado.
    */
    Alarma(int codigoSeguridad);

    /* * post: enciende la alarma.
    */
    void encender();

    /* * post: si codigoSeguridad es correcto, apaga la alarma.
    */
    void apagar(int codigoSeguridad);

    /* * post: devuelve si alguno de los sensores está activado.
    */
    boolean activada();
};
```

Ejercicio 1.15. Diseñar e implementar la clase TarjetaViaje. Debe tener operaciones para:

- Conocer la cantidad de viajes realizados (máximo 50).
- Indicar si hubo algún viaje en determinada línea de colectivo.
- Indicar si hubo algún viaje en determinada Fecha.

La clase TarjetaViaje debe tener por lo menos:

- Indicar la Fecha del viaje.
- Indicar la línea de colectivo.

La clase Fecha tiene día, mes y anio.

Ejercicio 1.16. Para cada una de las clases construidas en todos los ejercicios, implementar un *main* que las utilice.

1.3. Herencia y polimorfismo

Ejercicio 1.17. Nos solicitan modelar como parte del sistema de RRHH de una empresa, la nómina de los distintos tipos de empleados que trabajan en la empresa y sus características. Los posibles tipos de empleados a considerar son: *Director*, *Jefe de área* y *Administrativo*. (Hay que tener presente que un director es un empleado, pero no a la inversa). Construir una estructura de empleados que permita guardar la siguiente información de cada uno: legajo, nombre y apellido, salario y antigüedad. Debemos proveer métodos que permitan visualizar las características de cada empleado. Además debemos proveer de un método incrementarSalario() que se ejecutará anualmente y permitirá incrementar el salario de toda la planta de empleados. Para esto debe considerarse que a los directores se les incrementará un 10 %, a los jefes un 15 % y a los empleados administrativos un 20 % anual.

Ejercicio 1.18. Escribir un main que genere en forma aleatoria un vector de 100 Empleados, teniendo en cuenta la implementación del punto anterior. Para eso se debe "sortear" de forma random un número entre 0 y 1, tomando un 10 % para los directores, un 20 % para los jefes y el resto serán administrativos. Por ejemplo:

- De 0 a 69 es Administrativo
- De 70 a 89 es Jefe
- De 90 a 100 es Director

Además debe sortear un nombre y un apellido de una lista de nombres y apellidos, una antiguedad y un salario entre un rango de 3 salarios: uno para directores, otro para jefes y otro para administrativos.

Luego, debe listar los datos y los sueldos de cada uno y calcular un promedio.

Ejercicio 1.19. Nos piden diseñar un sistema que permita calcular el cobro de un estacionamiento. Los vehículos a considerar tienen las siguientes características: Peso, Potencia y Cilindrada. Desarrolle un sistema que permita administrar un total de 50 cocheras, en las cuales podemos guardar tres tipos

de vehículos: autos, camiones y motocicletas. Además debe calcular el costo del estacionamiento de cada vehículo, considerando las siguientes reglas y propiedades:

- Coches: Nº puertas.
  - Monovolúmenes: Se les cobra un 1 % respecto al producto de peso y numero de puertas que tenga.
  - Deportivos: Se les cobra 2 % de su cilindrada.
- Camiones: Tara, Nº ejes.
  - Rígidos: Se les cobra un 2,5 % respecto de su Tara
  - Con remolque: Se les cobra 2,5 % respecto del producto de Tara por Peso.
- Motocicletas.
  - Con matrícula: Se les cobra 0.5 % de su potencia.
  - Sin matrícula: Se les cobra un 0,5 % de su cilindrada.

Generar un vector de 50 de estos vehículos, el cual se generará a partir de un archivo de texto, en donde en cada línea habrá una A (Auto), C (Camion) o M (Moto). Luego, dependiendo de si es Auto, Camion o Moto se sortearán en forma aleatoria los demás atributos.

Al final listar los costos de cada Vehiculo y listar la recaudación total y promedio.

Ejercicio 1.20. En la Argentina, suele haber dos tipos de cuentas bancarias: la caja de ahorro, que permite extraer solamente un monto inferior al saldo de la cuenta; y la cuenta corriente, en la cual se define un valor de giro en descubierto, y que por lo tanto permite extraer más que el saldo, usando ese crédito especial que es el giro en descubierto. Implementar las clases necesarias para este problema. Generar en forma aleatoria un vector de 10 de estas cuentas con un saldo actual cada una.