Guía de Ejercicios 3 - POO

Parte I: Conceptos introductorios

1) Defina los conceptos de:

   a) Clase
   b) Objeto
   c) Atributo
   d) Sobrecarga
   e) Sobreescritura
   f) Jerarquía de clases
   g) Herencia
   h) Polimorfismo
   i) Binding
   j) Características y ventajas de:
      - Abstracción
      - Encapsulamiento
      - Modularidad

2) Dados los siguientes objetos (instancias de clases), agrupe los objetos de acuerdo a sus características comunes en sus respectivas clases.

- Rubí	- Mansión	- Lapicero
- Apartamento	- Gato	- Agenda
- Cerdo	- Cabaña	- Carpeta
- Lápiz de Color	- Casa	- Lápiz
- Conejo	- Oveja	- Cristal
- Esmeralda	- Perro	- Hexágono

3) Explicar la diferencia entre sobrecarga y sobreescritura.

4) ¿La sobrecarga es polimorfismo? Justificar su respuesta.

5) Definir las características de la programación orientada a objetos. Dar ejemplos de la vida real de cada uno de ellos.

6) ¿Qué características se tienen que dar para que haya sobrecarga de funciones?

7) La función primordial de un medio de transporte es de trasladar personas y objetos, estos se caracterizan por desarrollar velocidades determinadas (que van desde 0 km/h hasta la velocidad del sonido), y presentar un estado de mantenimiento (bueno, regular o malo). Estos medios de transporte, pueden clasificarse de distintas maneras. Una forma de hacer esto, es considerando su origen: pueden ser naturales o artificiales. Sin importar cual sea su origen, la función principal de estos, se redefine como la de trasladar personas u objetos. Los medios de transporte naturales, se caracterizan por su origen (animal o humano), número de extremidades, raza, color, entre otros. Además, complementan su función principal con otras tales como: comer, beber y cumplir necesidades fisiológicas. Cuando se habla de los medios de transporte artificiales, no se debe pasar por alto los materiales con que están hechos, los cuales forman una gama que va desde el plástico hasta los metales más fuertes, pasando por madera, fibras sintéticas y otros. En vista de que estos medios son creados por el hombre, también es importante considerar el nombre del fabricante y la fecha de fabricación. Por supuesto, la función de estos se define como trasladar personas u objetos. Los medios de transporte artificiales se dividen en terrestres, aéreos, acuáticos y anfibios.

Los medios terrestres presentan características comunes, tales como la medida de los cilindros del motor, el tipo de combustible que utilizan y número de ruedas. Los medios acuáticos, tienen las características comunes de calado y eslora. Los medios aéreos comparten los atributos de modelo y propulsión Un comentario especial se merecen los medios anfibios, ya que por sus características permiten el traslado tanto en tierra como en agua.

Se quiere que:

a) Identifique las clases existentes.
b) Incorpore herencias entre las clases.
c) Identifique atributos en las clases.

8) ¿Qué diferencias existen entre un lenguaje orientado a objetos tipificado y otro no tipificado? ¿En qué situaciones sintácticas se presenta la tipificación? Dar algunos ejemplos.

9) Diseñar 3 ejemplos (pueden ser mediante diagramas de clase) donde se violen principios SOLID. En cada ejemplo se debe explicar qué principios se violan y por qué. Los 5 principios deben reflejarse al menos una vez en alguno de los diseños propuestos.

Parte II: Introducción al Lenguaje Java

10) Implemente la clase Lamparita, que sirva para representar el estado de encendido de una lamparita (encendido o apagado). Defina, asimismo, dos métodos que permitan encender y apagar la luz de la lamparita y otro que indique en qué estado se encuentra. La lamparita inicialmente está apagada.

11) Dada la siguiente definición de clase:

```
public class Archivo {
   String nombre;
   int longreg;

   void init (int r) {
      longreg = r;
   }
};

Archivo f = new Archivo();
```

¿Qué sucede cuando se aplica f.init(80)?

12) Indique cuál es la salida del siguiente programa:

```
public class Punto {
   public int x;
   public int y;
   public Punto(int a, int b) {x = a; y = b;}
   public Punto(int z) {this(z, z);}
}

public class Prueba {
   public static void main(String[] args) {
      Punto p = new Punto(25);
      System.out.println("x = " + p.x + " y = " + p.y);
   }
}
```

13) ¿Cuál es el constructor que se invoca en la siguiente declaración?

```
public class Prueba {
   private int num;
   public Prueba() {num = 0;}
   public Prueba(int n) {num = n;}
   public int valor() {return num;}
}

Prueba prueba = new Prueba();
```

14) Se dispone de los siguientes métodos de una clase dada:

```
public void f(char c);
public void f(int i);
public void f(double d);
```

¿A cuál corresponde cada una de las siguientes invocaciones?

- f(72.25);
- f(0);
- f('z');

15)

a) Implemente la clase Hora que contenga miembros datos separados para almacenar horas, minutos y segundos. Un constructor inicializará estos datos en 0 y otro a valores dados. Una función miembro deberá visualizar la hora en formato *hh*:*mm*:*ss*. Otra función miembro sumará dos objetos de tipo hora, retornando la hora resultante. Realizar otra versión de la suma que asigne el resultado de la suma en el primer objeto hora.

b) Programar un procedimiento main(), que cree dos horas inicializadas y uno que no lo esté. Se deberán sumar los dos objetos inicializados, dejando el resultado en el objeto no inicializado. Por último, se pide visualizar el valor resultante.

16) Implemente la clase Empleado, que contenga como atributos miembros el número y nombre de empleado, y los siguientes métodos miembros:

a) getNumero(): Obtiene el número del empleado dado
b) getNombre(): Obtiene el nombre del empleado dado

c) setNumero(int num): Modifica el número del empleado dado
d) setNombre(String nom): Modifica el nombre del empleado dado
e) verDatos(): Visualiza los datos del empleado por standard output

Luego escribir un método que cree varios empleados, los complete con datos de empleados y luego visualice los datos de los mismos.

17) Implemente la clase Punto (pares de coordenadas de tipo float x, y). Defina constructores y métodos para asignar valores a las coordenadas de los puntos, retornar el valor de cada coordenada, y sumar dos puntos, retornando su resultado. Definir un método booleano de igualdad entre dos puntos.

18) Implemente la clase Vector3D (ternas de coordenadas de tipo float x, y, z). Defina constructores y métodos para asignar valores a las coordenadas de los vectores3D, retornar el valor de cada coordenada, y sumar dos vectores3D, retornando su resultado. Definir un método booleano de igualdad entre dos vectores3D. Implementar esta clase a partir de la implementación de la clase Punto.

19) Implemente la clase Complejo (números complejos). Defina constructores y el método de multiplicación de complejos. Además programe tres funciones suma(), una que reciba dos números de tipo int, otra que reciba dos números de tipo float, y otra que reciba dos números complejos.

20) La coordenadas propias de la clase anterior Punto representan coordenadas rectangulares (coordenadas x, y). Estas coordenadas se pueden escribir como coordenadas polares (radio, ángulo) de la siguiente manera:

    x = radio * cos(ángulo)

    y = radio * sen(ángulo)

    Implemente la clase Polar, cuyos miembros sean el radio y ángulo, y que posea métodos de obtención de cada coordenada polar, y otro que convierta las coordenadas polares en rectangulares. Programar un método que permita sumar coordenadas polares. Realizar un método de prueba para la clase.

21) Implemente la clase Fecha, que permita representar una terna de día, mes y año, todos de tipo entero. Programar un método que determine si una fecha es mayor a otra. Programar la sobrecarga del método toString().

22) Implemente la clase Fraccion con las operaciones aritméticas comunes (suma, resta, multiplicación y división), dos constructores, y el método toString().

23) Implemente la clase Ecuacion, que sea capaz de construir una ecuación de segundo grado (con sus componentes de tipo flota), y de calcular las raíces reales de la ecuación.

24) Implemente la clase Potencia, que permita crear objetos de la forma (base -float-, exponente -natural-). Programar un método evaluar(), que retorne el resultado de la evaluación de la potencia, utilizando recursión.

25)

a) Implemente las clases NumeroBinario y NumeroDecimal . La primera construye números binarios expresados como sucesiones de caracteres String de dígitos '0' y '1', y la segunda construye números naturales en notación decimal. Programar métodos aDecimal() y aBinario(), que permitan convertir números de binarios a decimal, y viceversa (retornando los resultados como objetos de la otra clase).

b) Programar una variante de las clases anteriores con sus métodos, si éstos fueran de tipo static. Justificar.

26) Implemente una clase Monedero que permita gestionar la cantidad de dinero que una persona dispone en un momento dado. La clase deberá tener un constructor que permitirá crear un monedero con una cantidad de dinero inicial y deberá definir un método para meter dinero en el monedero, otro para sacarlo y finalmente, otro para consultar el disponible; solo podrá conocerse la cantidad de dinero del monedero a través de este último método. Por supuesto, no se podrá sacar más dinero del que haya en un momento dado en el monedero.

27) Implemente una clase CuentaCorriente, que permita llevar el control del estado de una cuenta corriente. La cuenta corriente está caracterizada esencialmente por su saldo, y sobre ella se pueden realizar los siguientes tipos de operaciones:
    - saldo(): Retorna el saldo de la cuenta corriente (puede ser negativo).
    - deposito(float imp): Deposita un importe dado a la cuenta corriente.
    - extraccion(float imp): Extrae un importe dado de la cuenta corriente.
    - cantidadOperaciones(): Retorna la cantidad de operaciones válidas realizadas.
    - cantidadExtraccionesInvalidas(): Retorna la cantidad de extracciones en las que se intentó extraer más dinero del que existía en la cuenta corriente.

28) Implemente una clase Microondas que permita cocinar una comida. El microondas posee una puerta que puede estar abierta o cerrada, y un contenido que puede tener una comida o estar vacío. Se puede insertar una comida (si la puerta está abierta, y no hay otra comida adentro), retirar una comida (si la puerta está abierta y hay una comida adentro). También se puede iniciar una cocción con un nivel de intensidad de cocción y una cantidad de segundos de cocción, si la puerta está cerrada, y hay una comida adentro. También se puede finalizar la cocción (cuando se acaba su tiempo), lo cual la comida cocinada tendrá tantos puntos adicionales de cocción como el nivel de intensidad de la cocción finalizada multiplicado por la cantidad de segundos de esa cocción. Cuando una cocción se encuentra en curso, la puerta no se puede abrir. Lo que sí se puede hacer es abortar una cocción a una cantidad de segundos faltantes, si existe una cocción en curso. El efecto de esto último sería cocinar la comida por sólo el tiempo parcial en que estuvo cocinándose (el tiempo de la cocción del inicio menos el tiempo que faltaba al abortar) y abrir la puerta.

También se solicita que se pueda mostrar el estado completo del microondas en cualquier momento, incluyendo el de la comida que puede estar dentro. La comida se caracteriza por poseer un nombre y una cantidad de puntos de cocción.

29)

a) Explique qué ocurre en el entorno de una porción de código si una expresión propia de ese código, de un tipo básico de Java se pasa como parámetro a un método invocado, y esta expresión es modificada en el método invocado. ¿En qué estado queda la expresión en la porción de código original?

b) Idem anterior, pero lo que se pasa como parámetro no es una expresión de un tipo básico, sino un objeto de una clase. ¿Qué ocurre si dentro del método invocado se le aplican métodos que modifican su estado interno?

c) Idem b), pero preguntando si dentro del método invocado se le cambia la identidad a ese objeto (a la variable que representa ese objeto se le asigna otro objeto distinto)?

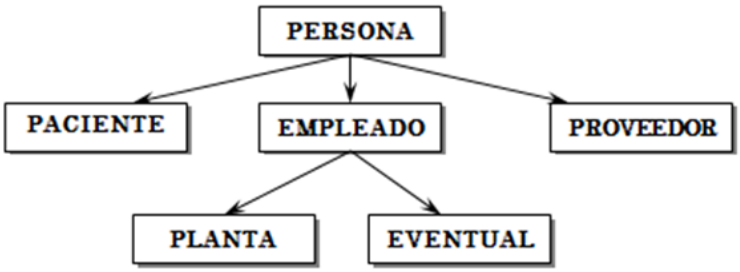
Parte III: Herencia y Polimorfirmo

30) Definir una jerarquía de animales que contenga al menos seis niveles de derivación y doce clases.

31) Definir una clase Persona que contenga información de propósito general común a todas las personas. Derive (subclasifique) las clases Estudiante y Empleado.

32) Una editorial de libros y discos desea crear fichas que almacenen el título y el precio de cada publicación. Definir la correspondiente clase Publicacion que implemente los datos anteriores. Derive dos clases, una llamada Libro, que contenga para cada libro el número de páginas, año de publicación y precio, y la clase Disco, con la duración en minutos y precio. Programar una aplicación que pruebe las clases.

33) El departamento de informática de un hospital está realizando un nuevo registro de datos del personal, pacientes y proveedores del hospital, y desea definir la siguiente jerarquía:



Definir las clases correspondientes de acuerdo a las atributos y métodos miembros que se describen abajo:

- Persona: nombre, direccion, ciudad, leer(), mostrar().
- Paciente: nombre, direccion, ciudad, codigoDiagnostico, telefono, fechaNacimiento, leer(), mostrar(), enviarFactura().
- Empleado: nombre, direccion, ciudad, codigoEmpleado, horasExtras, companiaSeguro, leer(), mostrar(), enviarSalario().

- Proveedor: nombre, direccion, ciudad, codigoVendedor, saldo, fax, telefono, descuentos, leer(), mostrar(), pagarFactura()
- Planta: datosEmpleado, salario, anosAntiguedad, pagarSalario()
- Eventual: datosEmpleado, honorariosPorHora, pagarSalario()

34) Definir una jerarquía de clases de tipos numéricos que extienda a una clase abstracta Numero. Las subclases a diseñar pueden ser Complejo, Fraccion, Vector y Matriz. Definir las operaciones de suma, producto y mostrar sobre las clases.

35) Suponga que posee dos clases, una llamada Rectangulo (cuadrilátero con dos lados iguales entre sí y otros dos lados también iguales entre sí), y Cuadrado (cuadrilátero con los cuatro lados iguales). Suponiendo que solamente importe poseer como atributo la medida de sus lados, y como método el cálculo del área, ¿cómo establecería la jerarquía de herencia entre las dos clases antes mencionadas? Discutir.

36) Definir la clase Automovil, que puede subclasificarse en AutoMediano o Camion. Los autos medianos son capaces de estar habilitados luego de la adquisición de un permiso en una fecha dada. Los camiones también podrán estar habilitados luego de la adquisición de un permiso, pero éste sólo podrá expedirse con la debida autorización previa de la concesionaria donde fue adquirido. Las concesionarias de camiones verifican ciertas características del camión para poder registrar al mismo. Este dato también es registrado dentro de la misma concesionaria.

37) Definir la clase Transaccion, que corresponde a una operación bancaria que se aplica sobre una cuenta bancaria. Estas operaciones pueden ser una consulta de saldo (mostrando por standard output), una extracción de un importe sobre la cuenta, y un depósito de un importe sobre la cuenta. Toda transacción debe ser capaz de mostrar el número de cuenta, y de aplicar su acción asociada sobre la cuenta.

38)

a) Definir la clase ExpresionAritmetica, que permita representar constantes numéricas enteras, operaciones binarias de suma y producto, y operaciones unarias de negación aritmética, incrementar y decrementar. Toda expresión deberá poder evaluar el

resultado de la expresión, retornando el valor entero resultante. Definir tantas subclases como posibilidades existan de armar expresiones aritméticas.

b) Idem anterior, pero definiendo sólo tres subclases, correspondientes a expresiones elementales, expresiones de operaciones unarias, y expresiones de operaciones binarias.

c) Definir un constructor adicional de esta clase (que construya expresiones aritméticas) a partir de una lista de símbolos que representen la expresión a construir expresada con un recorrido postorden. Asumir que se expresión es construida correctamente a partir de dicha lista.

39) Una empresa ferroviaria administra viajes en tren entre dos estaciones terminales de su red. Un viaje tiene asociado un trayecto (desde una estación terminal de origen a una de destino, con una distancia determinada y una cantidad de estaciones) y una cierta cantidad de vagones y una capacidad máxima de pasajeros. También posee qué tipo de viaje corresponde en relación a sus características técnicas, si es un viaje con tecnología diesel, si es eléctrico o si es de alta velocidad (esto es independiente del trayecto recorrido). El tiempo de demora promedio -en minutos- en un tipo de viaje diesel es la distancia en kilómetros multiplicada por la cantidad de estaciones dividido 2 sumada a la cantidad de estaciones y de pasajeros dividido 10. El tiempo de demora promedio -en minutos- en un tipo de viaje eléctrico es la distancia en kilómetros multiplicada por la cantidad de estaciones dividido 2. El tiempo de demora promedio -en minutos- en un tipo de viaje de alta velocidad es la distancia en kilómetros dividido 10. Definir dentro de la clase Viaje el método tiempoDeDemora, que retorne la cantidad de minutos que tarda en efectuar su recorrido, can las siguientes variantes:

a) Especializando la clase Viaje en función del tipo de viaje.

b) Sin especializar la clase Viaje, relacionándola con la clase TipoDeViaje, que está especializada por cada tipo de viaje.

Parte IV: Generics

40) Definir una clase como generic por un lado, y como su correspondiente no generic (raw type) por otro. Intente utilizar la clase genérica como raw type. ¿Qué conclusiones saca?

41) Definir la clase genérica Wrapper<T>, que contenga un atributo de la clase T. Definir métodos getters, setters y un visualizador por standard output que deleguen en métodos del atributo.

42) Definir la clase genérica Par<A,B> que permita armar pares de elementos de las clases A y B. Definir getters y setters sobre sus elementos.

43) Modificar la clase Par<A,B> previa para hacer que sea inmutable.

44) Definir la clase genérica Pila<E>, con sus operaciones elementales (push, pop, size, etc). Utilizar una implementación utilizando un array, y otra utilizando un ArrayList.

45) Definir la clase genérica Cola<E>, con sus operaciones elementales (enqueue, dequeue, size, peek, etc). Utilizar una implementación utilizando un array, y otra utilizando un ArrayList.

46) Definir la clase genérica ColaPrioridad<E>, con sus operaciones elementales (enqueuePriority, dequeueHigh, size, peek, etc). Utilizar una implementación utilizando un array, y otra utilizando un ArrayList.

47) Definir las clases genéricas Either<A,B> y Maybe<A> con sus proyectores conocidos. Luego utilizarlas para definir la clase de las listas como Lista<A>.

Parte V: Interfaces

48)

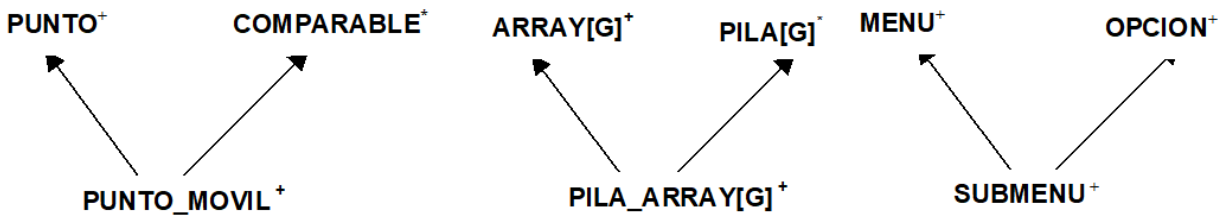
a) Definir una interfaz Medible, que se caracteriza por la existencia de métodos de obtención de medida (retorna un float), de incremento (a partir de un float dado) y de decremento (a partir de un float dado, retornando un booleano si tuvo errores).

b) Definir la clase Termómetro (se crea a partir de una temperatura inicial dada en ºK, y no puede bajar de -273ºK), la clase CuentaBancaria (se crea con saldo de $0, y no puede tener saldo negativo) y la clase MedidorDePresion (se crea con cero unidades). El medidor de presión cuenta internamente la cantidad de veces que tuvo incrementos consecutivos sin decrementos, incrementando en cada caso las unidades indicadas. El próximo decremento será de la cantidad de unidades indicadas dividido la cantidad de incrementos consecutivos inmediatamente anteriores. No pueden existir dos decrementos consecutivos. Las tres clases deberán implementar la interfaz Medible.

c) Armar un ArrayList de Medibles (con termómetros, cuentas bancarias y medidores de presión) en los que a cada elemento se le aplican 100 operaciones, que consisten en la aplicación arbitraria de operaciones de incremento y decremento (mostrando las operaciones de decremento erróneas), y cada 10 operaciones, se le obtiene la medida de ese momento y se la visualiza por standard output.

49) Definir una interfaz Pila (pila de enteros), con operaciones de encolar, desencolar (obteniendo el elemento desencolado) y un método de verificación de cola vacía. Implementar la pila con una clase que contenga un ArrayList, y con otra clase que contenga un array, donde si al array se le acabaran sus elementos, será necesario crear otro similar con una cierta cantidad de elementos más (indicada en su constructor).

50) Exprese en Java las siguientes jerarquías utilizando interfaces y/o subclasificación:



51) Definir un método que permita el ordenamiento de un ArrayList<T>. Considerar que el tipo paramétrico T debe implementar la interfaz Comparable.

52) Se desea definir una clase abstracta Sort que es la raíz de una jerarquía de clases que implementan algoritmos de ordenación de arrays, y que incluye: un atributo para almacenar el array de elementos, un atributo para llevar cuenta del número de comparaciones y otro para los intercambios, un método que implementa el algoritmo de ordenación pero que en esta clase es diferido, un método para añadir un elemento al array, un método para comparar dos elementos del array en dos posiciones cualesquiera y un método para intercambiar dos elementos del array en dos posiciones cualesquiera. Definir esta clase, especificando los atributos y signaturas de los métodos mencionados, y otros que considere necesarios. ¿Qué condición tendrá que cumplir una determinada clase para que sus objetos puedan ser ordenados aplicando uno de estos algoritmos? ¿Podría la clase Sort ser una interfaz?

53) Definir una clase VehiculoMotorizado que sirva como clase padre para vehículos de tipo Motocicleta, Automóvil y Camión. Todos los vehículos poseen un fabricante, modelo, año de fabricación y kilometraje. Los automóviles son de distintos estilos y las motocicletas se dedican a usos determinados. A su vez, los camiones pueden tener uno o varios remolques y tienen un nivel de seguridad dependiendo de si sobrepasan o no el número máximo de pasajeros autorizados.

    Definir también una interfaz llamada CapacidadLimite implementada por las clases Automóvil y Camión. Esta interfaz debe incluir constantes para el límite de pasajeros admitidos en automóviles y camiones. Los límites para automóviles deben incluir límite de pasajeros para automóviles normales y para vans. Con esta estructura de clases escriba un programa principal que usando una referencia polimórfica construya un objeto de clase Automovil, Motocicleta o Camión según decisión del usuario. El programa deberá imprimir la información del vehículo considerado.

54) Una fórmula química se representa utilizando un array ordenado de componentes químicos. Cada componente químico lleva esencialmente un String (el texto de su componente), y debe ser capaz de compararse con otro componente químico. Un componente químico es menor o igual a otro si su nivel de electronegatividad es menor o igual (el componente químico C posee la menor electronegatividad, luego H, y luego el resto de los componentes químicos ordenados lexicográficamente).

Se solicita que se definan clases que implementen las interfaces Comparable y Comparator para poder comparar dos componentes químicos con el objeto de ordenar listas de componentes químicos, y llevarlos a una fórmula química, utilizando un algoritmo de ordenamiento que utilice alguna de estas dos interfaces de comparación.

55) Un sistema permite generar diversos tipos de reportes que tienen un formato y operaciones particulares, por ejemplo reporte de ventas, compras, stock, etc. Los reportes comparten algunas propiedades en común, como título, contenido, etc. A su vez, algunos reportes pueden ser impresos por el sistema, el cual posee una operación que permite imprimir todos los reportes en su cola de impresión. Por otra parte, algunos reportes pueden ser customizables, permitiendo operaciones para cambiar el color de fondo, color de letra, etc. Diseñar el sistema de tal forma que los reportes de venta puedan imprimirse y customizarse, mientras que los de compras sólo puedan imprimirse.

56) Definir una clase GeneradorDeNumeros, que a partir de un valor inicial, un valor final y un incremento (indicados en su constructor), sea capaz de obtener números enteros ordenados a pedido cada vez que se lo requiera, entre el valor inicial y el final, con el incremento dado. La clase GeneradorDeNumeros deberá implementar las interfaces Iterator<Integer> y List<Integer>.

57) Definir una clase EnumeradorString, que tome un String desde su constructor y que implemente las interfaces Iterator<Character> y List<Character> (asumir que cada elemento de un String corresponde a un Character).

58)

a) Definir una interfaz Estadistica<T> con métodos de mínimo, máximo y cantidad de elementos.

b) Definir la subinterfaz EstadisticaSumable<T> con métodos de suma y promedio (que herede de la anterior). ¿Hay alguna restricción sobre T?

c) Modificar las clases definidas GeneradorDeNumeros y EnumeradorString para que implementen la interfaz Estadistica.

d) Definir la clase ArrayListContainer<T> (clase que contiene un ArrayList<T> como atributo) que implemente la interfaz EstadisticaSumable<T>.

Parte VI: Excepciones

59) Explique cómo amplía la aparición del componente sintáctico de excepciones al criterio de tipificación en un lenguaje como tipificado orientado a objetos como Java. Justificar.

60) Agregue el manejo de excepciones a todos los ejercicios realizados en las prácticas anteriores.

a) Implemente una clase MatriculaAuto (matrícula de automóvil) que va a estar formada por dos atributos letra y número. En el método main se deberán crear objetos matrículas, a partir de sus componentes. La matrícula se considerará valida si la letra introducida es igual a la letra 'A' o 'B', y si el número tiene ocho dígitos. Si la matrícula fuera correcta se creará un objeto matrícula y se mostrarán por pantalla los valores de sus atributos. En caso de que la letra sea incorrecta o el número de matrícula no tuviera ocho dígitos, se lanzará una excepción. El método llamador deberá mostrar un mensaje ante la excepción recibida indicando que la letra es errónea o el formato es erróneo, dependiendo de la situación ocurrida.

b) Dado que está desestimado que un constructor de una clase lance una excepción, ¿cómo cambiaría el punto anterior para que el constructor no lance la excepción sugerida?

61) Determine si el siguiente código es correcto. Si produce un error, ¿de qué tipo es? Solucionelo.

```
class Excepcion1 extends Exception {}
class Excepcion2 extends Exception1 {}
public class Test {
    public static void main(String[] args) {
       try {
          throw new Exception2();
       }
       catch(Excepcion1 e1) {
            System.out.println("Se capturó la Excepción1");
       }
       catch( Excepcion2 e2) {
          System.out.println("Se capturó la Excepción2");
       }
    }
}
```

62) Ejecute el siguiente código. ¿Cuál es el resultado? Elimine los comentarios y vuelva a ejecutarlo. ¿Cuál es el resultado?

```
public class Test {
      public int unMetodo(){
      //    try {
                  System.out.println("Va a retornar 1");
                  return 1;
      //    } finally {
            System.out.println("Va a retornar 2");
                  return 2;
      //    }
      }

      public static void main(String[] args) {
            Test1 res = new Test1();
            System.out.println(res.unMetodo());
      }
}
```

63) ¿Qué mejoras sugiere al siguiente código?

```
public class Test {
      boolean estado;
      void on() {estado = true;}
      void off() {estado = false;}
      public static void main(String[] args) {
            Test2 switch = new Test2();
            try {
                  switch.on();
                  // algún código
                  switch.off();
            } catch (NullPointerException e) {
                  System.out.println("Ocurrió    un    error    de
puntero");
                  switch.off();
            } catch (IllegalArgumentException e) {
                  System.out.println("Ocurrió un error de I/O");
                  switch.off();
            }
      }
}
```

64) Ejecute el siguiente código. ¿Cuál es la salida del programa?

```
class Ex extends Exception {}
public class Test {
      public static void main(String[] args) {
            System.out.println("Test3");
            try {
                  System.out.println("Primer try");
               try {
                  throw new Ex();
                  } finally {
                  System.out.prinln("Finally del Segundo try");
                  }
            }
            catch(Ex e) {
                  System.out.println("Se capturó la Excepción Ex del
Primer try");
            } finally {
                  System.out.println("Finally del Primer try");
            }
      }
}
```

65) Analice el siguiente código y determine si es correcto. Si hay errores, escriba el motivo de cada uno y proponga una solución.

```
class FutbolException extends Exception {}
class Falta extends FutbolException {}
class EquipoIncompleto extends FutbolException {}
class ClimaException extends Exception {}
class Lluvia extends ClimaException {}
 class Mano extends Falta{}
class Partido {
      Partido() throws FutbolException {}
      void evento() throws FutbolException {}
      void jugada() throws EquipoIncompleto, Falta {}
      void penal() {}
}
inteface Tormenta {
      void evento() throws Lluvia;
      void diluvio() throws Lluvia;
}
public class Encuentro extends Partido implements Tormenta {
```

```
        Encuentro() throws Lluvia, FutbolException {}
        Encuentro (String fecha) throws Falta, FutbolException {}
        void penal() throws Mano {}
        public void evento() throws Lluvia {}
        public void diluvio() throws Lluvia {}
        public void evento() {}
        void jugada() throws Mano {}
        public static void main (String[] args) {
             try {
                  Encuentro enc = new Encuentro();
                  enc.jugada();
             }
             catch (Mano e) {}
             catch(Lluvia e) {}
             catch(FutbolException e) {
                 try {
                  Partido par = new Encuentro();
                  par.jugada();
                  }
                  catch(EquipoIncompleto e) {}
                  catch(Falta e) {}
                  catch(Lluvia e) {}
                  catch(FutbolException e) {}
             }
        }
}
```

66) Modifique el código de las siguientes clases para disparar sus propias clases de excepciones, en lugar de imprimir a *System.err*.

```
public class Asercion {
private static void imprimirError(String msg) {
     System.err.println(msg);
}
public final static void es_verdadero(boolean exp) {
     if (!exp) imprimirError("Aserción Falsa");
}
public final static void es_falso(boolean exp) {
     if (exp) imprimirError("Aserción Falsa");
}
public final static void es_verdadero(boolean exp, String msg) {
     if (!exp) imprimirError("Aserción Falsa"+msg);
}
public final static void es_falso(boolean exp, String msg) {
     if (exp) imprimirError("Aserción Falsa"+msg);
}
```

```
}


public class Test4 {
      Asercion.es_verdadero((2+2) == 5);
      Asercion.es_falso((1+1) == 2);
      Asercion.es_verdadero((2+2) == 5, "2+2 == 5");
      Asercion.es_falso((1+1) == 2, "1+1 != 2");
}
```

Parte VII: Colecciones

67) Definir la clase SucesionEstadistica, que permita agregar elementos a una sucesión acotada de elementos numéricos, para que luego permita obtener la cantidad de elementos, la media aritmética y el desvío standard.

68) Implementar una función múltiplos que dados dos números enteros X e Y, devuelva un arreglo de Y elementos que sean los primeros múltiplos de X. Por ejemplo:
múltiplos(3, 6) = [3, 6, 9, 12, 15, 18].

69) Implementar una función reversa que dada una cadena devuelva otra cadena pero con los caracteres invertidos. Por ejemplo:
reversa("hola") = "aloh"

70) Implementar una función maxMin que, dada una cadena con números enteros (positivos y negativos) separados por espacios, devuelva el máximo y el mínimo.

71) Implementar una función cantidadUnicos que, dada una colección de elementos, devuelva la cantidad elementos únicos que tiene.

72) Implementar una función únicos que, dada una colección de elementos, devuelva otra colección pero sin elementos repetidos (conjunto).

73) Implementar una función cuentaSuma que, dado un arreglo de números enteros, devuelva un Par<A, B> donde el primer elemento es la cantidad de números positivos del arreglo y el segundo elemento es la suma de los números negativos del arreglo.

74) Implementar una función esAnagrama que dadas dos cadenas devuelva si la segunda es un anagrama de la primera.

75) Implementar una función esSecuencia que permita identificar (verdadero o falso) si dado un arreglo de números enteros es posible reordenarlos para que formen una secuencia de números consecutivos sin repeticiones.

76) Implementar una función sumaX que devuelva verdadero si, dado un arreglo de números enteros y un número X, existen 3 números del arreglo que sumados den el valor X. Por ejemplo:
sumaX([1, 6, 0, 3, 5, 4, 3], 8) = true (1 + 3 + 4 = 8)

77) Implementar una clase ArregloDinamico de enteros que modele un arreglo indexado pero sin la necesidad de especificar un tamaño fijo del arreglo. A medida que se incorporan elementos la estructura debería adecuarse internamente para insertarlos. Se debe utilizar como estructura un arreglo nativo de Java (por ejemplo, int[]).

78) Implementar una versión genérica del punto anterior: ArregloDinamico<T>.
https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html#newInstance-java.lang.Class-int-

79) Definir las clases Pila y Cola con sus operaciones (ver ejercicio Pila<E>) como subclases de la clase ArregloDinamico.
    a) Definir las mismas clases con sus operaciones (ver ejercicio Cola<E>), pero conteniendo dentro de su estructura un objeto de la clase ArregloDinamico.
    b) ¿Qué conclusiones puede sacar?

80) Definir la clase Matriz<E> que incluya operaciones básicas sobre la construcción, proyección y modificación de una matriz:
    Crear vacía o desde arreglo nativo
    Agregar fila/columna
    Eliminar fila/columna
    Cambiar orden de filas/columnas
    Ordenar filas según valores en conjunto de columnas

81) Definir la clase MatrizCuadrada<E>, donde E deberá corresponderse con una clase numérica. Se deberán definir los métodos de obtención de la matriz traspuesta y obtención de suma de matrices. Se puede aprovechar la Matriz del ejercicio previo.

82) Definir la clase ArrayListRest<E> (arraylist con restricciones), que permita verificar una cierta condición dada sobre sus elementos cada vez que se quiere modificar el contenido de sus elementos. Si esa condición no se cumpliera sobre cómo quedarán sus elementos al momento de una determinada operación, entonces esa operación no se llevará a cabo.

83) Implementar un método que permita buscar un elemento en una colección y lo devuelva si existe, de lo contrario devuelva un valor por defecto que se pasa como argumento.

84) Implementar un método que permita filtrar una colección de elementos según un cierto criterio. Por ejemplo, filtrar elementos menores a cierto valor, filtrar elementos distintos a cierto valor, etc.

Parte VIII: Integradores

85) Definir la clase JuegoEncastre que modele un juego donde existe un conjunto de huecos con diferentes tamaños (largo, ancho, profundidad) y otro conjunto de bloques también con diversos tamaños (largo, ancho, altura). El juego consiste en verificar si un bloque puede colocarse perfectamente en un hueco (coinciden sus medidas). Al inicializarse el juego se generan X huecos y bloques con tamaños varios y luego se intenta colocarlos en cada hueco hasta completarlos todos. Devolver la cantidad de intentos que fueron necesarios.

86) Definir la clase Libro que contiene páginas divididas en capítulos. Cada página tiene un contenido en texto asociado y su número (que depende de la ubicación en el libro). El libro tiene también un índice de capítulos donde se puede consultar el número de página donde comienza. Se desea modelar la siguiente funcionalidad:
    a) Obtener los nombres de capítulos del libro
    b) Obtener la cantidad de páginas totales del libro o de un capítulo
    c) Obtener la página inicial y final de un capítulo
    d) Obtener la cantidad de palabras totales del libro o de un capítulo
    e) Obtener la cantidad de caracteres totales del libro o de un capítulo
    f) Agregar una referencia en una página particular del libro. La referencia tiene un texto identificatorio asociado.
    g) Implementar la capacidad de agregar y eliminar páginas en un capítulo.
    h) Permitir buscar una página específica por número y obtener su contenido.

87) Definir la clase MazoPoker que modele un mazo de cartas de Poker. El mazo contiene cartas numeradas del 1 al 10, más las figuras J, Q, K, en los 4 palos disponibles: Pica, Corazón, Diamante y Trébol. Las cartas de corazón y diamante son de color rojo, las otras son de color negro. Incorporar los siguientes métodos:
    Barajar el mazo
    Sacar X cartas de arriba
    Sacar X cartas de abajo
    Colocar X cartas arriba
    Colocar X cartas abajo
    Ordenar el mazo según orden de palos solicitado (en cada palo se ordena del 1 al 10, J, Q, K)

88) Definir la clase PartidaCartas que permita agregar 2 jugadores donde a cada uno se le entregan 5 cartas de un mazo de póker.
    a) Implementar una versión donde gana quien suma más puntos con sus cartas (la J vale 11, la Q vale 12 y la K vale 13).
    b) Implementar una versión donde gana quien pueda formar un número de dos dígitos mayor con cartas numeradas entre 1 y 9. Por ejemplo, si tengo cartas 4, 1, J, 6 y 10, el número de dos dígitos más grande sería 64.
    c) Implementar una versión donde gana quien tiene la mayor cantidad de cartas con mismo número/letra, si hay empate gana quien tiene la carta más alta.
    d) Implementar una función que verifique si un jugador tiene una jugada de "escalera" en la que sus cartas forman una secuencia numérica consecutiva, como 2-3-4-5-6. Diferenciar si la escalera es con el mismo palo o no.
    e) Agregar un método que determine si un jugador tiene cartas del mismo palo.
    f) Agregar un método que determine si un jugador tiene cartas del mismo color.
    g) Permitir la opción de jugar múltiples rondas y llevar un registro de las victorias de cada jugadxr a lo largo de las rondas.

89) Definir la clase Estanteria, que permita mantener una serie de estantes, el cual cada uno de ellos contiene varios libros numerados correlativamente. Cada libro se caracteriza por poseer un nombre, un autor, una editorial y un año de publicación. Se pueden agregar estantes, agregar libros en un estante, eliminar un libro y buscar un libro, obteniendo si no existe, o en el caso de que exista, en qué estante se encuentra y cuál es su número asociado. Agregar la siguiente funcionalidad:
    a) Agregar la funcionalidad para listar todos los libros en un estante específico.
    b) Permitir la posibilidad de reorganizar los libros en un estante al cambiar su orden.
    c) Implementar un método que calcule la edad promedio de los libros en un estante (basado en el año de publicación).
    d) Agregar la capacidad de buscar libros por autor y listar todos los libros de un autor en la estantería.

90) Definir la clase AgendaTelefonica, que permita registrar individuos con su nombre, dirección y teléfono. Se deberá permitir que a partir de un nombre dado, devuelva la dirección y el número de teléfono. Además deberá permitir la carga de individuos nuevos, y la eliminación de individuos ya existentes.

91) Definir la clase Empresa, que permita contener empleadxs de distintas áreas de una empresa. Lxs empleadxs se caracterizan por poseer un nombre, apellido y número de

empleado. Considere a la empresa como una lista de áreas (se puede buscar por nombre o código de área). Definir métodos de:

obtención del área a la que pertenece un empleadx
registro de un nuevo empleadx
transferencia de un empleadx de su área a otra área de la empresa
baja de un empleadx de la empresa.

92) Diseñar un sistema de inventario simplificado para un negocio. El negocio vende productos que tienen un nombre, un precio y una cantidad en stock. Implementar las siguientes funcionalidades:
    a) Agregar productos al inventario.
    b) Actualizar la cantidad en stock de un producto después de una venta.
    c) Calcular el valor total del inventario.
    d) Mostrar una lista de productos agotados (con cantidad en stock igual a cero).
    e) Permitir la búsqueda de productos por nombre y mostrar su información detallada.
    f) Permitir la búsqueda de productos por nombre parcial y mostrar su información detallada.

93) Crear una simulación de una red social simplificada. Diseñar clases que representen usuarixs, publicaciones y amistades entre usuarixs. Implementar funcionalidades como:
    a) Publicar un mensaje o foto en el perfil de un usuarix.
    b) Agregar amistades y gestionar solicitudes de amistad.
    c) Mostrar el feed de noticias de un usuarix, que incluye las publicaciones de amigxs.
    d) Calcular estadísticas, como la cantidad de amigxs de un usuarix y la cantidad de "me gusta" en sus publicaciones.

94) Diseñar un sistema de reservas de hoteles simplificado. Crear clases que representen hoteles, reservas, habitaciones y clientes. Implementar funcionalidades para:
    a) Reservar una habitación en un hotel.
    b) Agregar la capacidad para que los clientes cancelen sus reservas.
    c) Calcular el costo total de una reserva.
    d) Consultar la disponibilidad de habitaciones en una fecha específica.
    e) Mostrar información detallada de las reservas realizadas por un cliente.