Cuando y como sobreescribir Equals() y hashCode():

Cuando quiero comparar 2 clases correctamente debe sobreescribir equals(), comparando campos de la clase, y si la comparación se basa en una colección que utiliza hashCode, entonces también debe sobreescribir el método public int hashCode(); (las LinkedList no usan hash).

```
@Override
public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Persona persona = (Persona) o;
        return edad == persona.edad && nombre.equals(persona.nombre);
}
```

Si se trata de un HashMap, HashSet, LinkedHashSet, etc…

```
@Override
public int hashCode() {
        return Objects.hash(nombre, edad); // Genera un hashCode consistente
con equals
}
```

Herencia vs Composición:

Herencia es cuando una subclase hereda los atributos y métodos de una superclase.

Composición es cuando una clase instancia a otra dentro de ella como un atributo. (suele ser preferible la composición antes que la herencia)

Casteo:

El **casteo** (o **casting**) en Java es un mecanismo que te permite convertir una variable de un tipo de datos a otro tipo, ya sea en un tipo primitivo o en un tipo de objeto. Existen dos tipos de casteo:

1. **Casteo implícito (upcasting)**: Java lo hace automáticamente cuando es posible realizar la conversión sin pérdida de datos. Por ejemplo, cuando conviertes un tipo `int` en un `long`, o un `char` en un `int`.
2. **Casteo explícito (downcasting)**: El programador lo realiza cuando es necesario, por ejemplo, al convertir un tipo de datos más general a uno más específico (o viceversa). Este tipo de casteo es común cuando trabajas con objetos y jerarquías de clases, como cuando conviertes entre tipos de clases que están relacionadas mediante **herencia** o **interfaces**.

Para realizar un downcasting, es necesario usar una conversión explícita, que se hace con la sintaxis de paréntesis: `(TipoDeseado)`.

```
Perro perro = (Perro) animal; // Downcasting: Convertimos de Animal a Perro

perro.hacerSonido(); // Imprime: El perro ladra
```

Iterator:

Todas las colecciones entienden iterator().
Es útil para trabajar con diferentes tipos de colecciones (como listas, conjuntos y colas) de manera uniforme.

```
Iterator iterator = coleccion.iterator();
while (iterator.hasNext()) {
    Cliente cli = iterator.next(); //me devuelve el siguiente elemento
    …//hago lo que tengo que hacer con cada elemento
}
```

Stream():

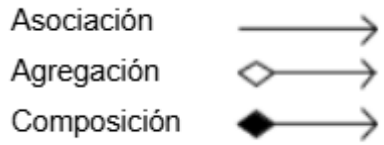
filter(): retorna un nuevo stream que solo “deja pasar” los elementos que cumplen cierto predicado.
map(): nos da un stream que transforma cada elemento de entrada aplicando una función que indiquemos.
sorted(): Se usa para ordenar los elementos de la secuencia en un orden específico.
¿Cuándo es bueno usar Streams()?

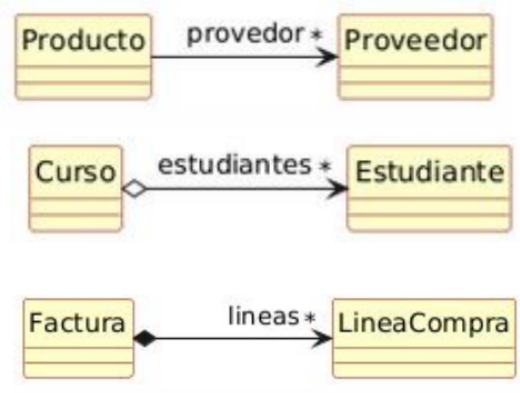
- Quiero ordenar una colección y que se mantenga ordenada por un criterio.
- Quiero eliminar los elementos que cumplen una condición.
- Cuando no requiero recorrer secuencialmente porque el algoritmo no lo requiere

Dates:

fecha.minusDays(cantidad): a un LocalDate le resta x días y lo puedo almacenar en una variable.
fecha.plusDays(cantidad): a un LocalDate le sumó x días y lo puedo almacenar en una variable.
fecha.isAfter(LocalDate otraFecha): retorna true si fecha está después que otraFecha.
fecha.isBefore(LocalDate otraFecha): retorna true si fecha está antes que otraFecha.
fecha.isEqual(LocalDate otraFecha): retorna true si fecha es igual a otraFecha.

UML:





Asociación: La asociación es adecuada cuando una clase necesita referenciar otra clase para cumplir sus funciones, pero la vida de una no depende de la otra.

Agregación: Cuando el objeto no depende de un solo “todo” para existir. (dependencia débil)

Composición: Cuando el objeto depende de un solo “todo” para existir (dependencia fuerte)

Dato: No es necesario declarar el tipo y el nombre de la relación (si es private, protected o public y un nombre representativo como en el ejemplo: líneas, estudiantes, proveedores, etc) pero si es recomendable indicar la multiplicidad en caso de que sea mayor a 1.

—----------

←—------>

Bidireccional: Cuando ambas clases necesitan conocerse mutuamente. (se puede indicar en cada extremo de la flecha la multiplicidad)

Herencia: Usar flecha normal, con la punta vacía.

Implementación: Usar flecha normal, con la punta vacía y línea punteada.

Dependencia: Se utiliza para mostrar que una clase o componente utiliza o necesita otro para funcionar correctamente, pero sin mantener una referencia duradera a él. - - - - - ->

TEST:

Anatomía de un test suite jUnit

- Una clase de test por cada clase a testear
- Un método que prepara lo que necesitan los tests (el fixture) y queda en variables de instancia.
- Uno o varios métodos de test por cada método a testear.
- Un método que limpia lo que se preparó, si es necesario.

```
public class nombreTest {
    Atributos
    @BeforeEach
    void setUp() {
        prepara lo que necesita el test, inicializa los atributos declarados arriba, etc.
    }

    @Test
    void testMetodoATestear() {
        código que testea el método en cuestión.
    }
```

Variantes del assert (algunas)

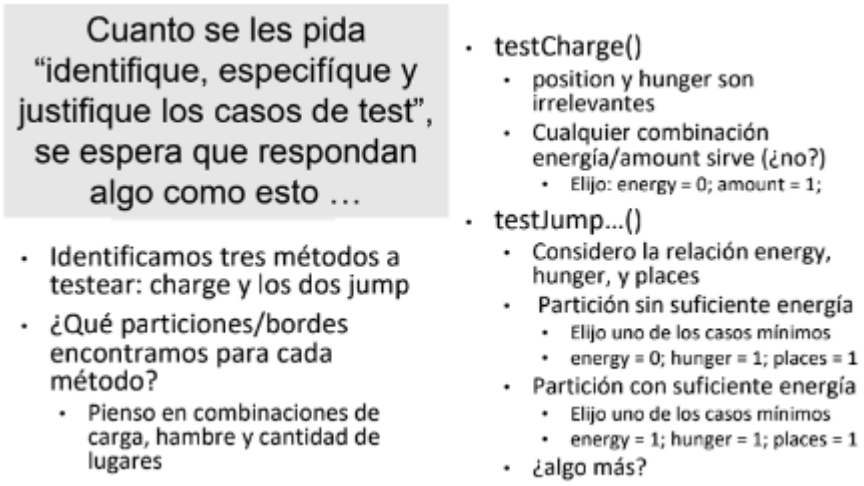
```
@Test
void assertExamples() {
    assertEquals(5, "Hello".length());
    assertNotEquals("Hello","Bye");
    assertNotNull(myList);
    assertSame(myList, someList);
    assertTrue(myList.isEmpty());
    assertFalse(someList.isEmpty());
    assertThrows(IndexOutOfBoundsException.class, () -> {
        myList.remove("Hello");
    });
}
```

Estrategias:

Partición de equivalencias: prueban lo mismo o revelan el mismo bug. Se asume que si un ejemplo de una partición pasa el test, los otros también lo harán. Se elige uno.

1. **Rangos numéricos**: Cuando un método acepta números dentro de un rango, puedes dividir ese rango en **subrangos** equivalentes y probar solo algunos de esos valores para verificar que la lógica del método funciona correctamente. (tomo un valor dentro del rango, y uno por fuera en cada lado del rango)
2. **Valor de parámetros booleanos**: Si un método acepta un valor **booleano** (verdadero o falso), entonces hay dos particiones equivalentes: **verdadero** y **falso**.

Valores borde: Se intenta identificar los bordes de la partición de equivalencia y se eligen esos valores. (se utiliza más que nada para rangos numéricos, verificar que funciona correctamente en los bordes, y fuera de ellos no)



Ejercicio 25. Veterinaria

Se debe desarrollar una plataforma para una veterinaria donde se permita registrar los servicios médicos provistos por un médico veterinario a una mascota.
De los médicos veterinarios se conoce el nombre, fecha de ingreso a la Veterinaria y honorarios a cobrar por atención. De las mascotas se conoce su nombre, fecha de nacimiento y especie (por simplicidad un String).

Los servicios médicos pueden ser consultas médicas, vacunaciones o servicio de guardería. Solo las consultas médicas y la vacunación tienen intervención de un médico veterinario.

- De las consultas médicas se conoce el médico, la mascota y la fecha de atención.
- De las vacunaciones se conoce el médico, la mascota, el nombre de la vacuna a aplicar y su costo.
- De los servicios de guardería se conoce la mascota y la cantidad de días.

Nos piden implementar la siguiente funcionalidad:

Dar de alta una consulta médica para una mascota: Dado un médico y una mascota, se crea una consulta médica para dicha mascota considerando la fecha actual como fecha de atención y la retorna.

Dar de alta una vacunación para una mascota: Dado un médico, una mascota, el nombre de la vacuna a aplicar y su costo, se crea la vacunación de dicha mascota considerando la fecha actual como fecha de vacunación y la retorna.

Dar de alta un servicio de guardería para una mascota: Dada una mascota y la cantidad de días, se crea el servicio de guardería para dicha mascota considerando la fecha actual como inicio del período y lo retorna.

Calcular el costo de un servicio: Los costos de los servicios se calculan de la siguiente manera:

- La consulta médica se cobra calculando la suma de los siguientes valores:
  - honorarios del médico veterinario que interviene
  - adicional de materiales descartables ($300).
  - adicional por atención en día domingo ($200).
  - adicional por antigüedad del médico ($100 por año de antigüedad).
- La vacunación se cobra calculando la suma de los siguientes valores:
  - honorarios del médico veterinario que interviene
  - adicional de materiales descartables ($500).
  - adicional por atención en día domingo ($200).
  - el costo de la vacuna utilizada
- La guardería se cobra según un costo diario de $500 y, si la mascota utilizó previamente 5 o más servicios, se le aplica un descuento del 10%.

Determinar la recaudación generada por una mascota en una fecha dada: dada una fecha, se debe retornar el monto recaudado en todos los servicios recibidos por esa mascota en dicha fecha.

Tareas:

a) **Modele e implemente**
   - **i)** Diseño de su solución en un diagrama de clases UML.
   - **ii)** Implementación en Java de la funcionalidad requerida.

b) **Pruebas automatizadas**
   - **i)** Diseñe los casos de prueba teniendo en cuenta los conceptos de valores de borde y particiones equivalentes vistos en la teoría.
   - **ii)** Implemente utilizando JUnit los tests automatizados diseñados en el punto anterior.