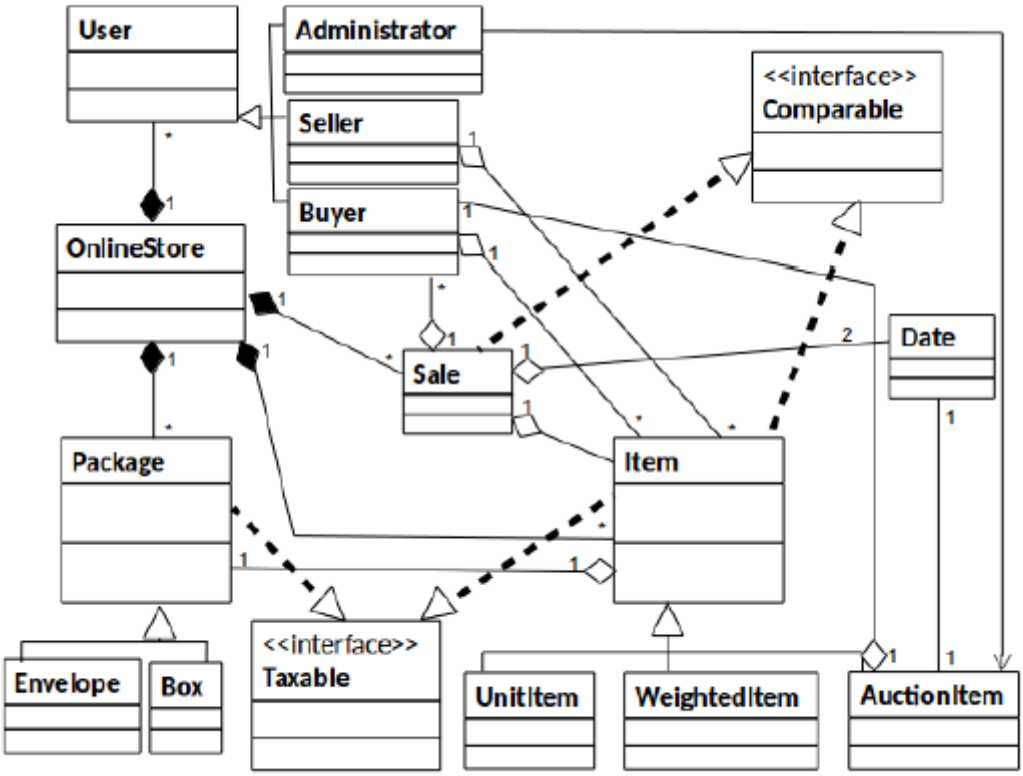
Oriol Estabanell, Albert Gubau u172749, u172750

Lab4
Programación Orientada a Objetos

-Descripción del problema a solucionar

En este laboratorio deberemos implementar el diseño realizado en el seminario 4, este diseño se basa en una tienda online (Online Store) en la que encontramos diferentes clases que mantienen relaciones de herencia con otras y nos permiten implementar correctamente como funcionaría una tienda online básica.

Este diseño se basará en una ampliación del diseño del laboratorio 3 en el que tendremos:



Como podemos ver, hemos añadido dos nuevas clases (Date y Sale) y dos interfaces(Taxable y Comparable).

Por lo tanto, deberemos modificar nuestra tienda online para implementar de manera correcta estas clases e interfaces. Se nos pide que:

Modifiquemos el **main** para tener un método que procese las ventas (**sell**) y un método que actualice el día en el que estamos (**dayPass**). La clase **OnlineStore** deberá tener almacenadas las **ventas** y el **día en el que estamos**.

Ordenemos los **items** por precio y las ventas por fecha (implementación de **Comparable**).

Implementemos la interfaz **Taxable** en **Item** y **Package**.

- Modificaciones en el main (OnlineStore)

- Añadimos atributos nuevos

En nuestro main deberemos añadir un atributo que tenga en cuenta el día en el que nos encontramos y otro atributo que mantenga un registro de todas las ventas realizadas.

Para poder añadir de manera correcta estos atributos hemos pensado en tener un atributo de tipo Date que sea la fecha actual al que llamamos currentDate y una LinkedList de ventas.

Para poder implementar estos atributos, primero deberemos definir las clases Date y Sale:

- Clase Date

Como sabemos por nuestro diagrama, la clase **Date** implementará la interfaz **Comparable**, por lo que tendrá que redefinir el método **compareTo** para poder ordenar las ventas por fecha, pero eso lo veremos más adelante. Nuestra clase Date la implementaremos de la siguiente manera:

```
public class Date implements Comparable{

    private int day;
    private int month;
    private int year;

    Date(int d, int m, int y){
        day = d;
        month = m;
        year = y;
    }

    public int getDay(){...3 lines }
    public int getMonth(){...3 lines }
    public int getYear(){...3 lines }
    public void setDay(int d){...3 lines }
    public void setMonth(int m){...3 lines }
    public void setYear(int y){...3 lines }

    @Override
    public int compareTo(Object o){

        if(o instanceof Date){

            Date date = (Date)o;

            if(year < date.getYear()){
                return -1;
            }
            else if(year > date.getYear()){
                return 1;
            }
            else if(month < date.getMonth()){
                return -1;
            }
            else if (month > date.getMonth()){
                return 1;
            }
            else if(day < date.getDay()){
                return -1;
            }
            else if (day > date.getDay()){
                return 1;
            }
        }
        return 0;
    }
}
```

Nuestra clase Date tendrá tres atributos (**day, month, year**), podemos observar su método constructor y sus métodos **getters** y **setters**.

En esta imagen tenemos ocultos los getters y los setters, estos métodos no tienen mator complicación que la que ya hemos visto en anteriores prácticas, los getters devuelven el atributo que indica su nombre y los setters modifican el valor del atributo por el que entra por parámetro del método.

Debajo de los getters y setters de la clase podemos ver la redefinición del método **compareTo** de la interfaz Comparable en la que se devuelve -1 si la fecha con la que comparamos es mayor que en la que estamos, 0 si es la misma fecha y 1 si la fecha en la que estamos es mayor que la fecha con la que comparamos.

La implementación de este método nos permitirá en un futuro ordenar las ventas por su fecha de realización.

- Clase Sale

Como sabemos por nuestro diagrama, la clase **Sale** implementará la interfaz **Comparable**, por lo que tendrá que redefinir el método **compareTo** para poder ordenar las ventas por fecha. Nuestra clase Sale la implementaremos de la siguiente manera:

```
public class Sale implements Comparable{

    private Item item;
    private Buyer buyer;
    private Date saleDate;
    private Date shippingDate;
    private Package pack;

    public Sale(Item i, Buyer b, Date sd, Date shd, Package p){

        item = i;
        buyer = b;
        saleDate = sd;
        shippingDate = shd;
        pack = p;
    }

    public Item getItem(){...3 lines }

    public Buyer getBuyer(){...3 lines }

    public Date getDate(){...3 lines }

    public Date getShippingDate(){...3 lines }

    public void setSaleDate(Date d){...3 lines }

    public void setShippingDate(Date d){...3 lines }

    @Override
    public int compareTo(Object o){

        if(o instanceof Sale){

            Sale s = (Sale)o;
            return s.getDate().compareTo(getDate());
        }
        return 5;
    }
}
```

En nuestra clase Sale declaramos 5 atributos referentes al **item** que se vende, el comprador que lo compra, la fecha de compra, la fecha de envío y el empaquetado del item que se vende.

Primeramente, podemos ver su método **constructor** y sus respectivos **getters y setters** que los mostramos ocultos porque sabemos como funcionan y no presentan ninguna novedad con respecto a los getters y setters que hemos visto en anteriores prácticas.

Por último, encontramos la implementación del método **compareTo** de la interfaz **Comparable** en el que llamamos al compareTo de la fecha para poder ordenar las ventas por su fecha.

Una vez hemos creado las dos clases necesarias para las modificaciones de nuestro main de la clase OnlineStore, ya podemos empezar a añadir los atributos nuevos de manera que tendremos:

```
public class OnlineStore {

    public static LinkedList< Item > itemsAvailable;
    public static LinkedList< Item > itemsSold;
    public static LinkedList< User > users;
    public static LinkedList< Package > packages;
    public static LinkedList<Sale> sales;
    public static Date currentDate;
    public static double totalPrice;
    public static double totalProfit;

    public static void init(){

        itemsAvailable = new LinkedList< Item >();
        itemsSold = new LinkedList< Item >();
        users = new LinkedList< User >();
        packages = new LinkedList< Package >();
        sales = new LinkedList< Sale >();
        currentDate = new Date(29,10,2020);
        totalPrice = 0.0;
        totalProfit = 0.0;
    }
```

Podemos ver como hemos añadido un atributo de tipo fecha que tiene en cuenta el diía en el que nos encontramos y una lista de ventas para mantener un registro de estas.

En el método **init** declaramos que el inicio de nuestra **OnlineStore** es el da **29/11/2020**.

- Añadimos el método sell y el método dayPass

También se nos pide añadir el método sell y el método **dayPass**, estos métodos nos servirán para procesar correctamente las ventas de nuestra tienda online y para actualizar el día en el que nos encontramos respectivamente.

- Método sell

Este método deberá vender los artículos de nuestra tienda online de manera que cuando venda un artículo cree una nueva instancia de **Sale** y la añada a la lista de ventas que tenemos como atributo de la tienda online. También calculara el precio, los beneficios y aplicará el método **buy** y el método **sell** del **Buyer** y el **Seller** respectivamente:

```
public static void sell(Item item, Buyer b, Seller s){

    b.buy(item);

    totalPrice += item.getPrice();

    if(item instanceof UnitItem){
        ((UnitItem)item).sell(0);

    }else if(item instanceof WeightedItem){
        ((WeightedItem)item).sell(0.0);
    }

    s.sell(item);

    int saleday = currentDate.getDay();
    int salemonth = currentDate.getMonth();
    int saleyear = currentDate.getYear();

    Date saledate = new Date(saleday, salemonth, saleyear);
    Date shippingdate = saledate;
    Sale sale = new Sale(item, b, saledate, shippingdate, item.getPackage());

    sales.add(sale);

    totalProfit += item.calculateProfit();

    itemsSold.add(item);

    itemsAvailable.remove(item);
}
```

Como podemos ver, el método **sell** toma por parámetro el **Buyer**, el **Seller** y el item que vendemos. Primero aplicamos el método buy del **Buyer**, después aumentamos el precio total y aplicamos el sell según el tipo de item que estamos vendiendo, seguidamente aplicamos el método sell del **Seller**, después creamos una nueva instancia de Sale con los parámetros necesarios según nuestra implementación de la clase Sale y por último, añadimos la venta a la lista de ventas, aumentamos el beneficio total, añadimos el item vendido a la lista de items vendidos y eliminamos el item de la lista de items disponibles.

- Método dayPass

Este método lo que hará será actualizar el día en el que estamos y si el nuevo dia coincide con la fecha de finalización de una subasta procesará la venta del artículo que se estaba subastando. Este método lo implementamos de la siguiente manera:

```
public static void dayPass(){

    int currentDay = currentDate.getDay();
    int currentMonth = currentDate.getMonth();
    int currentYear = currentDate.getYear();

    currentDay++;

    if(currentDay==32){

        currentDay = 1;
        currentMonth++;

        if(currentMonth == 13){

            currentMonth = 1;
            currentYear++;
        }
    }

    currentDate.setDay(currentDay);
    currentDate.setMonth(currentMonth);
    currentDate.setYear(currentYear);

    for(Item i: itemsAvailable){

        if(i instanceof AuctionItem){

            if(((AuctionItem)i).getDeadline().compareTo(currentDate) == 0){

                for(User u:users){

                    if(u instanceof Seller){
                        sell(i, ((AuctionItem)i).getBuyer(), (Seller)u);
                    }
                }
            }
        }
    }
}
```

Como podemos observar, tomamos la fecha actual y la actualizamos teniendo en cuenta los años los meses y los días. En nuestra implementación, todos los meses acaban el día **31**, de manera que si sobrepasamos ese día se pasa al día **1** del siguiente mes y con los años algo similar pero acabando en el mes **12**.

Después de actualizar el día recorremos la lista de **items** disponibles en busca de los items de subasta, cuando encontramos uno comparamos la nueva fecha con la de la finalización de la puja del **item** y si coinciden procesamos la venta con el método **sell** que hemos visto anteriormente.