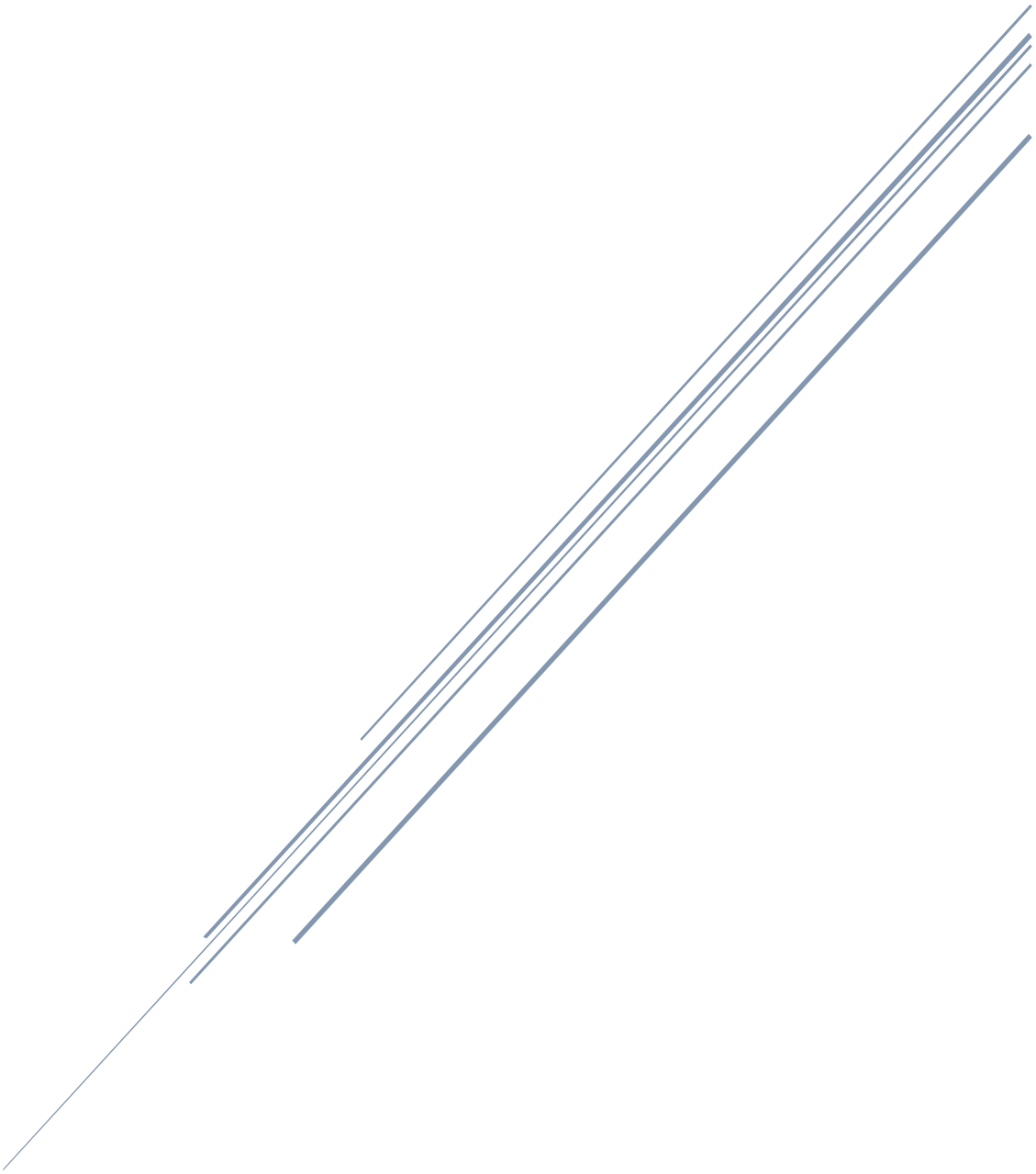
LAB SESSION 3: IMPLEMENTING INHERITANCE



Amanda Pintado Lineros u137702
Antonio Pintado Lineros u172771

El objetivo de esta practica consistía en crear una tienda online mediante la cual podríamos manejar sus usuarios, ítems y paquetes. Donde cada uno era heredado de otras subclases.

A su vez cada subclase es capaz de añadir nuevos métodos a su comportamiento para así tener otras funcionalidades diferenciables de las otras clases hijo.

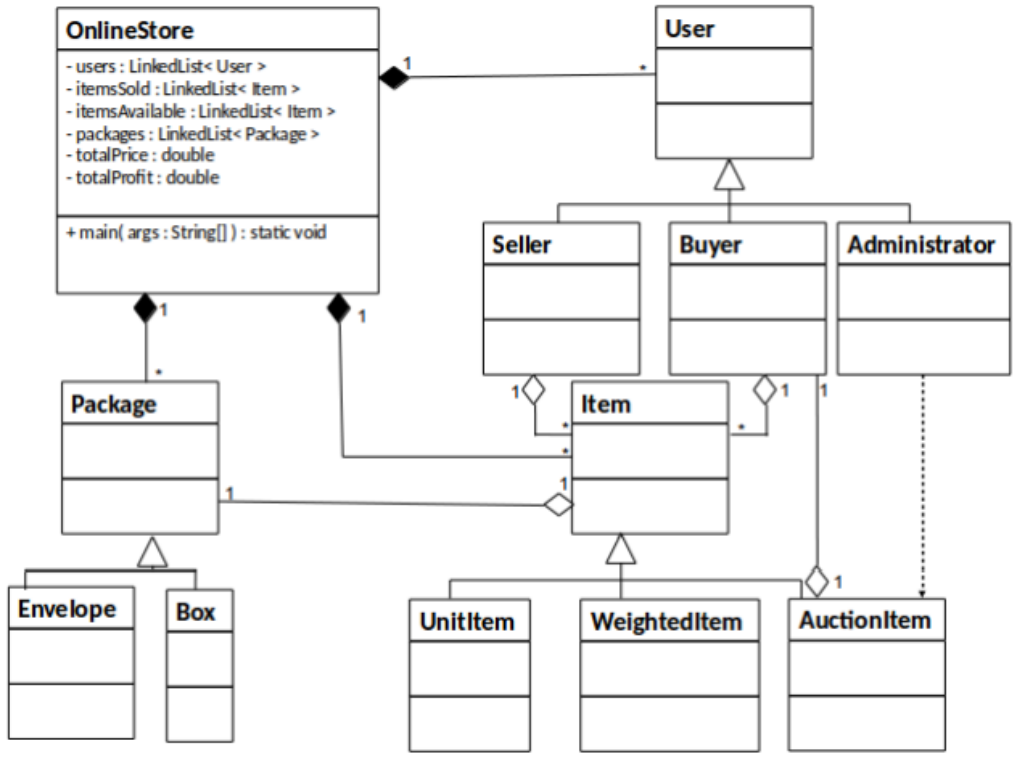
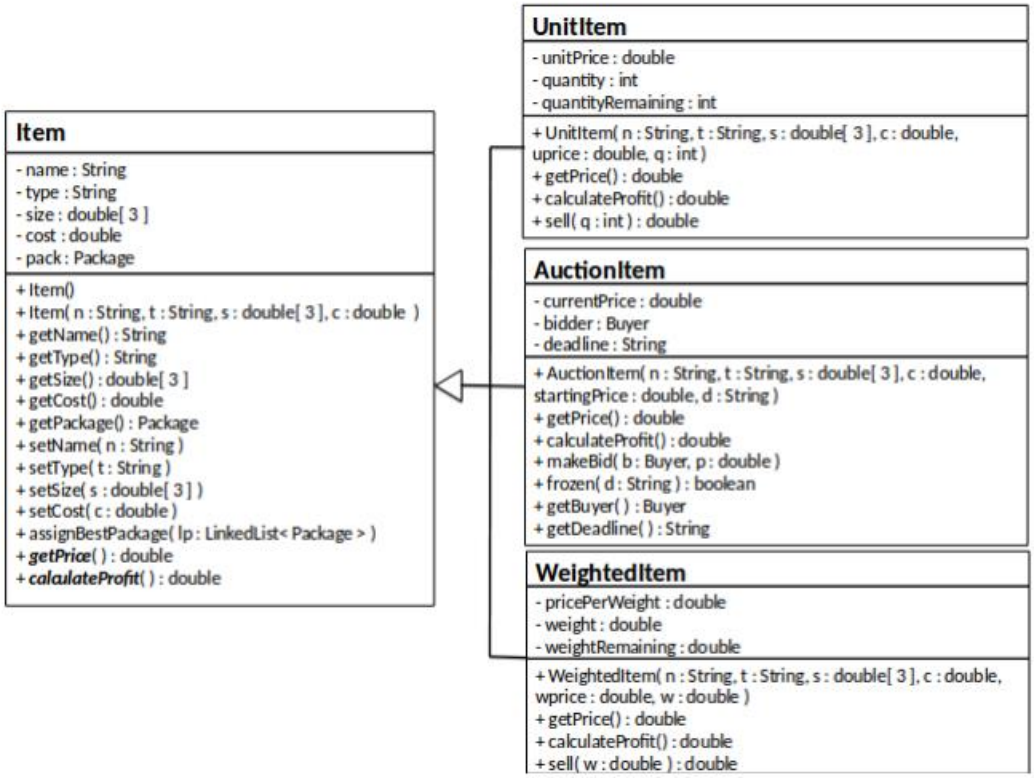


Figure 1: Class diagram for the store

Aquí por ejemplo se muestran la clase ítem y las clases que heredan de ella misma, UnitItem, AuctionItem, WeightedItem. En este caso las clases hijas heredaran todas las funcionalidades de ítem, y deberán implementar clases abstractas que no estarán definidas en ítem para que sean capaces de ser instanciadas.



Para llevar a cabo este concepto hemos creado las clases con los métodos necesarios para que todo funcione correctamente. En este ejemplo, de la clase UnitItem, podemos ver como se realiza un override de los métodos abstractos de la clase padre. Cada una de las clases hijas tendrán unas implementaciones que podrán ser iguales o diferentes del resto.

```java
@Override
public double getPrice() {
    return unitPrice * quantity;
}

@Override
public double calculateProfit() {
    //coste "real", ya que unitItem aquí es más barato que el precio original
    double originalCost = super.getCost() * quantity;
    return originalCost - getPrice();
}
```

Un método destacable es el método frozen de la clase AuctionItem, este método recibirá un parámetro de string, que indicara la fecha en la que se va a realizar la puja.

A partir de aquí comprobará si la fecha en la que se quiere pujar esta antes que la fecha límite, se podrá realizar la puja. Si no, no se podrá realizar. Se tiene que tener en cuenta que la fecha ingresada deberá estar en el formato “dd/MM/yyyy” para que el deadline sea añadido correctamente, de lo contrario no se añadirá.

```
public boolean frozen(String d) throws ParseException {
    Date dateInput = new SimpleDateFormat("dd/MM/yyyy").parse(d);
    Date dateFinal = new SimpleDateFormat("dd/MM/yyyy").parse(deadline);
    //Si la fecha del input está antes que la fecha final, no esta congelado y se puede comprar.
    if (dateInput.compareTo(dateFinal) < 0) {
        return false;
    }
    return true;
}

public void setDeadline(String deadline) {
    this.deadline = deadline;
}
```

Uno de los métodos que más trabajo ha llevado ha sido el método assignBestPackage de la clase Item.

La cual se compone de dos grandes partes, una que comprobará si el objeto al cual se le quiere asignar un paquete tiene una profundidad menor a tres, y otra si es mayor a 3. En función del resultado se le asignara un envelope o un box, al miembro de instancia pack, teniendo en cuenta que habrá que realizar un upcast cuando se haya encontrado.

La parte de abajo irá destinada a encontrar el mejor envelope para el ítem, con sus respectivos comentarios.

```
public void assignBestPackage(LinkedList<Package> lp) {
    //primero habra que comprobar la anchura del paquete
    if (size[2] <= 3) {
        //Se le debe asignar el mejor envelope, ya que la profundidas es
        //menor a 3
        //Creamos un envelope para luego guardarlo
        Envelope candidateEnvelope = new Envelope(0, 0, "");
        //Package candidatePackage = candidateEnvelope;
        //recorreremos todos los paquetes
        for (int i = 0; i < lp.size(); i++) {
            //primero comprobaremos que el paquete es un envelope
            //-------------------------
            if (lp.get(i) instanceof Envelope) {
                //ahora comprobamos que el envelope sea lo suficientemente
                //grande como para que quepa nuestro item
                Envelope currentEnvelope = (Envelope) lp.get(i);//downcast
                if (currentEnvelope.isSuitable(this.convertSize(size))) {
                    //si aun no hemos guardado un evelope candidato y
                    //seguimos teniendo guardado el envelope "vacio"
                    if ((candidateEnvelope.getWidth() == 0) && (candidateEnvelope.getHeight() == 0)) {
                        //nos quedaremos con el envelope que hemos encontra-
                        //do ahora
                        candidateEnvelope = currentEnvelope;
                    }
                    //si el envolpe que hemos encontrado es menor al que ya
                    //tenemos como candidato, significa que es lo suficiente
                    //grande como para guardar el item pero es más pequeño y
                    //se ajusta mejor al tamaño del item
                    if ((currentEnvelope.getWidth() < candidateEnvelope.getWidth()) &&
                            (currentEnvelope.getHeight() < candidateEnvelope.getHeight())) {
                        candidateEnvelope = currentEnvelope;
                    }
                }
            }
        }
    }
    //cuando nos hemos quedado con el envelope mas optimo, hacemos un
    Package candidatePackage = candidateEnvelope;//upcast
    //con nuestro candidatePackage escogido, ya podemos asignar al item su
    //tipo de paquete
    pack = candidatePackage;
    System.out.println("Envelope " + candidateEnvelope.getName() + " asignado al item " + name + "\n");
```

La segunda parte ira destinada a encontrar la mejor caja para nuestro ítem, ya que en este caso el depth de nuestro ítem será mayor a 3, y no cabrá en un envelope (sobre).

```
} else {
    //si el envelope no sirve, habra que hacer mas o menos las mismas
    //comprobaciones y quedarnos con el paquete que se acerque más a las
    //dimensiones del item
    Box candidateBox = new Box(0, 0, 0);
    //Package candidatePackage = candidateBox;
    //recorreremos todos los paquetes
    for (int i = 0; i < lp.size(); i++) {
        //primero comprobaremos que el paquete es un box
        //------------------------
        if (lp.get(i) instanceof Box) {
            //ahora comprobamos que el envelope sea lo suficientemente
            //grande como para que quepa nuestro item
            Box currentBox = (Box) lp.get(i);//downcast
            if (currentBox.isSuitable(this.convertSize(size))) {
                //si aun no hemos guardado un box candidato y
                //seguimos teniendo guardado el box "vacio"
                if ((candidateBox.getWidth() == 0) && (candidateBox.getHeight() == 0) && (candidateBox.getDepth() == 0)) {
                    //nos quedaremos con el box que hemos encontra-
                    //do ahora
                    candidateBox = currentBox;
                }
                //si el envolpe que hemos encontrado es menor al que ya
                //tenemos como candidato, significa que es lo suficiente
                //grande como para guardar el item pero es más pequeño y
                //se ajusta mejor al tamaño del item
```

```
                    if ((currentBox.getWidth() < candidateBox.getWidth())
                            && (currentBox.getHeight() < candidateBox.getHeight())
                            && (currentBox.getDepth() < candidateBox.getDepth())) {
                        candidateBox = currentBox;
                    }
                }
            }
        }
        //cuando nos hemos quedado con el Box mas optimo, hacemos un
        Package candidatePackage = candidateBox;//upcast
        //con nuestro candidatePackage escogido, ya podemos asignar al item su
        //tipo de paquete
        pack = candidatePackage;
        System.out.println("Box de tamaño (" + candidateBox.getWidth() + ", " + candidateBox.getHeight() +
                ", " + candidateBox.getDepth() + ") asiganado al item " + name + "\n");
    }

}
```

Otras clases que hemos tenido que implementar son las clases de paquetes, junto a sus subclases, envelope y box.

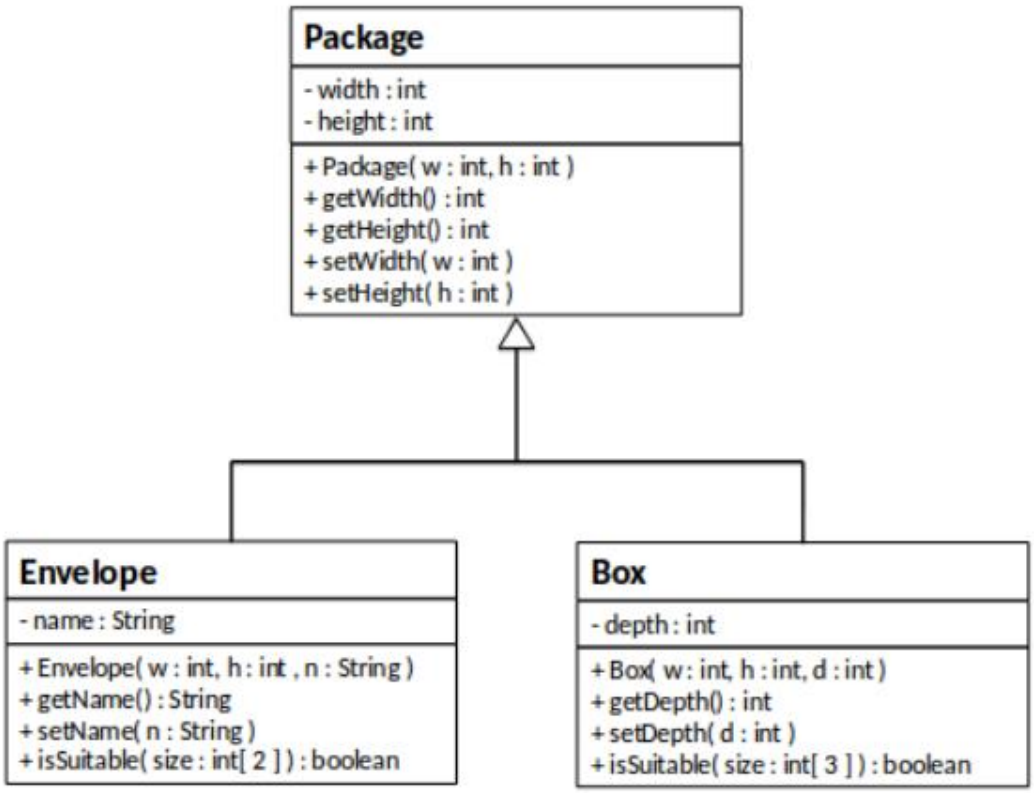


Figure 3: Class diagram for packages

La implementación de estas clases no ha sido de gran complejidad, pero cabe destacar los métodos de isSuitable de las clases Envelope y Box, este método será de utilidad para la implementación del método assignBestPackage de la clase Item a la hora de comprobar si un ítem entra dentro de un paquete o no.

Para ambas funciones tendremos que tener en cuenta que un objeto puede caber o no en un paquete si es rotado, así que se realizaran las dos comprobaciones oportunas para ver si esto se cumple.

Abajo se muestran los métodos implementados isSuitable, de clases Envelope y Package, respectivamente.

```java
public boolean isSuitable(int[] size) {
    //Haremos comprobaciones con el item "rotado" también
    if ((size[0] <= super.getWidth()) && (size[1] <= super.getHeight())
            || (size[1] <= super.getWidth()) && (size[0] <= super.getHeight())) {
        return true;
    }
    return false;
}
```

```java
public boolean isSuitable(int[] size) {
    //Haremos comprobacion del tamaño tambien con el objeto rotado SOBRE EL EJE
    //DE DEPTH rotando el item
    //Vamos a considerar que el eje depth es "como toca" porque sino no sabemos
    //muy bien como hariamos las comprobaciones para el envelope cuando no
    //tiene un atributo depth
    if ((size[0] <= super.getWidth()) && (size[1] <= super.getHeight()) && (size[2] <= depth)
            || (size[1] <= super.getWidth()) && (size[0] <= super.getHeight()) && (size[2] <= depth)) {
        return true;
    }
    return false;
}
```

También se han implementado la clase User, junto a sus subclases Buyer, Seller y Administrator. Buyer será el usuario que comprará objetos de nuestra tienda, seller será el que los vendera y el administrador será el que controlará el inventario de nuestra tienda, modificará los objetos a subasta y será capaz de expulsar un usuario. En nuestra implementación un administrador no será capaz de eliminar a otro administrador.

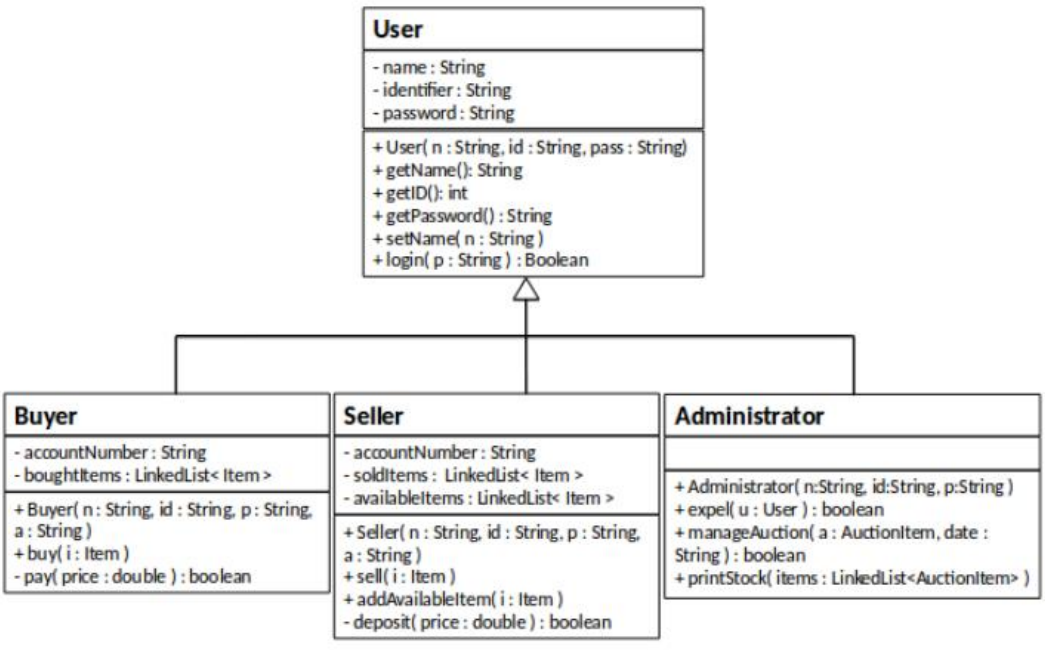


Figure 4: Class diagram for users

Aquí tenemos dos ejemplos de implementación, de los métodos buy y sell de las clases Buyer y Seller, los cuales emularán una compra/venta de un ítem mediante unos print que se realizaran a través de los métodos de pay y deposit a la vez que también a través de los métodos de buy/sell.

```
public void Buy(Item i) {
    if(pay(i.getPrice())){
        System.out.println("El usuario "+getName()+" ha comprado el objeto "+i.getName()+" por un precio de "+i.getPrice()+"\n");
        boughtItems.add(i);
    }
}

public boolean pay(double price) {
    if(!accountNumber.isEmpty()){
        System.out.println(price+" estan siendo cobrados en la cuenta "+accountNumber+" del usuario "+this.getName()+"\n");
        return true;
    }
    System.out.println("Error: El usuario "+getName()+" no tiene un numero de cuenta asociado.\n");
    return false;
}
```

```
public void sell(Item i) {
    if (availableItems.contains(i) && deposit(i.getPrice())) {
        soldItems.add(i);
        availableItems.remove(i);
        System.out.println("El objeto " + i.getName() + " ha sido vendido por el vendedor " +
                getName() + " por un precio de " + i.getPrice() + "\n");
        return;
    }
    System.out.println("El vendedor con id " + this.getID() +
            " no tiene el item ingresado. O no tiene un numero de cuenta asociado.\n");
}

public void addAvailableItem(Item i) {
    availableItems.add(i);
}

public boolean deposit(double price) {
    if (!accountNumber.isEmpty()) {
        System.out.println("Deposito de " + price + " realizado en la cuenta con numero: " + accountNumber
                + " a nombre del usuario " + getName() + ".\n");
        return true;
    }
    return false;
}
```

Por otra parte, el administrador también emulará una expulsión de otro usuario y a través de los métodos manageAuction y printStock será capaz de modificar el deadline de una instancia de auctionItem y printar el stock de una lista que ha sido para como parámetro.

```
public boolean expel(User u) {
    if(u instanceof Buyer ){
        System.out.println("El administrador "+getName()+" esta expulsando al comprador "+u.getName()+".\n");
        return true;
    }
    else if(u instanceof Seller){
        System.out.println("El administrador "+getName()+" esta expulsando al vendedor "+u.getName()+".\n");
        return true;
    }
    System.out.println("Error: Un administrador no puede expulsar a otro administrador.\n");
    return false;
}
```

```
public boolean manageAuction(AuctionItem a, String date) {
    DateFormat dateFormat = new SimpleDateFormat("dd/MM/yy");
    dateFormat.setLenient(false);
    try
    {
        dateFormat.parse(date);
        a.setDeadline(date);
        System.out.println("La fecha limite del objeto a subasta "+a.getName()+" ha sido cambiada a la fecha de "+date+"\n");
        return true;
    }
    catch (ParseException ex)
    {
        System.out.println("No se ha podido cambiar la fecha del objeto de subasta "+a.getName()
                +" por que no se ha introducido una fecha valida.\n");
        return false;
    }
}

public void printStock(LinkedList<AuctionItem> items) {
    System.out.println("El administrador "+getName()+" esta imprimiendo el stock actual. Se tiene como resultado:\n");
    for(int i=0;i<items.size();i++){
        System.out.println((i+1)+"- "+items.get(i).getName()+", price: "+items.get(i).getPrice()
                + ", con un coste de "+items.get(i).getCost()+"\n");
    }
}
```

Por último tenemos que añadir en la clase de OnlineStore los atributos de necesarios para tener guardados todos los usuarios, ítems y paquetes. A la vez que tener un recuento de los objetos que han sido vendidos, el precio total y el precio en beneficio total de la tienda online.

Para comprobar que todo funcione correctamente se ha añadido en la clase main las comprobaciones necesarias, como pueden ser añadir objetos a nuestra tienda y al vendedor, comprar ítems mediante los compradores, realizar pujas mediante los compradores, expulsar usuarios de nuestra lista…

Como resultado de nuestras pruebas tenemos el print que se muestra a continuación.

```
run:
Item Mobiliario Sofa asignado correctamente

Item Mobiliario Mesa asignado correctamente

Item Papeleria Libro asignado correctamente

Item Jardineria SustratoUniversal asignado correctamente

Item Comida Arroz asignado correctamente

Usuario Joe Black con ID Reaper registrado correctamente

Usuario Antonia Maria con ID AntoniaMiau registrado correctamente

Usuario Enrique Tomas con ID 4chi registrado correctamente

Usuario Maria con ID Miri registrado correctamente

Usuario John con ID Rapir registrado correctamente

Usuario Antonio Jose de los Palomares con ID Manolo registrado correctamente

Usuario Elyas Dorian con ID EdgeRed registrado correctamente

Box de tamaño (200, 300, 100) asiganado al item Sofa

Box de tamaño (100, 100, 100) asiganado al item Mesa

Envelope A4 asignado al item Libro

Box de tamaño (100, 100, 100) asiganado al item SustratoUniversal

Box de tamaño (10, 10, 10) asiganado al item Arroz
```

```
3495.0 estan siendo cobrados en la cuenta 123456789 del usuario Joe Black

El usuario Joe Black ha comprado el objeto Sofa por un precio de 3495.0

Deposito de 3495.0 realizado en la cuenta con numero: 154546789 a nombre del usuario Antonio Jose de los Palomares.

El objeto Sofa ha sido vendido por el vendedor Antonio Jose de los Palomares por un precio de 3495.0

105.0 estan siendo cobrados en la cuenta 123546789 del usuario Antonia Maria

El usuario Antonia Maria ha comprado el objeto Libro por un precio de 105.0

Deposito de 105.0 realizado en la cuenta con numero: 154546789 a nombre del usuario Antonio Jose de los Palomares.

El objeto Libro ha sido vendido por el vendedor Antonio Jose de los Palomares por un precio de 105.0

1.5 estan siendo cobrados en la cuenta 323126789 del usuario Enrique Tomas

El usuario Enrique Tomas ha comprado el objeto Arroz por un precio de 1.5

Deposito de 1.5 realizado en la cuenta con numero: 154546789 a nombre del usuario Antonio Jose de los Palomares.

El objeto Arroz ha sido vendido por el vendedor Antonio Jose de los Palomares por un precio de 1.5

Item Entretenimiento Ordenador asignado correctamente

Box de tamaño (100, 100, 100) asiganado al item Ordenador

El administrador Elyas Dorian esta imprimiendo el stock actual. Se tiene como resultado:

1- Ordenador, price: 8.0, con un coste de 1000.0
```

```
La fecha límite del objeto a subasta Ordenador ha sido cambiada a la fecha de 25/12/2020

El administrador Elyas Dorian esta expulsando al comprador Antonia Maria.

650.0 estan siendo cobrados en la cuenta 123456789 del usuario Joe Black

El usuario Joe Black ha comprado el objeto Ordenador por un precio de 650.0

Deposito de 650.0 realizado en la cuenta con numero: 154546789 a nombre del usuario Antonio Jose de los Palomares.

El objeto Ordenador ha sido vendido por el vendedor Antonio Jose de los Palomares por un precio de 650.0

Total price: 4251.5
Total profit: 591.8
BUILD SUCCESSFUL (total time: 2 seconds)
```

En general primero lo que se hará será inicializar todas las instancias necesarias de las clases ítems, usuarios y packages, asignar paquetes a los ítems, asignar objetos para vender a un vendedor. Comprar objetos a través de los usuarios y comprobar que se pueden realizar pujas.

Por ultimo se mostrará por pantalla cual es el total Price y total profit de nuestra tienda una vez se han realizado todas las pruebas.

Conclusiones

Hemos estado muy satisfechos del resultado obtenido. No hemos tenido grandes complicaciones a la hora de crear el programa, sin embargo algunos métodos o clases si que nos han llevado algo de tiempo como puede ser el método assignBestPackage de la clase Item. Pero se ha conseguido un resultado bastante aceptable teniendo en cuenta el esquema ofrecido.

Hemos añadido alguna que otra función auxiliar a nuestro programa ya que fue necesario para realizar alguna que otra funcionalidad, una de ellas es la de setDeadline de la clase auctionItem, este método ayudará a que el administrador sea capaz de modificar un auctionItem concreto a través del método manageAuction.