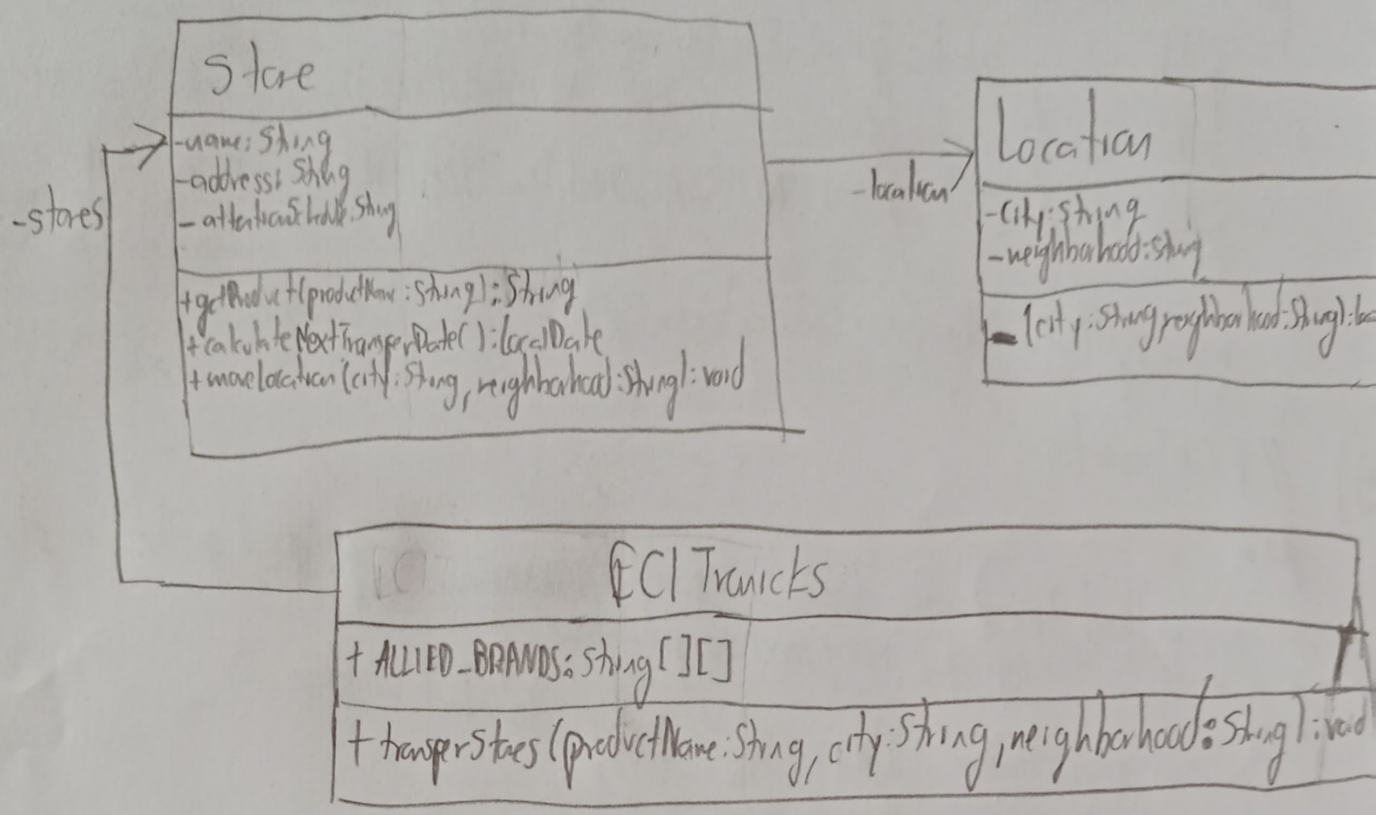
I Implementación

MDD

2)



3)

```
public class ECITrucks{
    public String[][] static final ALLIED_BRANDS;
    private HashMap<String, Store> stores;

}

public Interface PriceReduction{
    public void reducePrice(String productName, int percentage){

    }
```

```
4) public class ECITrucksException extends Exception {
    public String static final PRODUCT_NO_FOUND = "There is no product with that name".
    public String static final CITY_AND_NEIGHBORHOOD_REQUIRED = "If city or neighborhood are invalid".
    public String static final NOT_ALLOWED_TRANSFER = "If the transfer rate per the store has not been met yet or if no store could be transferred".

    public ECITrucksException(String message){
    super(message);
    }
}
5) // ECITrucks

/**
* This method transfer the location of all stores with the specified product in stock. The transfer
* is made to the specific city and neighborhood
*
*@param productName The name of the product to check in store
*@param city        city to move the mobile store
*@param neighborhood Neighborhood to move the mobile store
*@throws ECITrucksException three different exceptions
**/

public void transferStores(String productName, String city, String neighborhood) throws ECITrucksException{
    for(Store nombreTienda : stores.values()){
        if(nombreTienda.getProduct().equals(productName) && nombreTienda instanceOf MobileStore){
            nombreTienda.calculateNextTransferDate();
            nombreTienda.moveLocation(city, neighborhood);
        }
    }
    catch (ECITrucksException e){
        System.out.println(e.getMessage());
    }
}
```

```
// Store
/**
* Obtain the product
*
* @param productName the name of the product
* @return The product
* @throws ECITravicksException Product_NO_Found
**/
public String getProduct(String productName) throws ECITravicksException{
    for(String p: products.keySet()){
        if(p.equals(productName)){
            return p;
        }
        else{
            return "";
        }
    }
}
```

```
// Location
public void setLocation(String city, String neighborhood){
    this.city = city;
    this.neighborhood = neighborhood;
}
```

```
// MobileStore
public LocalDate getLastTransperDate(){
    return lastTransperdate;
}
public LocalDate getTransperDate(){
    return transperRate;
}
```

```
/**
* Move Location
*
* @param city City
* @param neighborhood Neighborhood
**/
public void moveLocation(String city, String neighborhood)
        throws ECITravicksException{
    d = this.calculateNextTransperDate();
    if(d != null && d < LocalDate.now()){
        this.setLocation(city, neighborhood);
    }
    else{
        throw new ECITravicks(ECITravicks.NOT_ALLOWED_TRANS);
    }
}
```

```
/**
*
* Calculate the next transper date
*
* @return LocalDate
**/
public LocalDate calculateNextTransperDate(){
    a = this.getLastTransperRate();
    b = this.getTransperRate();
    return (a+b);
}
```

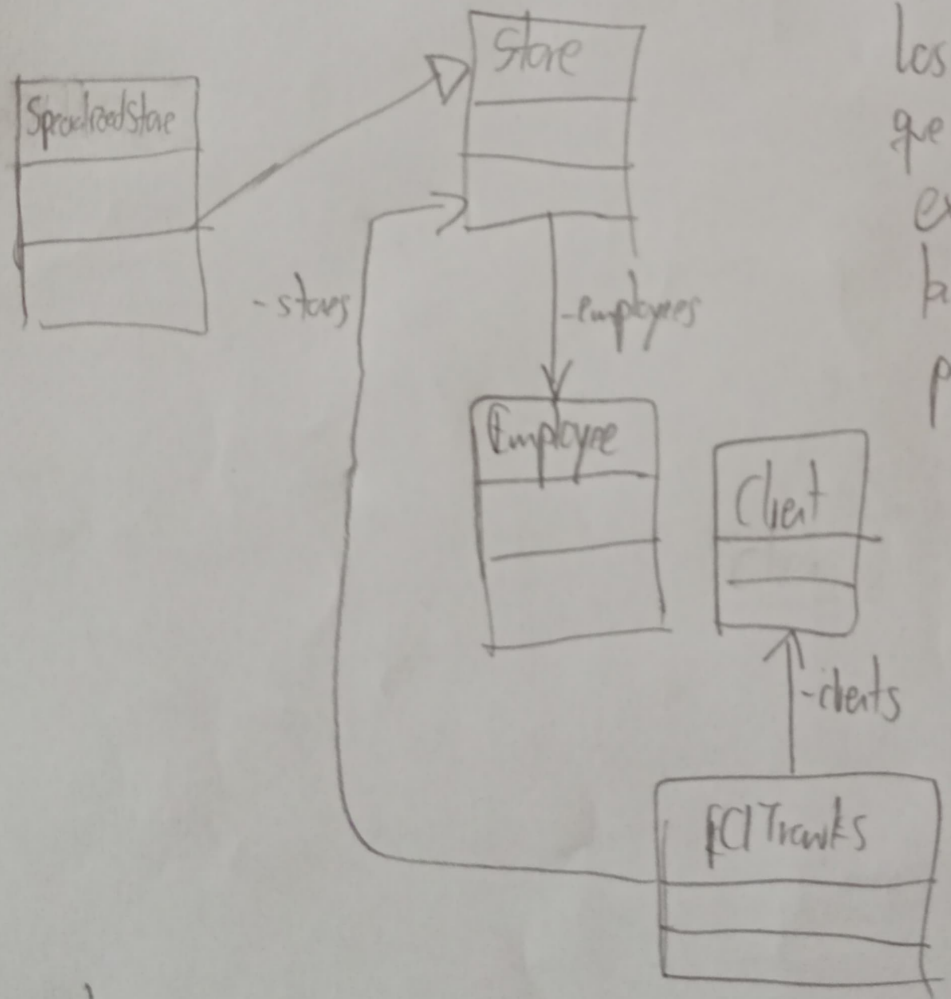
```
// Location
public Location(String city, String neighborhood)
        throws ECITravicksException{
    if(city.equals("") || neighborhood.equals("")){
        throw new Exception(Exception.CITY_AND
```

II Diseñando

```
// Interface PriceReduction
public void reducePrice(String productName, int percentage){
/**
 * Reduce price
 * @param productName Name of the product
 * @param percentage Percentage
 */
}
```

Requisitos para sobreescribir un método
Método definido en clase Padre (superclase)
Nombre parámetros deben ser los mismos
Visibilidad del método sobreescrito igual accesibilidad que el método original
Método no final
Ejemplo: Método declarado como final en la clase padre

III Extendiendo



Los cambios necesarios para que otra clase SpecializedStore se extienda de Store, y se pueden las clases que se necesitan para esta III parte

SOLID

Open-Closed Principle: se extiende más no se modifica porque no se altera el comportamiento de Store al tener otra clase que almacena propiamente SpecializedStore

IV Conceptos

Polimorfismo: Un objeto de una clase se comporta de diferentes maneras. Diferentes métodos-clases responde de manera diferente a una sola instrucción
Una clase Animal puede makeSound(), una clase Animal y clase Gato() sobreescriben el método para cada uno implementarlo a su manera