**INSTITUTO TECNOLÓGICO DE COSTA RICA**

**ESCUELA DE INGENIERÍA EN COMPUTACIÓN**

**LENGUAJES DE PROGRAMACIÓN**

**CUARTA TAREA PROGRAMADA**

**PARADIGMA ORIENTADO A OBJETOS**

**PROFESOR: OSCAR VÍQUEZ ACUÑA**

**ESTUDIANTES: BRANDON CHAVES VILLEGAS**

**XAVIER BLANCO ARIAS**

**FECHA DE ENTREGA: 13/06/2014**

**Descripción del problema**

El proyecto consiste en implementar una aplicación que sea útil para del funcionamiento de una tienda de reparación y venta de equipos electrónicos. El proyecto debe de implementarse en el lenguaje de programación Java, se debe hacer uso de todo lo aprendido sobre la programación orientada a objetos, hacer un diagrama de las clases implementadas y la explicación d porqué se utilizaron.

**Clases utilizadas**

**Articles**

Esta clase es utilizada para guardar las características generales que va a poseer cualquier artículo en la tienda, sin tomar en cuenta si es para vender o reparar, esta clase se hizo con el motivo de poder heredar de ella a sus dos hijos los cuales serían los artículos para reparar y los que son para vender, al hacer esta herencia permite el uso de una sola estructura para guardar cualquiera de estos dos tipos de artículos, evitar la repetición de código y de esta manera hacer el programa más eficiente.

**ToFixArticles**

Esta clase es hija de la clase Articles y define los atributos de los artículos que ingresan a la tienda para ser reparados, es necesaria en el modelo ya que los objetos para reparar cuentan con ciertas características que no están incluidas en la clase Articles.

**ToSoldArticles**

Esta clase también es hija de la clase Articles y se hizo para representar a los artículos de la tienda que son únicamente para ser vendidos, es necesaria porque al igual que en la clase toFixArticles, los artículos que son para vender poseen ciertos atributos que no están contemplados en la clase Articles.

**ListArticles**

Esta clase posee una lista que almacena objetos tipo articles, en la cual se van a almacenar todos los artículos de la tienda, la lista puede almacenar artículos para vender y también los que deben ser reparados ya que como la clase articles hereda a estos dos tipos de artículos esto es posible, también en esta clase se definen todos los métodos de manipulación de datos requeridos por la aplicación solicitada en cuanto a los artículos de la tienda.

**Bill**

En esta clase se definen los atributos generales que puede tener una factura tales como el monto o la cédula del empleado que realizó la venta o hizo alguna reparación, a su vez es padre de dos clases las cuales representan las facturas de reparaciones y las facturas de ventas de artículos. Esto evita la repetición de código, permite métodos más genéricos y hace el programa más eficiente.

**BillToFix**

Es hija de la clase Bill, posee ciertos atributos que en la clase padre no son contemplados tales como la descripción del daño que tenía el artículo que fue reparado, por esa razón es necesario que sea implementada en el programa.

**BillToSold**

De la misma manera que la clase anterior esta también hereda de la clase Bill, además de los atributos que tiene la clase padre se le agrega una lista de artículos, la cual es para almacenar los artículos que están siendo vendidos en la factura, lo que le permite a la factura tener más de un artículo vendido.

**ListBills**

Esta clase posee una lista tipo Bill que será utilizada para almacenar todas las facturas que se realicen en la tienda, una de las ventajas de utilizar herencia también se ve en esta clase ya que permite guardar los dos tipos de facturas en una sola estructura.

**Employee**

Esta clase se diseña para que represente a un empleado con todos sus respectivos atributos, en la tienda se deben de representar cuatro tipos de empleados, por lo tanto en esta clase es padre de cuatro clases las cuales representan distintos tipos de empleados con distintas funciones cada uno.

**Manager**

Es hija de la clase Employee y posee los mismos atributos pero es necesaria para identificar la función específica que tiene el manager.

**Cashier**

Es hija de la clase Employee y a pesar de que tiene los mismos atributos de la clase es necesaria ya que representa a los cajeros de la tienda y solamente este tipo de empleado se relaciona con las facturas de artículos para vender.

**Maintainer**

Representa a los empleados, es hija de la clase Employee y posee los mismos atributos, fue diseñada ya que este rango de empleados se encargan de hacer las reparaciones de los artículos en mal estado, por lo que esta clase se relaciona con las facturas de reparaciones de artículos.

**Dispenser**

Hereda de la clase Employee, posee los mismos atributos. Es implementada para que represente únicamente las funciones que poseen los dispensadores de la tienda.

**PaySheet**

En esta clase se utiliza para es almacenar todos los empleados, al utilizar herencia la lista soporta cualquier tipo de empleado, lo cual le da mayor eficiencia para la manipulación de datos.

**Payment**

Esta clase representa un pago con sus respectivos atributos, incluyendo la cédula del empleado para saber con quién se debe asociar ese pago, no requiere heredar a hijos ni tampoco es heredada ya que es una clase que no comparte atributos en los que utilizar herencia mejore la eficiencia de la aplicación.

**PaymentHistory**

Esta clase se utiliza para guardar todos los pagos que se han realizado, es necesaria ya que se desea llevar un control de todos los pagos realizados para consultas importantes en el programa.

**Conclusiones**

Fue un proyecto que a ambos miembros nos pareció muy interesante ya que se deben generar muchas clases representando a distintos objetos de la vida real. Mediante el uso correcto de la programación orientada a objetos, a las propiedades de java y la utilización de todo lo aprendido en clases nos fue posible desarrollar la aplicación de manera correcta y al mismo tiempo contribuir al desarrollo de la abstracción para solucionar problemas con soluciones computacionales.