Tarea de entrega Tema 5

Programación de una consigna y el uso de conjuntos

Por: Veronica González Bravo

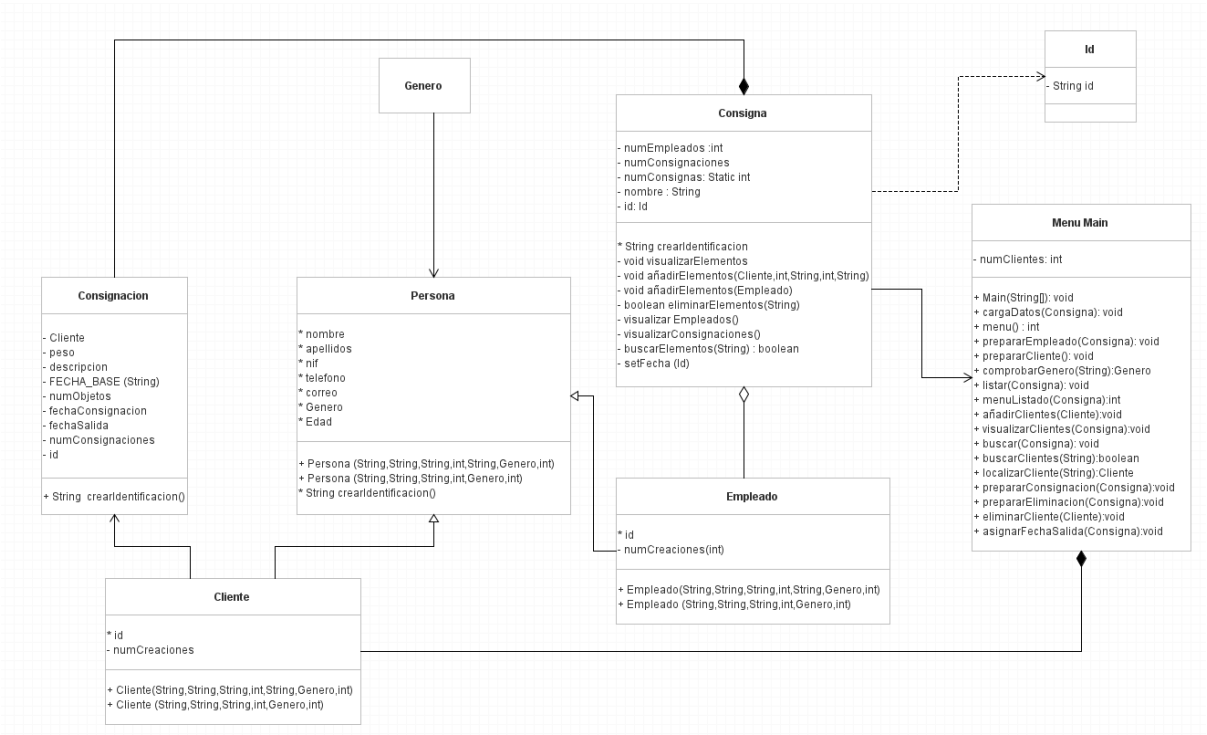
Indice

1 Enunciado del problema

Se necesita de la creación de una base de datos de pequeño calibre para una consigna en una cadena de hoteles, para ello se le asigna a un programador la tarea, las siguientes cuestiones que le pone el empleador sobre la mesa son las siguientes:

- La consigna llevará un registro de los objetos ingresados con la fecha de ingreso y de extracción una vez efectuada.
- Deberá también guardarse los datos del cliente(Cliente previamente registrado) y asociarlos a la consignación que ha realizado.
- Cada consigna tendrá uno o varios empleados enlazados con las instalaciones, podrán ir dejando o ingresando en las instalaciones a lo largo del día, sin embargo, no es necesario indicar la hora en la que han estado trabajando. (se representará mediante la eliminación del elemento en la lista)
- A pesar de que tanto los empleados como los clientes tienen los mismos datos los clientes estarán enlazados con la consignación y los empleados con la Consigna.

2 Esquema de implementación



3 Justificación de diseño

- Clase Consignación: Estará conformada por la id de consignación, el peso de los objetos consignados, el número de los mismos, una descripción y tendrá una fecha de entrada y otra de salida, esta última tendrá el dato por defecto de “aun consignado” hasta que se actualice este atributo, ya que damos por hecho que una vez se crea una consignación esta no puede tener fecha de salida.
  Esta fecha se actualizará una vez el cliente saque el objeto y la consignación quedará registrada dentro del vector de consignaciones de Consigna.

- Persona: Poseerá los datos básicos que cualquier humano tendrá (Nombre, apellidos,teléfono, correo (por defecto pondrá “ninguno”),id. Esta será abstracta y será el padre de empleado y cliente que se diferenciarán por sus respectivas ids.

- Consigna: Tendrá una composición de consignaciones y agregaciones de empleados ya que los empleados podrán trabajar en otras consignas, las gestiones de eliminación, actualización y lecturas estarán implementadas en la clase Main.

- Empleado: Tendrá una id propia para la realización de las búsquedas y eliminaciones, por el resto heredará todos los atributos de persona. Empleado como existe por si mismo y podrá trabajar en otras consignas será una agregación de Consigna.

- Cliente: Al igual que empleado tendrá su propia id, y estará mencionado dentro de cada asignación, será elemento de composición dentro del menú main simulando el registro de clientes desde la red de la Empresa de consignas ya que un cliente existe siempre por si mismo y puede ser cliente de varias consignas. Debido a la naturaleza de las pruebas será una composición dentro de Main, pero en situaciones normales sería una agregación.

- PruebaMain: El menú de pruebas main presenta un método llamado cargaDatos() que cargara los datos iniciales dentro de la consigna, presentará un conjunto de opciones para el usuario donde podrá, crear,visualizar,eliminar y actualizar datos de las diferentes clases (en este caso solo la fecha de salida)
  Entre las funciones básicas están:
  - ☐ Listar: Tendrá un submenú donde escoger si queremos ver los empleados, los clientes, las consignaciones, o, en su defecto, todos los datos
  - ☐ Crear: Permite insertar nuevos objetos. Clientes,Empleados,etc…
  - ☐ Borrar: Permite eliminar un objeto asociado a su id pertinente
  - ☐ Actualizar Fecha: Actualizará la fecha de salida, si ya hay una fecha no dejará.
  - ☐ Buscar: Buscará un objeto introduciendo previamente su id.

- **IDS:** Las ids de cada objeto se declararán mediante statics ,para definir las ids de los clientes y empleados, el número asociado a la id saldrá del aumento de esta static con cada uso del constructor persona o objeto.
  La id se formulara de la siguiente manera:
    - ☐ Empleado: Empezará por la letra E seguida del número que marca la Static, aunque se borre un empleado en la consigna seguirá conservando su id ya que se sobreentiende que podrá trabajar en otra consigna de la empresa..
    - ☐ Cliente: Empezará por la letra C seguido del número que marca su Static.
    - ☐ Consignación: Empezará con la letra O seguido del número que marcará su static, si esta se borra la id no podrá ser asignada a otro objeto.

- **Declaración de statics y constantes**
  - ☐ **Static numCreaciones:** Sirve para dar una id numérica a empleado y cliente, aunque se elimine una entrada está siempre irá in crescendo, asegurando que no se repita la id.
  - ☐ **Static clientes <Cliente>** Se declara un Array list de clientes para establecer el número de clientes disponibles en la franquicia de consignaciones.

4. Justificación de uso de contenedores

Durante la elaboración de esta tarea añadiremos el plus del uso de los contenedores y conjuntos pertinentes, entre ellos una lista, un conjunto y un mapa.

- **Lista de empleados:** Tendremos una lista de empleados, ya que serán tan pocos que la comprobación de errores de redundancia se realizará en el propio código. Será de tipo ArrayList ya que no tendremos más de 5 empleados en lista de manera normal dentro de la base de datos, por lo que será conveniente el uso de este tipo de Lista para almacenarlos.
- **Conjunto de Clientes:** Los clientes podrán llegar a ser cientos con el paso del tiempo, al no estar visible para el usuario de la consigna (se entiende que solo podrá observarlo la centralita que maneja todos los datos de las Consignas) no es necesario que estos estén en orden. Sin embargo si que deberemos evitar la redundancia para evitar que un mismo cliente este registrado dos veces dentro del sistema.
  Entenderemos pues que si un cliente tiene el mismo nif es el mismo, escribiremos un método equals en la clase padre persona y l especificaremos que haga distinción exclusiva del nif.

```
@Override
public int hashCode() {
    return Objects.hash(nif);
}

@Override
public boolean equals(Object obj) {
    System.out.println("Entra");
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Persona other = (Persona) obj;
    return this.nif.compareTo(other.nif)==0;
}
```

  Finalmente escogeremos un hashSet ya que no tendremos que borrar e introducir datos muy a menudo, por lo que será el más eficiente.

- **<u>Mapa de consignaciones;</u>** Generamos un mapa de consignaciones que estará vinculada a una clase ID de consignación, así podremos agilizar las búsquedas y evitar redundancia, de igual manera que en el anterior caso especificaremos que todas las ids vinculadas tienen que ser distintas.

```java
@Override
public int hashCode() {
    return Objects.hash(id);
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Id other = (Id) obj;
    return this.id.compareTo(other.id) == 0;
}
```

De esta manera simplificamos mucho más el código y evitaremos la creación de múltiples clases redundantes. Se barajó la introducción de los clientes como clase llave pero debido a que los clientes pueden tener varias consignaciones vinculadas se descarto.