# ***PayReg***
# Sistema de administración de aportes económicos

Christian Camilo Cuestas Ibáñez[1]
[1]cccuestasi@unal.edu.co,
GitHub: https://github.com/ChrisCuestas/PayReg.git

## I. Definición de la necesidad/problema

Se observa la necesidad de mejorar el sistema de administración de aportes el cual debe cumplir con los siguientes requerimientos generales:

- Administración de las cuentas de recaudo.
- Generación de informes resumidos y exhaustivos de cualquier persona que hace aportes.
- Generación de informes resumidos y exhaustivos del estado general de las cuentas.
- Manejo de directorio de personas y de proveedores.

## II. Definición de la solución

*PayReg* es un sistema que agiliza y facilita la administración de cuentas de recaudo para eventos.

### *II-A. Servicios para las personas aportantes:*

*PayReg* les permite saber para cada evento el detalle de cada uno de sus aportes, el aporte total hasta el momento de la consulta, el saldo pendiente y el costo total.

Cada vez que se hace un aporte, se le envía a través del medio preferido por el aportante (sea Whatsapp, correo o SMS) una notificación que detalla el aporte registrado por los administradores así como también el aporte total hasta el momento y el saldo pendiente.

### *II-B. Funcionalidades:*

- Registro de los aportes en un único documento.
- Crear, gestionar y eliminar cuentas de recaudo.
- Generar informes de resumen de cada cuenta (de recaudo y/o particular).
- Generar y visualizar tablas que contengan la información detallada de cada cuenta de recaudo y particular.
- Crear, editar y eliminar personas.
- Registrar, modificar, cancelar y mover cantidades entre distintas cuentas particulares de una persona.

## III. Diseño del programa

*PayReg* es un programa diseñado sobre le lenguaje de Java.

### *III-A. Arquitectura*

Se pensó en un programa de dos capas: La capa de datos y la capa de interfaz con el usuario.

- *Capa de datos*
  La capa de datos se divide en dos: Una unidad que maneja la lista de personas guardadas y toda su información personal necesaria y otra unidad que maneja todo lo relativo a las cuentas de recaudo.
  *La unidad de cuentas de recaudo* está compuesta por 3 clases:
  - Payment: esta clase de Java es la abstracción de cada uno de los aportes hechos por persona a un evento específico.
  - Event: Esta clase de Java es la abstracción de cada una de las cuentas de recaudo. Ésta clase contiene una estructura de datos de acceso directo que organiza cada aporte hecho a ese evento.
  - EventHandler: Esta clase de Java es la encargada de manejar cada acción permitida sobre los eventos y tenerlos ordenados en una tabla hash.
- *Capa de interfaz con el usuario*
  Esta capa contiene una única clase, la cual permite la comunicación con el usuario desde consola. Está diseñada de forma que el programa guía al usuario en el camino de comandos para lograr hacer operaciones sobre el programa. En esta clase se crean las instancias de las clases EventHandler y PersonHandler, necesarias para toda la dinámica de la información ingresada al programa.

## IV. Descripción detallada de clases

### *IV-A. Payment*

#### *IV-A1. Atributos:*

1. date: Fecha y hora en la cual se efectuó el registro del aporte. Tipo: Date.
2. amount: Cantidad de dinero aportado. Tipo: int.
3. personId: un código de identificación generado por el programa que identifica quién hizo el aporte. Tipo: String

#### *IV-A2. Métodos:*

1. Se implementaron todos los getters y setters.
2. El constructor Payment(Date,int,String).
3. Se redefinió el método toString() heredado de la clase Object.

### *IV-B. Event*

#### *IV-B1. Atributos:*

1. name: Nombre del evento. Tipo: String.
2. numPeople: Cantidad de personas aportando al evento. Tipo: int.
3. cost: Costo total del evento. Tipo: int.
4. totalCollected: Suma de todos los aportes realizados. Tipo: int.
5. isActive: Determina si se puede modificar el evento o no. Tipo: boolean.
6. data: Estructura de datos donde se guardan los aportes realizados. Tipo: Hashtable.
7. totalAditionalPayments: Cantidad de dinero adicional que se aporta al evento. Tipo: int.
8. additionalPayments: Estrutura de datos donde se guardan cada aporte adicional realizado. Tipo: List.

#### *IV-B2. Métodos:*

1. Se implementaron todos los getters y setters.
2. El constructor Event(String,int).
3. getCostByPerson(): (int) Calcula el costo por persona que se debe aportar, teniendo en cuenta el costo total del evento y la cantidad de aportantes.
4. addToCost(int): Le agrega al costo total del evento una cantidad dada. Puede ser positiva o negativa.
5. getRemains(): (int) Obtiene la cantidad de dinero que falta por recaudar.
6. searchPersonPaymentList(String): (Tree) Devuelve la estructura de datos que contiene los aportes hechos por una persona, a partir de el código correspondiente a la persona.
7. addNewPayment(String,Date,int): Crea un nuevo aporte a partir del código de una persona, una fecha y la cantidad de dinero aportado; y lo ingresa a *data*.
8. addNewAdditionalPayment(String,Date,int): Crea un nuevo aporte a partir de la descripción del aporte, una fecha y la cantidad de dinero aportado; y lo ingresa a *additionalPayments*.
9. searchPayment(String,Date): (Payment) Devuelve el aporte que hizo una determinada persona en cierta fecha.
10. searchAdditionalPayment(String): (Payment) Busca el aporte que se hizo que tenga una determinada descripción.
11. searchAdditionalPayment(String): (Payment) Busca el aporte que se hizo que se realizó en determinada fecha.
12. makeList(): (String) Devuelve una cadena de caracteres que muestra la lista de aportes que se han hecho al evento.

### *IV-C. NotifWay*

Una enumeración que contiene:

- SMS: Para envío de mensajes de texto.
- WHATSAPP: Para envío de mensajes por Whatsapp.
- EMAIL: Para envío de mensajes por correo electrónico.

### *IV-D. Person*

#### *IV-D1. Atributos:*

1. id: Código identificador de la persona. Tipo: String.
2. firstName: Nombre de pila de la persona. Tipo: String.
3. lastName1: Primer apellido de la persona. Tipo: String.
4. lastName2: Segundo apellido de la persona. Tipo: String.
5. phone: Número de teléfono o celular de la persona. Tipo: long.
6. eMail: Correo electrónico de la persona. Tipo: String.
7. notifWay: Especifica la forma como se le va a notificar a la persona sobre sus aportes. Tipo: NotifWay.

#### *IV-D2. Métodos:*

1. Se implementaron todos los getters y setters.
2. El constructor Person(String,String,String, String,long,String,String).
3. Se redefinió el método toString() heredado de la clase Object.

### *IV-E. PersonHandler*

#### *IV-E1. Atributos:*

1. size: Cantidad de personas existentes. Tipo: int.

2. ids: Lista de los códigos identificadores de las personas. Tipo: List.
3. people: Estructura de datos que guarda a cada persona creada. Tipo: Tree.

*IV-E2. Métodos:*

1. Se implementaron todos los getters y setters.
2. El constructor PersonHandler().
3. addNewPerson(String,String,String,long,String,String): (Person)
4. buildId(String,String,String): (String)
5. deletePerson(String): (Person)
6. deletePerson(String,String,String): (Person)
7. Se redefinió el método toString() heredado de la clase Object.

### *IV-F. EventHandler*

*IV-F1. Atributos:*

1. size: Cantidad de eventos existentes. Tipo: int.
2. names: Lista de los nombres de los eventos. Tipo: List.
3. events: Estructura de datos que guarda los eventos. Tipo: Hashtable.
4. people: El PersonHandler correpondiente. Tipo: PersonHandler.

*IV-F2. Métodos:*

1. Se implementaron todos los getters y setters.
2. El constructor EventHandler().
3. addNewEvent(String,int): (int)
4. getEvent(int): (Event)
5. searchEvent(String): (Event)
6. deleteEvent(int): (Event)
7. idOfEvent(String): (int)
8. Se redefinió el método toString() heredado de la clase Object.