Tema: **Gestión de avance curricular de los alumnos en un instituto**

Nombres: **Sebastian Jeria**
Vicente Montiel
Luciano Cubillos

1.1 Realizar un análisis de los datos a utilizar y principales funcionalidades a implementar que dan sentido a la realización del proyecto.

El proyecto consiste en gestionar el avance curricular de los estudiantes de un instituto, permitiendo que se ingresen los estudiantes de una carrera y las asignaturas aprobadas por cada uno de ellos. Por lo que contaremos con la información de cada estudiante, ya sea académica y personal.

1.2 Diseño conceptual de clases del Dominio y su código en Java

- **Estudiante:** Esta clase contiene los datos de los estudiantes del instituto, sus atributos son: el rut, nombre, año de ingreso y una lista con las asignaturas aprobadas.
- **Asignatura:** Esta clase contiene los datos de las asignaturas, sus atributos son : nombre asignatura, id asignatura.

1.3 Todos los atributos de todas las clases deben ser privados y poseer sus respectivos métodos de lectura y escritura (getter y setter).

En el proyecto, se cumple el requisito de que todos los atributos de cada clase sean privados, y poseen los métodos de lectura y escritura, los que se mostraran a continuación:

```
public class Asignatura {
    private String nombreAsignatura;
    private String idAsignatura;

    public Asignatura(String nombreAsignatura, String idAsignatura) {
        this.nombreAsignatura = nombreAsignatura;
        this.idAsignatura = idAsignatura;
    }

    public Asignatura(){
        this.nombreAsignatura = null;
        this.idAsignatura = null;
    }

    public String getNombreAsignatura() {
        return nombreAsignatura;
    }

    public void setNombreAsignatura(String nombreAsignatura) {
        this.nombreAsignatura = nombreAsignatura;
    }

    public String getIdAsignatura() {
        return idAsignatura;
    }

    public void setIdAsignatura(String idAsignatura) {
        this.idAsignatura = idAsignatura;
    }

}
```

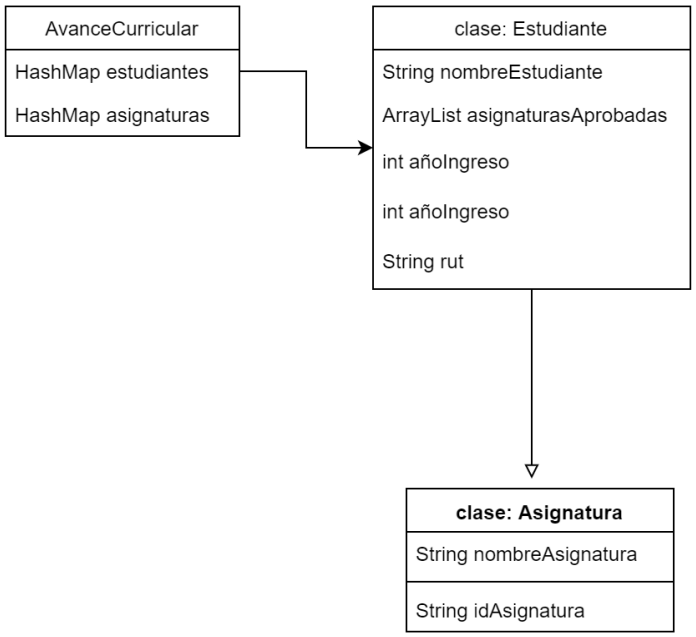
```java
public class Estudiante {
    private String nombreEstudiante;
    private ArrayList<Asignatura> asignaturasAprobadas = new ArrayList();
    private String añoIngreso;
    private String rut;

    public Estudiante(String nombreEstudiante, String añoIngreso, String rut) {
        this.nombreEstudiante = nombreEstudiante;
        this.añoIngreso = añoIngreso;
        this.rut = rut;
    }

    public Estudiante() {
        this.nombreEstudiante = null;
        this.añoIngreso = null;
        this.rut = null;
    }

    public ArrayList<Asignatura> getAsignaturasAprobadas() {
        return asignaturasAprobadas;
    }

    public void setAsignaturasAprobadas(ArrayList<Asignatura> asignaturasAprobadas) {
        this.asignaturasAprobadas = asignaturasAprobadas;
    }

    public String getNombreEstudiante() {
        return nombreEstudiante;
    }

    public void setNombreEstudiante(String nombreEstudiante) {
        this.nombreEstudiante = nombreEstudiante;
    }

    public String getAñoIngreso() {
        return añoIngreso;
    }

    public void setAñoIngreso(String añoIngreso) {
        this.añoIngreso = añoIngreso;
    }

    public String getRut() {
        return rut;
    }

    public void setRut(String rut) {
        this.rut = rut;
    }

    public void agregarAsignaturas(String nombre, String id){
        Asignatura asignatura = new Asignatura( nombreAsignatura: nombre,  idAsignatura: id);
        asignaturasAprobadas.add( e: asignatura);
    }
}
```

1.4 Se deben incluir datos iniciales dentro del código

El código contiene datos iniciales, los que son ingresados cargando archivos .csv, que contienen datos de los estudiantes, específicamente, sus datos personales y las asignaturas subidas al sistema.

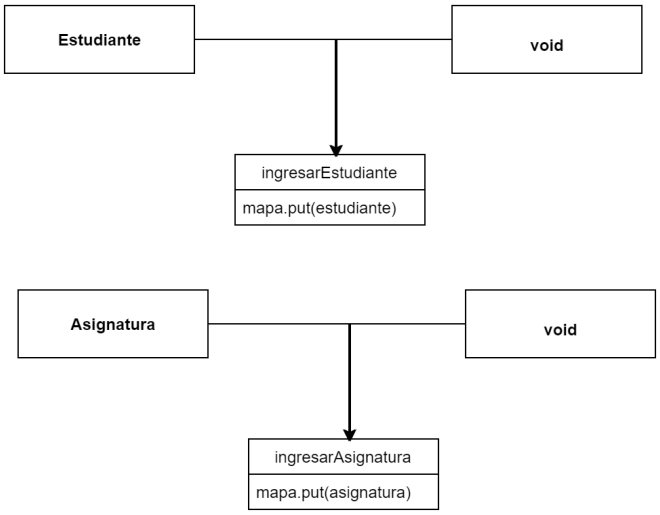
1.5 Diseño conceptual y codificación de 2 colecciones de objetos, con la 2ª colección anidada como muestra la figura. Las colecciones pueden ser implementadas mediante arreglos o clases del Java Collections Framework (JCF).

El proyecto incluye la codificación de 2 colecciones de objetos y la segunda colección anidada. En nuestro caso, vamos a contar con una de estudiantes y una colección de asignaturas aprobadas por estos mismos estudiantes.



1.6 Diseño conceptual y codificación de 2 clases que utilicen sobrecarga de métodos (no de constructores)

Se realizó la sobrecarga en los métodos de ingresarEstudiante e ingresarAsignatura

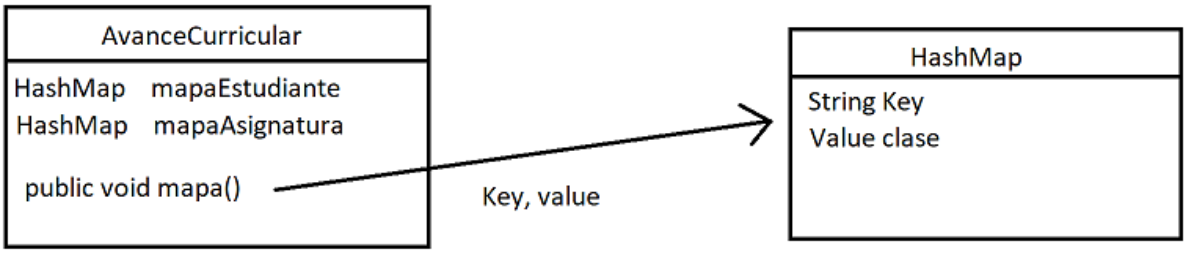



```
public void ingresarEstudiante(Estudiante estudiante) throws IOException{
    mapaEstudiante.put( key:estudiante.getRut(),  value:estudiante);
}

public void ingresarEstudiante() throws IOException{
    String ruta = "alumnos.csv";
    try (FileWriter fw = new FileWriter( fileName:ruta,  append:true)){
        System.out.println( x:"Ingrese rut: ");
        String rut = hh.readLine();
        System.out.println( x:"Ingrese nombre: ");
        String nombreE = hh.readLine();
        System.out.println( x:"Ingrese año de ingreso: ");
        String año = hh.readLine();
        Estudiante estudiante = new Estudiante( nombreEstudiante:nombreE,  añoIngreso:año, rut);
        mapaEstudiante.put( key:rut,  value:estudiante);
        fw.append( csq:"\n"); // salto de línea para agregar los datos en una nueva línea
        fw.append( csq:rut).append( csq:",");
        fw.append( csq:nombreE).append( csq:",");
        fw.append( csq:año);
    }catch (IOException e) {
     System.out.println( x:"Error");
  }
}
```

```java
public void ingresarAsignatura(Asignatura asignatura){
    mapaAsignatura.put( key:asignatura.getIdAsignatura(), value:asignatura);
}

public void ingresarAsignatura() throws IOException{
    String ruta = "asignaturas.csv";
    try (FileWriter fw = new FileWriter( fileName:ruta, append:true)){
        System.out.println( x:"Ingrese nombre asignatura: ");
        String nombreA = hh.readLine();
        System.out.println( x:"Ingrese id: ");
        String id = hh.readLine();
        Asignatura asignatura = new Asignatura( nombreAsignatura:nombreA, idAsignatura:id);
        mapaAsignatura.put( key:id, value:asignatura);
        fw.append( csq:"\n"); // salto de línea para agregar los datos en una nueva línea
        fw.append( csq:nombreA).append( csq:",");
        fw.append( csq:id);
    }catch (IOException e) {
     System.out.println( x:"Error");
   }
}
```

1.7 Diseño conceptual y codificación de al menos 1 clase mapa del Java Collections Framework

Se incluyeron mapas del Java Collections Framework en el código, por ejemplo un HashMap para guardar datos de Estudiante y otro para guardar asignaturas, tal como lo muestra la figura siguiente:



Además, la siguiente figura muestra cómo se implementó en el código:

```java
public class CopiaAvanceCurricular {

    private HashMap <String,Estudiante> mapaEstudiante = new HashMap();
    private HashMap <String,Asignatura> mapaAsignatura = new HashMap();
    BufferedReader hh = new BufferedReader(new InputStreamReader( in:System.in));

    public static void main(String[] args) throws IOException {
```

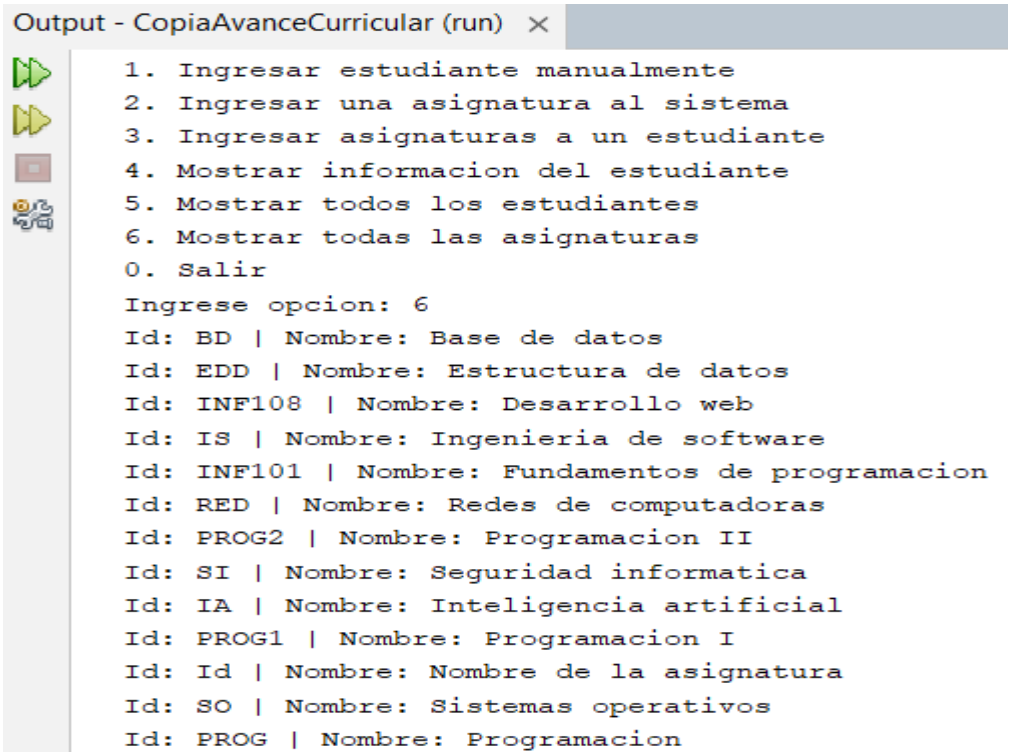
```java
public void ingresarEstudiante(Estudiante estudiante) throws IOException{
    mapaEstudiante.put( key:estudiante.getRut(),  value:estudiante);
}

public void ingresarEstudiante() throws IOException{
    String ruta = "alumnos.csv";
    try (FileWriter fw = new FileWriter( fileName:ruta,  append:true)){
        System.out.println( x:"Ingrese rut: ");
        String rut = hh.readLine();
        System.out.println( x:"Ingrese nombre: ");
        String nombreE = hh.readLine();
        System.out.println( x:"Ingrese año de ingreso: ");
        String año = hh.readLine();
        Estudiante estudiante = new Estudiante( nombreEstudiante:nombreE,  añoIngreso:año, rut);
        mapaEstudiante.put( key:rut,  value:estudiante);
        fw.append( csq:"\n"); // salto de línea para agregar los datos en una nueva línea
        fw.append( csq:rut).append( csq:",");
        fw.append( csq:nombreE).append( csq:",");
        fw.append( csq:año);
    }catch (IOException e) {
     System.out.println( x:"Error");
  }
}
```

```java
public void ingresarAsignatura(Asignatura asignatura){
    mapaAsignatura.put( key:asignatura.getIdAsignatura(),  value:asignatura);
}

public void ingresarAsignatura() throws IOException{
    String ruta = "asignaturas.csv";
    try (FileWriter fw = new FileWriter( fileName:ruta,  append:true)){
        System.out.println( x:"Ingrese nombre asignatura: ");
        String nombreA = hh.readLine();
        System.out.println( x:"Ingrese id: ");
        String id = hh.readLine();
        Asignatura asignatura = new Asignatura( nombreAsignatura:nombreA,  idAsignatura:id);
        mapaAsignatura.put( key:id,  value:asignatura);
        fw.append( csq:"\n"); // salto de línea para agregar los datos en una nueva línea
        fw.append( csq:nombreA).append( csq:",");
        fw.append( csq:id);
    }catch (IOException e) {
     System.out.println( x:"Error");
  }
}
```

1.8 Se debe hacer un menú para el Sistema donde ofrezca las funcionalidades de: 1) Inserción Manual / agregar elemento y 2) Mostrar por pantalla listado de elementos. Esto para la 2ª colección de objetos (colección anidada) del SIA1.5

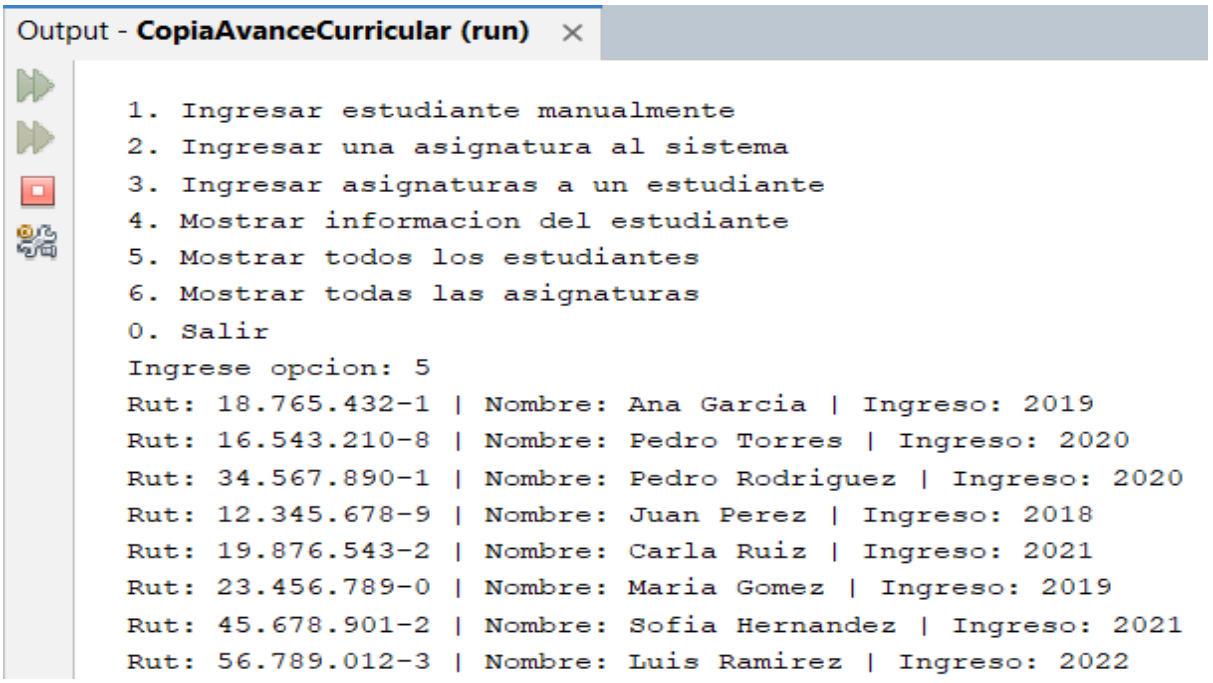
Las siguientes figuras muestran el menú y las funcionalidades requeridas de ingresarEstudiante, donde se puede ingresar manualmente un estudiante nuevo. Y el de ingresarAsignatura y agregarAsignaturaEstudiante, donde se puede ingresar una nueva asignatura al sistema y agregarle una asignatura aprobada a un estudiante anteriormente creado.

```
while(salir){
    System.out.println( x: "1. Ingresar estudiante manualmente");
    System.out.println( x: "2. Ingresar una asignatura al sistema");
    System.out.println( x: "3. Ingresar asignaturas a un estudiante");
    System.out.println( x: "4. Mostrar informacion del estudiante");
    System.out.println( x: "5. Mostrar todos los estudiantes");
    System.out.println( x: "6. Mostrar todas las asignaturas");
    System.out.println( x: "0. Salir");
    System.out.print( s: "Ingrese opcion: ");
    opcion = Integer.parseInt( s: leer.readLine());
    switch (opcion) {
        case 1:{
            ac.ingresarEstudiante();
            break;
        }
        case 2:{
            ac.ingresarAsignatura();
            break;
        }
        case 3:{
           ac.agregarAsignaturaEstudiante();
           break;
        }
        case 4:{
            ac.mostrarDatosEstudiante();
            break;
        }
        case 5:{
            ac.mostrarEstudiantes();
            break;
        }
        case 6:{
            ac.mostrarAsignaturas();
            break;
        }
        case 0:{
            System.out.println( x: "Programa finalizado.");
            salir = false;
            break;
        }
```



1.9 Todas las funcionalidades deben ser implementadas mediante consola (Sin ventanas)

Como se puede ver en la siguiente figura, todas las funcionalidades son implementadas mediante consola.



1.10 Utilización de GitHub (Realización de al menos 3 Commit)

Se realizan los commit requeridos.

https://github.com/seba-jeria/AvanceCurricular.git