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

Nombres: **Sebastian Jeria Lopez** **21.024.969-9**
Vicente Montiel **21.175.006-5**
Luciano Cubillos **20.211.723-6**

Para instalar y ejecutar el proyecto hay que seguir los siguientes pasos:

1. Descargar el proyecto (si es .zip, hay que extraerla).
2. Abrir el programa Apache NetBeans.
3. Cargar el proyecto.
4. Es probable que haya una advertencia al no coincidir la versión del jdk utilizado, pero puedes cerrar esas pestañas y continuar.
5. Click en la pestaña RUN y seleccionar "RUN PROJECT" o "(F6)"

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 y vigentes por cada uno de ellos. Por lo que contaremos con la información de cada estudiante, ya sea académica y personal.

Contaremos con los siguientes datos:

1. Carrera : Es la carrera en la cuál se basa el programa, recibe el nombre de la carrera, además de tener un mapa de alumnos, un mapa de asignaturas, un arraylist de alumnos y un arraylist de asignaturas.
2. Estudiante : Se refiere a cada estudiante perteneciente a la carrera, la cual se basará en recibir nombre, rut, año de ingreso, asignaturas vigentes y asignaturas aprobadas.
3. Asignatura : Es cada asignatura perteneciente a la carrera y al estudiante que las cursa o que las aprobó. Esta recibirá el nombre de la asignatura, su id y además la nota de un alumno en esa asignatura.

Las principales funcionalidades son:

1. Añadir estudiante manualmente al sistema: Ingresará manualmente por teclado un alumno en un mapa y lo guardará en un csv específico de alumnos.
2. Añadir asignatura manualmente al sistema : Similar a la anterior, se ingresará manualmente por teclado una asignatura al mapa y se guardará en un csv específico de asignaturas.
3. Ingresar asignatura aprobada a un estudiante: Asignará una asignatura aprobada al estudiante, mediante la clave del estudiante la cual será el rut.
4. Ingresar asignatura vigente a un estudiante: Asignará una asignatura vigente al estudiante, mediante la clave del estudiante la cual será el rut.
5. Modificar nota de una asignatura aprobada: Buscamos la asignatura, luego buscamos el estudiante al que le vamos a modificar la nota y si existe la modificamos.
6. Eliminar asignatura vigente: Buscamos la asignatura de un estudiante mediante su rut y la id de esta, para luego eliminarla.

7. Mostrar todos los estudiantes: Mostrará todos los estudiantes contenidos en el csv específico de estudiantes.
8. Mostrar todas las asignaturas: Mostrará todas las asignaturas contenidas en el csv específico de asignaturas.
9. Mostrar todas las asignaturas aprobadas de un estudiante: Mostrar todas las asignaturas aprobadas del estudiante.
10. Mostrar información del estudiante: Mostrará los datos del estudiante, además se mostrará las asignaturas que tenga aprobadas.
11. Mostrar alumnos por nota mayor o igual: Mostrará todos los estudiantes, ya sean, titulados o vigentes por la nota mínima ingresada.
12. Mostrar alumnos de una generación (anno ingreso): Mostrará todos los estudiantes, ya sean, titulados o vigentes por el año ingresado.

```
System.out.println( x: "1. Añadir estudiante manualmente al sistema");
System.out.println( x: "2. Añadir una asignatura manualmente al sistema");
System.out.println( x: "3. Ingresar asignatura aprobada a un estudiante");
System.out.println( x: "4. Ingresar asignatura vigente a un estudiante");
System.out.println( x: "5. Modificar nota de una asignatura aprobada");
System.out.println( x: "6. Eliminar asignatura vigente");
System.out.println( x: "7. Mostrar todos los estudiantes");
System.out.println( x: "8. Mostrar todas las asignaturas");
System.out.println( x: "9. Mostrar todos las asignaturas vigentes de un estudiante");
System.out.print( s: "10. Mostrar informacion del estudiante");
System.out.println( x: " ( coleccion de asignaturas aprobadas )");
System.out.println( x: "11. Mostrar alumnos por nota mayor o igual");
System.out.println( x: "12. Mostrar alumnos de una generacion (anno ingreso)");
System.out.println( x: "0. Salir");
System.out.print( s: "Ingrese opcion: ");
```

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

Nuestras clases son:

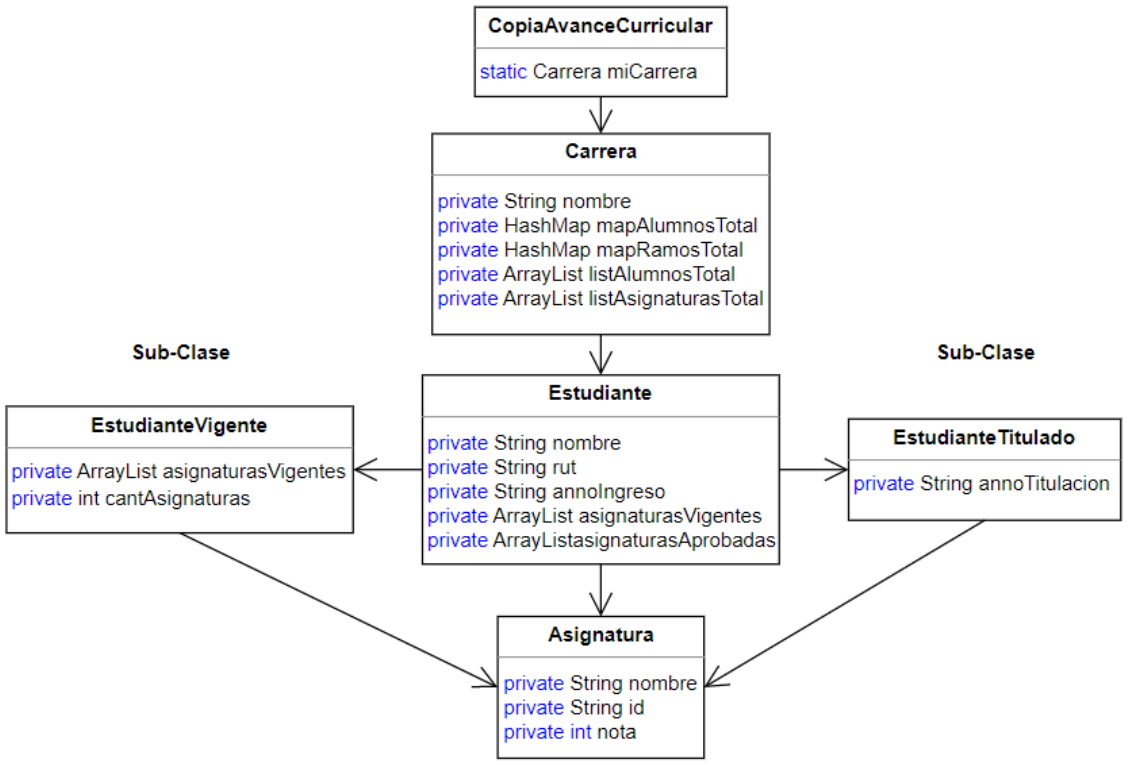
- **Estudiante:** Esta clase contiene los datos de los estudiantes del instituto, sus atributos son: el rut, nombre, año de ingreso, una lista con las asignaturas aprobadas y vigentes.

  Además contaremos con subclases de Estudiante, llamadas:

  - EstudianteVigente: Contiene una colección ArrayList de sus asignaturas que cursa actualmente, además de un contador con el total de asignaturas vigentes.
  - EstudianteTitulado: Contiene el año en el cuál el estudiante egresó de la universidad.
- **Asignatura:** Esta clase contiene los datos de las asignaturas, sus atributos son : nombre asignatura, id asignatura y nota.
- **Carrera:** Esta clase contiene mapas con todas las asignaturas y estudiantes contenidos en la carrera, además de listas para recorrer los datos.
- Además se crean las clases que indican excepciones en el programa, las que serán indicadas más en profundidad más adelante.

En la siguiente figura se puede apreciar de mejor manera las clases que utilizamos y cómo se relacionan entre ellas, aunque serán detalladas más adelante en el diagrama UML.

También cabe destacar que para coleccionar los estudiantes y asignaturas que componen la carrera usamos datos tipo ArrayList y HashMap




```java
public class Carrera {
    private String nombre;
    private HashMap <String, Estudiante> mapAlumnosTotal = new HashMap();
    private HashMap <String, Asignatura> mapRamosTotal = new HashMap();
    private ArrayList  listAlumnosTotal = new ArrayList();
    private ArrayList  listAsignaturasTotal = new ArrayList();

    public Carrera(){
        nombre = null;
        mapAlumnosTotal = new HashMap();
        mapRamosTotal = new HashMap();
        listAlumnosTotal = new ArrayList();
        listAsignaturasTotal = new ArrayList();
    }
    public Carrera(String nombre) {
        this.nombre = nombre;
        mapAlumnosTotal = new HashMap();
        mapRamosTotal = new HashMap();
        listAlumnosTotal = new ArrayList();
        listAsignaturasTotal = new ArrayList();
    }
```

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



    public Estudiante(String nombreEstudiante, String annoIngreso, String rut) {
        this.nombre = nombreEstudiante;
        this.annoIngreso = annoIngreso;
        this.rut = rut;
        asignaturasAprobadas = new ArrayList();
    }
    public Estudiante() {
        this.nombre = null;
        this.annoIngreso = null;
        this.rut = null;
        asignaturasAprobadas = new ArrayList();
    }
```

```java
public class EstudianteVigente extends Estudiante{
    private ArrayList <Asignatura> asignaturasVigentes = new ArrayList();
    private int cantAsignaturas;

    public EstudianteVigente(String nombreEstudiante, String annoIngreso, String rut){
        super(nombreEstudiante,  annoIngreso,  rut);
        asignaturasVigentes = new ArrayList();
        cantAsignaturas = 0;
    }
    public EstudianteVigente(){
        super();
        asignaturasVigentes = new ArrayList();
        cantAsignaturas = 0;
    }
```

```java
public class EstudianteTitulado extends Estudiante{
    private String annoTitulacion;

    public EstudianteTitulado(String annoTitulacion, String nombreEstudiante, String annoIngreso, String rut) {
        super(nombreEstudiante, annoIngreso, rut);
        this.annoTitulacion = annoTitulacion;
    }

    public EstudianteTitulado() {
        super();
        this.annoTitulacion = super.getAnnoIngreso() + 6;
    }
```

```java
public class Asignatura {
    private String nombre;
    private String id;
    private int nota;

    public Asignatura(String nombreAsignatura, String idAsignatura, int nota) {
        this.nombre = nombreAsignatura;
        this.id = idAsignatura;
        this.nota = nota;
    }
    public Asignatura(String nombreAsignatura, String idAsignatura) {
        this.nombre = nombreAsignatura;
        this.id = idAsignatura;
        this.nota = 40;
    }

    public Asignatura(){
        this.nombre = null;
        this.id = null;
        this.nota = 1;
    }
```

```java
public class NotaException extends Exception{
    public NotaException(){
        super( message: "Nota no valida");
    }
}

public class RutException extends Exception{
    public RutException(){
        super( message: "Rut no valido");
    }
}
```

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:

```java
public class Asignatura {
    private String nombre;
    private String id;
    private int nota;

    public Asignatura(String nombreAsignatura, String idAsignatura, int nota) {
        this.nombre = nombreAsignatura;
        this.id = idAsignatura;
        this.nota = nota;
    }
    public Asignatura(String nombreAsignatura, String idAsignatura) {
        this.nombre = nombreAsignatura;
        this.id = idAsignatura;
        this.nota = 40;
    }

    public Asignatura(){
        this.nombre = null;
        this.id = null;
        this.nota = 1;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public int getNota() {
        return nota;
    }

    public void setNota(Double nota){
        double redondeado = Math.round(nota * 10) / 10.0;

        if (redondeado < 10) {
            redondeado = redondeado * 10;
            redondeado = (int) redondeado;
        } else {
            redondeado = (int) redondeado;
        }

        int numeroEntero = (int) redondeado;
        this.nota = numeroEntero;
    }
}
```

```java
public class Carrera {
    private String nombre;
    private HashMap <String, Estudiante> mapAlumnosTotal = new HashMap();
    private HashMap <String, Asignatura> mapRamosTotal = new HashMap();
    private ArrayList  listAlumnosTotal = new ArrayList();
    private ArrayList  listAsignaturasTotal = new ArrayList();

    public Carrera(){
        nombre = null;
        mapAlumnosTotal = new HashMap();
        mapRamosTotal = new HashMap();
        listAlumnosTotal = new ArrayList();
        listAsignaturasTotal = new ArrayList();
    }
    public Carrera(String nombre) {
        this.nombre = nombre;
        mapAlumnosTotal = new HashMap();
        mapRamosTotal = new HashMap();
        listAlumnosTotal = new ArrayList();
        listAsignaturasTotal = new ArrayList();
    }


    public String getNombre() {return nombre;}
    public void setNombre(String nombre) {this.nombre = nombre;}



public class EstudianteVigente extends Estudiante{
    private ArrayList <Asignatura> asignaturasVigentes = new ArrayList();
    private int cantAsignaturas;

    public EstudianteVigente(String nombreEstudiante, String annoIngreso, String rut){
        super(nombreEstudiante,  annoIngreso,  rut);
        asignaturasVigentes = new ArrayList();
        cantAsignaturas = 0;
    }
    public EstudianteVigente(){
        super();
        asignaturasVigentes = new ArrayList();
        cantAsignaturas = 0;
    }

    public int getCantAsignaturas() {
        return cantAsignaturas;
    }

    public void setCantAsignaturas(int cantAsignaturas) {
        this.cantAsignaturas = cantAsignaturas;
    }
```

```java
public class EstudianteTitulado extends Estudiante{
    private String annoTitulacion;

    public EstudianteTitulado(String annoTitulacion, String nombreEstudiante, String annoIngreso, String rut) {
        super(nombreEstudiante, annoIngreso, rut);
        this.annoTitulacion = annoTitulacion;
    }

    public EstudianteTitulado() {
        super();
        this.annoTitulacion = super.getAnnoIngreso() + 6;
    }

    public String getAnnoTitulacion() {
        return annoTitulacion;
    }

    public void setAnnoTitulacion(String annoTitulacion) {
        this.annoTitulacion = annoTitulacion;
    }
public class Estudiante {
    private String nombre;
    private String rut;
    private String annoIngreso;
    private ArrayList<Asignatura> asignaturasAprobadas = new ArrayList();



    public Estudiante(String nombreEstudiante, String annoIngreso, String rut) {
        this.nombre = nombreEstudiante;
        this.annoIngreso = annoIngreso;
        this.rut = rut;
        asignaturasAprobadas = new ArrayList();
    }
    public Estudiante() {
        this.nombre = null;
        this.annoIngreso = null;
        this.rut = null;
        asignaturasAprobadas = new ArrayList();
    }

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getRut() {
        return rut;
    }
    public void setRut(String rut) {
        this.rut = rut;
    }
    public String getAnnoIngreso() {
        return annoIngreso;
    }
    public void setAnnoIngreso(String annoIngreso) {
        this.annoIngreso = annoIngreso;
    }
```

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.

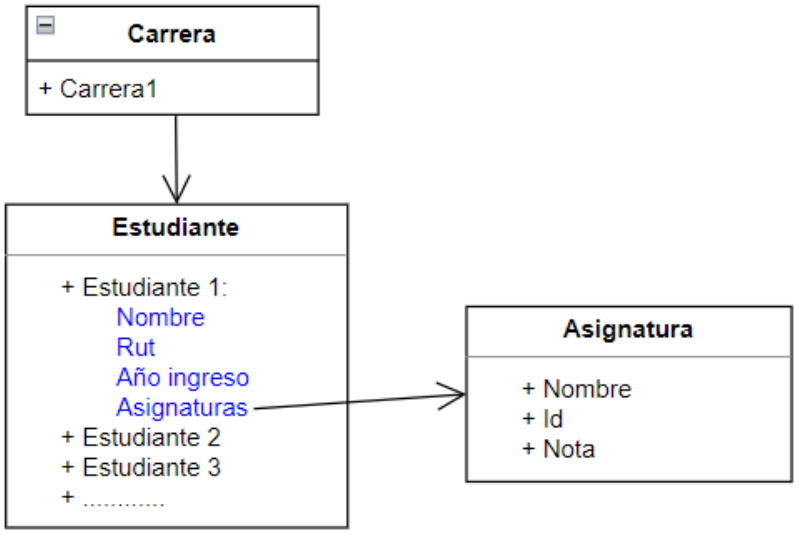
```
//funcion que lee los alumnos de un csv y los carga en un mapa
public static  void leerAlumno() throws RutException{
    String ruta = "alumnos.csv";
    try (BufferedReader br = new BufferedReader(new FileReader(ruta))) {
        String linea;
        while ((linea = br.readLine()) != null) {
            String[] datos = linea.split(",");
            EstudianteVigente estudiante = new EstudianteVigente();
            int i = 0;

            for (String dato : datos) {
               i++;

               switch(i){
                case 1:{
                    estudiante.setRut(dato);
                }
                case 2:{
                    estudiante.setNombre(dato);
                }
                case 3:{
                    estudiante.setAnnoIngreso(dato);
                }
               }
            }
            //System.out.println(); // salto de línea para la siguiente fila
            miCarrera.agregarEstudiante(estudiante);
        }
    } catch (IOException e) {
        System.out.println("Error");
    }
}
```

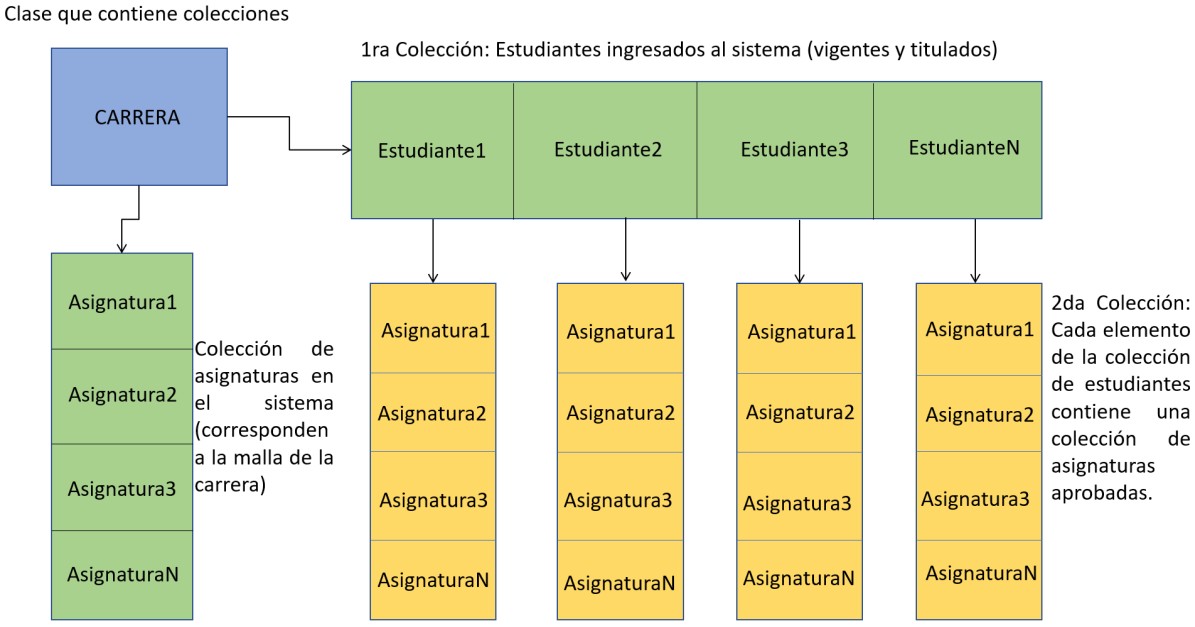
```java
//funcion que lee las asignaturas de un csv y los carga en un mapa
public static  void leerAsignatura() {
    String ruta = "asignaturas.csv";
    try (BufferedReader br = new BufferedReader(new FileReader(ruta))) {
        String linea;

        while ((linea = br.readLine()) != null) {
           String[] datos = linea.split(",");
           Asignatura asignatura = new Asignatura();
           int i = 0;

           for (String dato : datos) {
              i++;

              switch(i){
                  case 1:{
                      asignatura.setNombre(dato);
                  }
                  case 2:{
                      asignatura.setId(dato);
                  }
              }
           }
           //System.out.println(); // salto de linea para la siguiente fila
           miCarrera.annadirRamo(asignatura);
       }
    } catch (IOException e) {
        System.out.println("Error");
    }
}
```

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.

Como se puede apreciar en ambas figuras, hay 2 colecciones anidadas, las que son requeridas para el proyecto. Además si un estudiante pertenece a la clase EstudianteVigente, se vincula con otra colección extra, que guarda sus asignaturas que cursa actualmente.

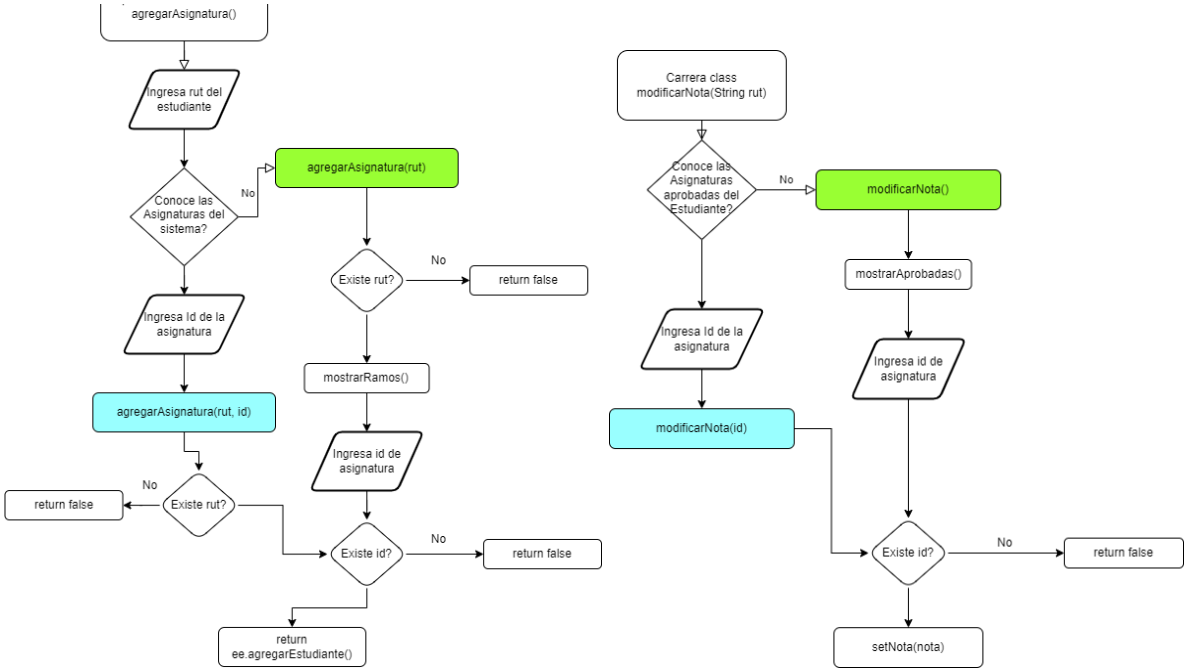




1.6 Diseño conceptual y codificación de 2 clases que utilicen sobrecarga de métodos (no de constructores)

Se realizó la sobrecarga en los métodos de agregarAsignatura y agregarAsignaturaVigente en la clase Carrera, la que agregan una asignatura aprobada o vigente respectivamente al estudiante, se hace la sobrecarga de modo que si no se conocen las asignaturas totales de la carrera (que contiene el sistema) se pueden mostrar por pantalla, y en el caso contrario no se muestran.

También se hace sobrecarga en la clase Estudiante del método modificarNota, el cual opera bajo la misma lógica, ya que si necesitan ver las asignaturas del estudiante se pueden mostrar por pantalla e ingresar el id, o ingresar el id sin que se muestran las asignaturas por pantalla en el caso contrario.

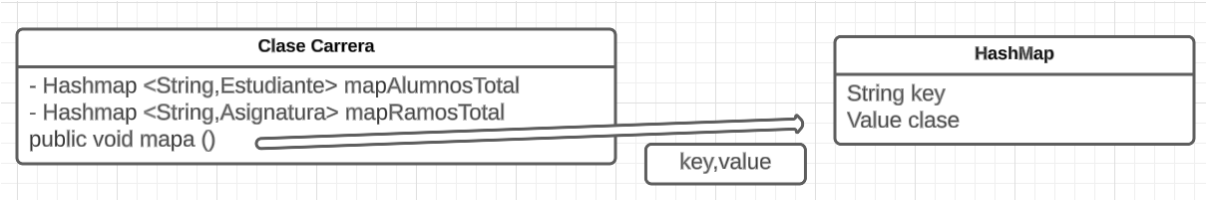


```java
public boolean modificarNota(String id) throws IOException, NotaException{
    BufferedReader leer = new BufferedReader(new InputStreamReader(System.in));
    Asignatura aa = buscarAsignatura(id);
    if (aa != null){
        System.out.println("Ingrese nota:");
        Double nota = Double.valueOf(leer.readLine());

        if((nota>=1.0) && (nota<=7.0)){
            aa.setNota(nota);
             return true;
        }
        else{
           if ((nota>=10) && (nota<=70)){
                aa.setNota(nota);
                return true;
           }
           throw new NotaException();
        }
    }
    return false;
}
public boolean modificarNota() throws IOException, NotaException{
    BufferedReader leer = new BufferedReader(new InputStreamReader(System.in));
    mostrarAprobadas();
    System.out.println("Ingrese id de la Asignatura:");
    String id = leer.readLine();

    Asignatura aa = buscarAsignatura(id);

    if (aa != null){
        System.out.println("Ingrese nota:");
        Double nota = Double.valueOf(leer.readLine());

        if((nota>=1.0) && (nota<=7.0)){
            aa.setNota(nota);
             return true;
        }
        else{
           if ((nota>=10) && (nota<=70)){
                aa.setNota(nota);
                return true;
           }
           throw new NotaException();
        }
    }
    return false;
}
```

```java
public boolean agregarAsignatura(String rut, String id){
    EstudianteVigente ee = (EstudianteVigente)buscarEstudiante(rut);
    Asignatura aa = buscarAsignatura(id);
    if(ee == null){
        return false;
    }
    if(aa == null){
        return false;
    }
    if(ee.agregarAsignaturaAprobada(id)){
        return ee.agregarAsignatura(aa.getNombre(), id, aa.getNota());
    }
    return false;

}
public boolean agregarAsignatura(String rut)throws IOException{
    EstudianteVigente ee = (EstudianteVigente)buscarEstudiante(rut);
    if(ee == null){
        return false;
    }
    BufferedReader leer = new BufferedReader(new InputStreamReader(System.in));
    mostrarRamos();

    System.out.println("Ingrese ID de la asignatura aprobada: ");
    String id = leer.readLine();

    if(buscarAsignatura(id) == null){
        return false;
    }
    Asignatura aa = buscarAsignatura(id);
    return ee.agregarAsignatura(aa.getNombre(), aa.getId(), aa.getNota());
}
```

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:

```
public class Carrera {
    private String nombre;
    private HashMap <String, Estudiante> mapAlumnosTotal = new HashMap();
    private HashMap <String, Asignatura> mapRamosTotal = new HashMap();
    private ArrayList  listAlumnosTotal = new ArrayList();
    private ArrayList  listAsignaturasTotal = new ArrayList();
```

Utilizamos los ArrayList para recorrer los datos, conservan sincronización igualmente.

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;
        }
```

```
Output - CopiaAvanceCurricular (run)
    1. Ingresar estudiante manualmente
    2. Ingresar una asignatura al sistema
    3. Ingresar asignaturas a un estudiante
    4. Mostrar informacion del estudiante
    5. Mostrar todos los estudiantes
    6. Mostrar todas las asignaturas
    0. Salir
    Ingrese opcion: 6
    Id: BD | Nombre: Base de datos
    Id: EDD | Nombre: Estructura de datos
    Id: INF108 | Nombre: Desarrollo web
    Id: IS | Nombre: Ingenieria de software
    Id: INF101 | Nombre: Fundamentos de programacion
    Id: RED | Nombre: Redes de computadoras
    Id: PROG2 | Nombre: Programacion II
    Id: SI | Nombre: Seguridad informatica
    Id: IA | Nombre: Inteligencia artificial
    Id: PROG1 | Nombre: Programacion I
    Id: Id | Nombre: Nombre de la asignatura
    Id: SO | Nombre: Sistemas operativos
    Id: PROG | Nombre: Programacion
```

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.

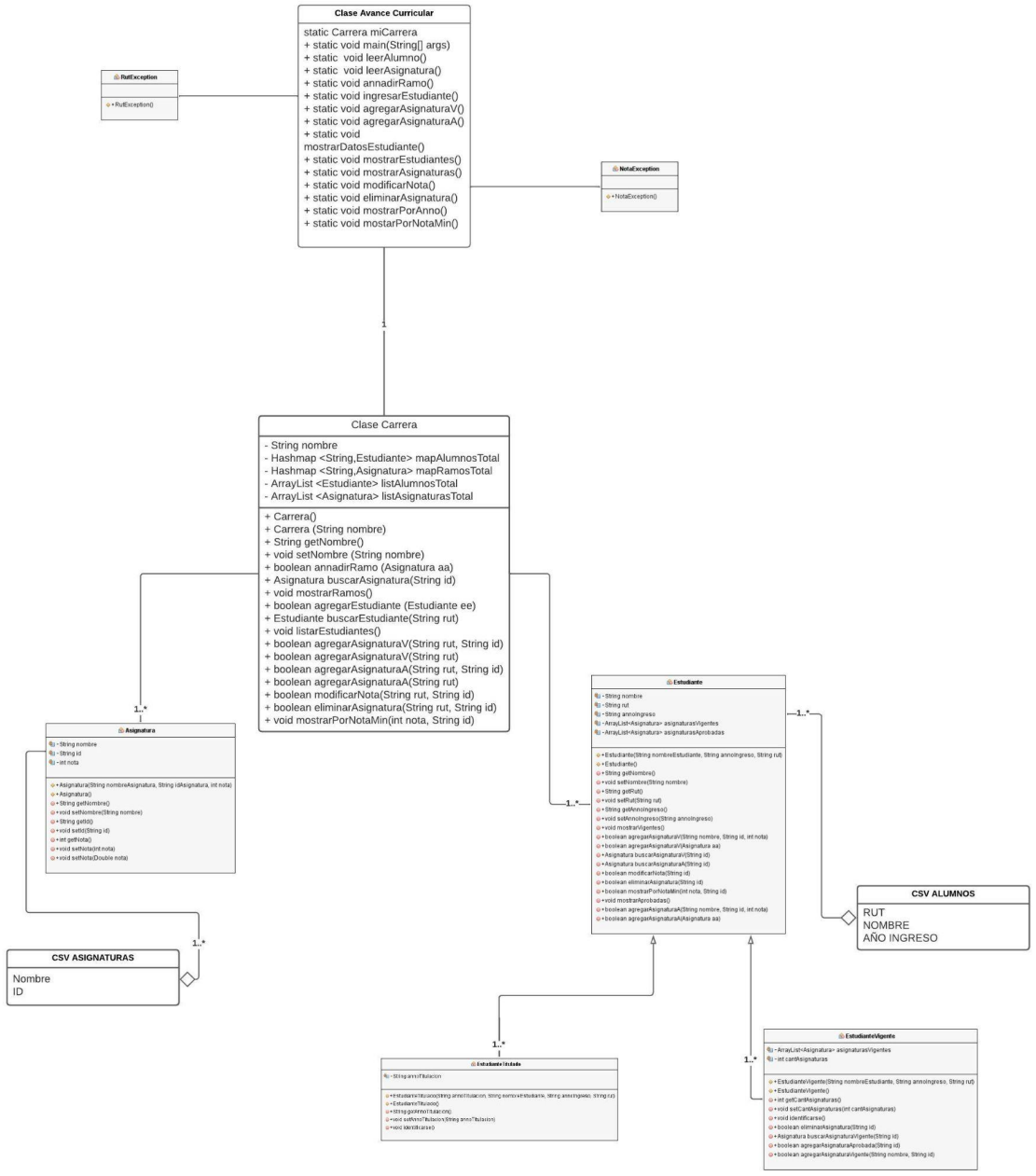
https://github.com/seba-jeria/AvanceCurricular.git

2.1 Diseño de diagrama de clases UML.

Se realiza el diagrama de clases UML respectivo.

Aunque se adjunte la figura siguiente, se recomienda entrar al siguiente link para poder apreciarlo más a detalle.

https://lucid.app/lucidchart/e84e0db1-2841-4785-a05d-31e4a7a896c3/edit?viewport_loc=-153%2C-1055%2C2220%2C1079%2C0_0&invitationId=inv_4b0c178b-d4cf-4c51-9846-12b1cd5d4729



2.2 Persistencia de datos utilizando archivo de texto, CSV, Excel, o conexión con DBMS local (ej. MySQL). Utiliza sistema batch (carga datos al iniciar la aplicación y graba al salir)

En el código se utilizan archivos .CSV para leer y exportar datos, ocupamos un sistema que nos permite al agregar un estudiante o ingresar una asignatura guardarla al instante en el CSV llamado en el programa. Específicamente hacemos esto en los métodos leerEstudiante, leerAsignatura y agregarEstudiante en la clase AvanceCurricular.

A continuación mostramos como quedaría su archivo .CSV.

La codificación de los métodos para leer datos se pueden apreciar en el punto 1.4 anteriormente mencionado. Sin embargo adjuntamos una pequeña parte de la codificación para ingresar un nuevo estudiante al sistema y que quede registrado en el archivo .CSV.

1	Programacion	PROG
2	Bases de datos	BD
3	Redes	RED
4	Sistemas operativos	SO
5	Programacion I	PROG1
6	Programacion II	PROG2
7	Estructura de datos	EDD
8	Base de datos	BD
9	Redes de computadoras	RED
10	Sistemas operativos	SO
11	Ingenieria de software	IS
12	Inteligencia artificial	IA
13	Seguridad informatica	SI
14	Desarrollo web	INF108
15	Fundamentos de programacion	INF101

1	123456789	Juan Perez	2018
2	187654321	Ana Garcia	2019
3	165432108	Pedro Torres	2020
4	198765432	Carla Ruiz	2021
5	234567890	Maria Gomez	2019
6	345678901	Carlos Rojas	2020
7	456789012	Ana Fernandez	2021
8	567890123	Luis Ramirez	2022
9	345678901	Pedro Rodriguez	2020
10	456789012	Sofia Hernandez	2021
11	211750065	Vicente	2031

```java
public static void ingresarEstudiante() throws IOException, RutException{
    BufferedReader hh = new BufferedReader(new InputStreamReader(System.in));
    String ruta = "alumnos.csv";
    try (FileWriter fw = new FileWriter(ruta, true)){
        System.out.println("Ingrese rut: ");
        String rut = hh.readLine();
        System.out.println("Ingrese nombre: ");
        String nombreE = hh.readLine();
        System.out.println("Ingrese año de ingreso: ");
        String año = hh.readLine();
        System.out.println("Ingrese si es estudiante vigente o titulado ");
        System.out.println("0 (vigente) || 1 (titulado)");
        String op = hh.readLine();
        if(op.equals("0")){
            EstudianteVigente estudiante = new EstudianteVigente(nombreE, año, rut);
            if(miCarrera.agregarEstudiante(estudiante)){
                fw.append("\n"); // salto de línea para agregar los datos en una nueva línea
                fw.append(rut).append(",");
                fw.append(nombreE).append(",");
                fw.append(año);

                System.out.println("Estudiante ingresado al sistema :)");
            }
            else{
               System.out.println("Error. No se ha ingresado el alumno al sistema.");
```

2.3 La implementación de todas las interfaces gráficas (ventanas) para interactuar con el usuario, considerando componentes SWING.

Debido a dificultades técnicas para implementar las interfaces gráficas, hemos decidido como grupo no incluirlas a la entrega final del proyecto.

2.4 Se debe hacer un menú para el Sistema donde ofrezca las funcionalidades de: 1) Edición o modificación del elemento y 2) Eliminación del elemento. Esto para la 2ª colección de objetos (colección anidada) del SIA1.5

Eliminar en la clase main CopiaAvanceCurricular:

```java
public static void eliminarAsignatura() throws IOException{
    BufferedReader leer = new BufferedReader(new InputStreamReader( in: System.in));
    System.out.println( x: "Ingrese rut del Estudiante:");
    String rut = leer.readLine();
    System.out.println( x: "Ingrese id de la Asignatura:");
    String id = leer.readLine();
    miCarrera.eliminarAsignatura(rut, id);
}
```

Modificar nota aprobada en la clase main CopiaAvanceCurricular:

```
public static void modificarNota() throws IOException, NotaException{
    BufferedReader leer = new BufferedReader(new InputStreamReader( in: System.in));
    System.out.println( x: "Ingrese rut del Estudiante:");
    String rut = leer.readLine();

    if(miCarrera.modificarNota(rut)){
        System.out.println( x: "Modificada con exito");
    }
}
```

2.5 Se deben incluir al menos 1 funcionalidad propia que sean de utilidad para el negocio (distintas de la inserción, edición, eliminación y reportes).

Hemos incluido 2 funcionalidades propias que son de gran utilidad para nuestro negocio. La primera de ellas es el método mostrarPorNotaMin, el que se basa en filtrar por los estudiantes que tuvieron una mejor nota que la indicada en una asignatura específica, por ejemplo: se podría filtrar por todos los estudiantes que tuvieron una nota mayor o igual que 55 en la asignatura Base de Datos.

La segunda es mostrarPorAnno, la cuál según un año específico, se muestran por pantalla todos los estudiantes que tengan ese año de ingreso.

```
public static void mostarPorNotaMin() throws IOException{
    BufferedReader leer = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Ingrese id de la Asignatura:");
    String id = leer.readLine();
    System.out.println("Ingrese nota minima:");
    int nota = Integer.parseInt(leer.readLine());
    miCarrera.mostarPorNotaMin(nota, id);
}

public static void mostrarPorAnno() throws IOException{
    BufferedReader leer = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Ingrese anno de ingreso: ");
    String an = leer.readLine();
    miCarrera.mostrarPorAnno(an);
}
```

2.6 El código fuente debe estar bien modularizado de acuerdo a lo descrito en el informe además de seguir las buenas prácticas de documentación interna y legibilidad.

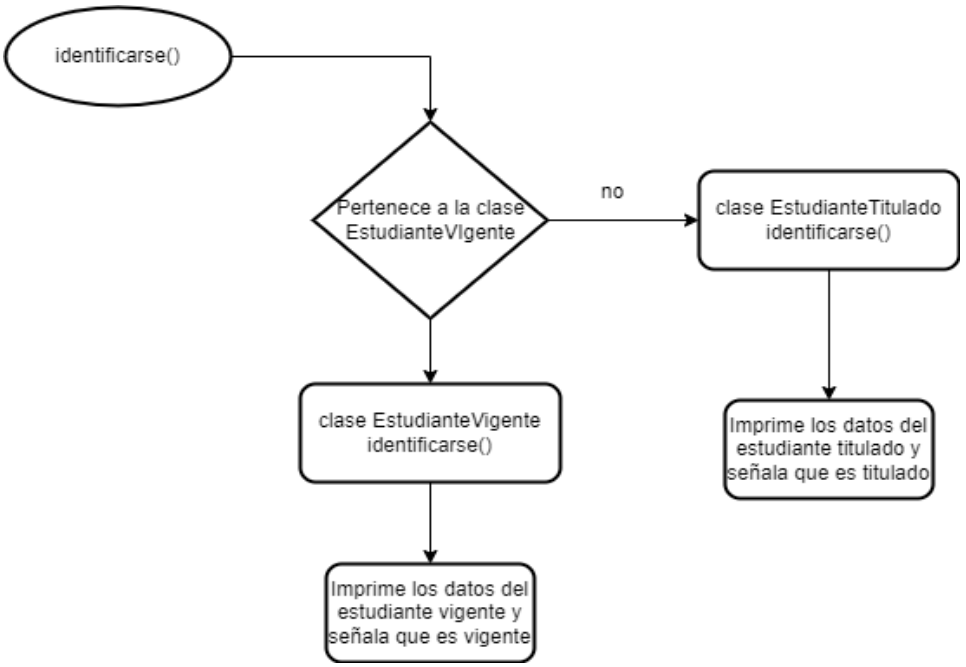
Cada módulo es independiente a otro, los cuales se enfocan en una tarea específica dentro del código, haciendo que el código sea más fácil de entender.

El código cumple con las buenas prácticas de documentación y legibilidad, además de estar bien modularizado.

2.7 Diseño y codificación de 2 (dos) clases que utilicen sobreescritura de métodos.

En el código, la clase estudiante tiene como herencia estudiante titulado y estudiante vigente, de las cuales en todas ellas existe el método identificarse, el que según la clase a la que pertenezcan se van a identificar de manera distinta.

Anteriormente, en el punto 2.1 se puede apreciar más como se extienden desde la clase Estudiante las subclases estudianteTitulado y estudianteVigente. Además se incluye la siguiente figura que puede explicar visualmente la sobreescritura de métodos en estas clases.



2.8 Implementar el manejo de excepciones capturando errores potenciales específicos mediante Try-catch.

En estos métodos hacemos el uso de try catch para asegurarnos de que se ingrese correctamente los datos solicitados para que sean leido desde el csv, si los datos son correctos los agregamos y si no el programa dirá error.

```
public static  void leerAlumno() throws RutException{
    String ruta = "alumnos.csv";
    try (BufferedReader br = new BufferedReader(new FileReader(ruta))) {
        String linea;

        while ((linea = br.readLine()) != null) {
            String[] datos = linea.split(",");
            Estudiante estudiante = new Estudiante();
            int i = 0;

            for (String dato : datos) {
               i++;

               switch(i){
                case 1:{
                    estudiante.setRut(dato);
                }
                case 2:{
                    estudiante.setNombre(dato);
                }
                case 3:{
                    estudiante.setAnnoIngreso(dato);
                }
               }
            }
            //System.out.println(); // salto de línea para la siguiente fila
            miCarrera.agregarEstudiante(estudiante);
        }
    } catch (IOException e) {
        System.out.println("Error");
    }
}
```

2.9 Crear 2 clases que extiendan de una Excepción y que se utilicen en el programa.

Creamos 2 extensiones de excepción una que se basa en que si se ingresa el rut con una letra final distinta de k o K en el dígito verificador el programa se caerá y dirá rut no válido. La otra extensión de excepción que creamos lo que hace es restringir las notas que pueda ingresar el usuario, si la nota ingresa está fuera del rango 1.0 a 7.0 o 10 a 70 se caerá el programa y dirá nota no válida.

La codificación de estas clases se pueden encontrar en el punto 1.2 anteriormente mencionado, por lo que ahora se adjunta cómo se utilizan estas excepciones en el código

```
public boolean agregarEstudiante (Estudiante ee) throws RutException{
    /* retorna true si la asignatura se agregó, y false si no lo hizo*/
    String rut = ee.getRut();
    String lastChar = Character.toString(rut.charAt(rut.length() - 1));
    if ((lastChar.equals("k")) || (lastChar.equals("K"))){
        if(buscarEstudiante(ee.getRut()) == null){
            listAlumnosTotal.add(ee);
            mapAlumnosTotal.put(ee.getRut(), ee);
            return true;
        }
    }
    try {
    int numAux = NumberFormat.getInstance().parse(lastChar).intValue();
        if (numAux >= 0 && numAux <= 9) {
            if (buscarEstudiante(ee.getRut()) == null) {
                listAlumnosTotal.add(ee);
                mapAlumnosTotal.put(ee.getRut(), ee);
                return true;
            }
        }
        else {
            throw new RutException();
        }
    } catch (ParseException e) {
        throw new RutException();
    }
    return false;
```

```
public boolean modificarNota(String id) throws IOException, NotaException{
    BufferedReader leer = new BufferedReader(new InputStreamReader(System.in));
    Asignatura aa = buscarAsignatura(id);
    if (aa != null){
        System.out.println("Ingrese nota:");
        Double nota = Double.valueOf(leer.readLine());

        if((nota>=1.0) && (nota<=7.0)){
            aa.setNota(nota);
             return true;
        }
        else{
           if ((nota>=10) && (nota<=70)){
                aa.setNota(nota);
                return true;
           }
           throw new NotaException();
        }
    }
    return false;
```

2.10 Continuidad en la utilización de GitHub (Realización de al menos 3 Commit adicionales a los ya hechos en el avance)

Se continúa la utilización del GitHub, y se realizan más de 3 Commit. Se adjunta link:

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

3.1 (opcional; Obligatorio en grupos de 4) Implementar las funcionalidades de agregar, eliminar o modificar en elementos de la 1ra colección o nivel.

Se implementa la funcionalidad de agregar en la 1ra colección del proyecto (ingresarEstudiante), la cuál nos permite agregar nuevos estudiantes al sistema de la carrera. Un ejemplo de esto se puede apreciar en el punto 1.2 y el 2.2 (codificación) anteriormente mencionados. Además se pueden ingresar Asignaturas nuevas al sistema en el método annadirRamo.

3.2 (opcional; Obligatorio en grupos de 4) Implementar la funcionalidad de buscar elementos en 1 o más niveles.

Se ofrece la posibilidad de buscar un Estudiante específico para mostrar sus datos en el método mostrarDatosEstudiante, ya que se ingresa el rut del estudiante y se busca en las colecciones de la carrera.

En general se implementa esta funcionalidad en los métodos de agregar, modificar u otros, ya que el programa necesita buscar entre las colecciones si el elemento ya existía o no.

Se puede apreciar las funcionalidades en el punto 1.2.

```
//Muestra los datos del estudiante y las asignaturas aprobadas
public static void mostrarDatosEstudiante()throws IOException{
    BufferedReader leer = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Funcion mostrar info del estudiante\n");
    System.out.print("Ingrese el rut (x para salir): ");
    String rut = leer.readLine();
    System.out.println("");
    if(rut.equals("x")){
        System.out.println("Ha salido de la funcion\n");
        return;
    }
    Estudiante ee = (Estudiante)miCarrera.buscarEstudiante(rut);

    if(ee == null){
        System.out.println("Alumno no encontrado");
        return;
    }

    System.out.println("Nombre: "+ee.getNombre());
    System.out.println("Año ingreso: "+ee.getAnnoIngreso());
    System.out.println("Rut: "+ee.getRut()+"\n");
    ee.mostrarAprobadas();


    System.out.println("");
}
```