1. Encapsulamiento

En la solución se aplicó el principio de ocultar los datos internos de cada clase declarando sus atributos como private. De esta forma, nadie puede acceder ni modificar directamente esos valores desde fuera de la clase. El único modo de hacerlo es a través de métodos públicos (getters y setters), lo que permite tener control total sobre cómo se leen o modifican los datos. Esto evita cambios indebidos y asegura que siempre pasen por un filtro de reglas previamente definidas.

2. Relaciones entre clases

Se representaron las conexiones naturales entre los objetos del dominio:

Estudiante – Materia: un estudiante puede estar inscripto en varias materias, por lo que se usó un ArrayList. Esto refleja una relación de uno a muchos.

Estudiante – Carrera: cada estudiante pertenece a una sola carrera, representada con un atributo de tipo Carrera. Aquí hablamos de una relación de uno a uno.

Además, la clase App se utilizó como programa principal, encargada de crear objetos y mostrar cómo interactúan entre sí estudiantes, materias y carreras.

3. Validaciones en los setters

Para mantener la coherencia de los datos, los métodos set incluyen comprobaciones:

En Estudiante, la edad debe ser mayor a 16, y tanto nombre como apellido no pueden ser nulos ni estar vacíos.

En Materia, se valida que el nombre no esté vacío y que los créditos sean mayores a cero.

En Carrera, se comprueba que el nombre no sea nulo ni vacío.

De esta manera se evita crear objetos con información incorrecta o incompleta.