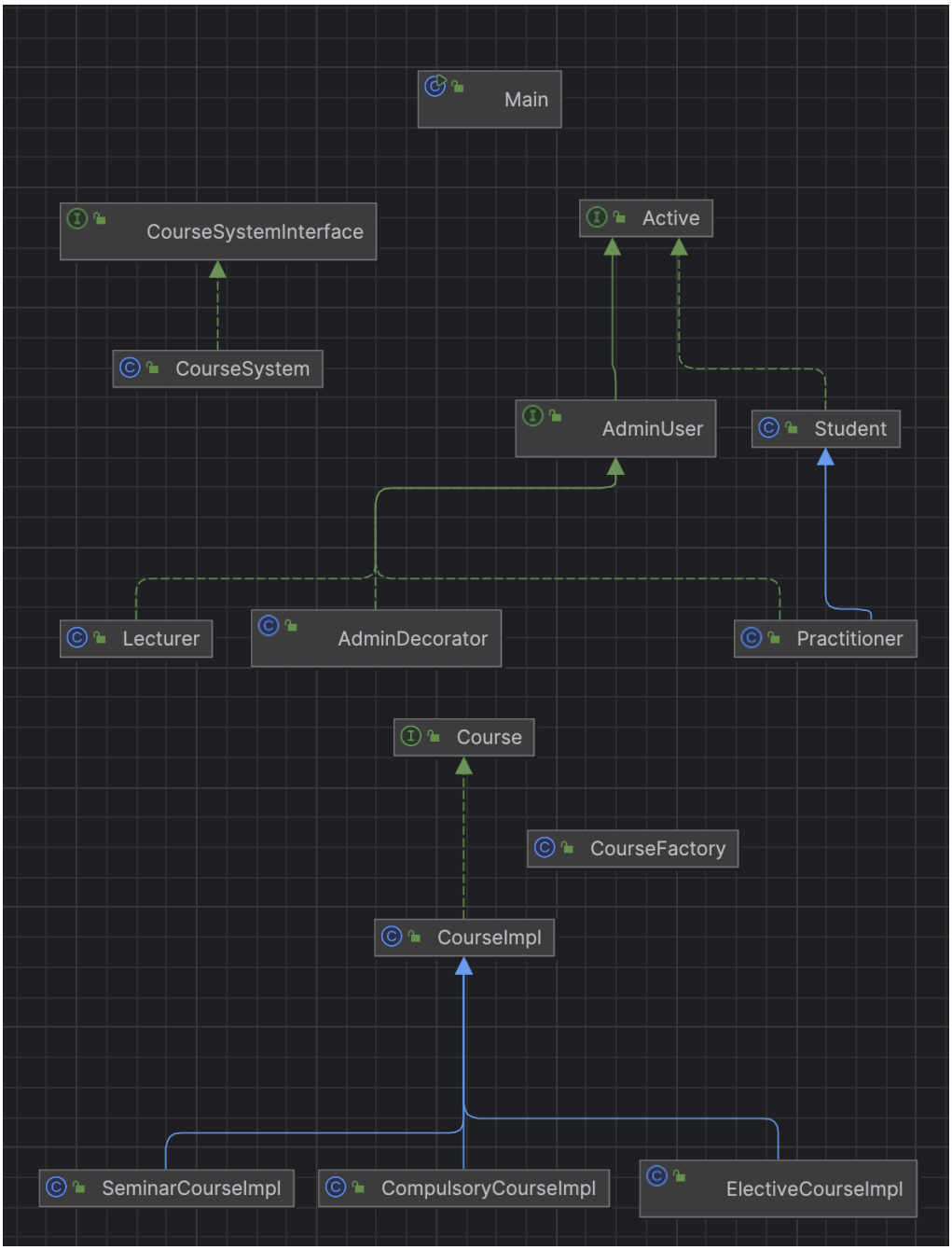
שאלה 2 – עיצוב חופשי

בשאלה זו התבקשנו לתכנן מערכת הרשמה לקורסים.
ניתן להריץ דוגמה למערכת כזאת על ידי הקובץ main.java המצורף במטלה.

מבנה המערכת:



המערכת מאותחלת על ידי האובייקט CurseSystem שהוא מהווה אימפלמנטציה של הממשק CourseSystemInterface – ממשק פשוט ונוח המאפשר יצירת מערכת, הוספת פעילים למערכת, קורסים במערכת, הסרת סטודנט מקורס מסוים, הרשמת סטודנט לקורס מסוים וכו..
מערכת זאת לא מאפשרת הרשמה של יותר מ100 פעילים, כאשר לכל פעיל יש מספר ID ייחודי, אם תנסו להכניס למערכת 2 פעילים עם id זהה היא לא תאפשר זאת (ניתן לראות זאת בmain באמצעות student4) .

כאן אנחנו יכולים לראות את הdesign pattern הראשון שלנו – **Façade**
מאחר ואנחנו רואים ממשק פשוט למערכת מורכבת המקלה על לקוחותיה ומאפשרת אינטראקציה עם המערכת בלי צורך להבין את המורכבות הפנימית שלה ובעצם מסתירה את פרטי היישום של המערכת מאחורי ממשק אחד מאוחד.

ממשקים נוספים מאחורי המערכת:

- Active
  יש לנו 3 סוגי פעילים אשר מיישמים את הממשק הזה: סטודנט, מרצה ומתרגל כאשר מתרגל מהווה extantion של סטודנט (מאחר ומתרגל הוא בעצם סטודנט בעצמו).
  רק למרצה או מתרגל יש אופציה להוסיף קורס באמצעות יצירת אוביקט של AdminDecorator (נסביר בהמשך) ומאחורי הקלעים המערכת תדאג ליצירה של הקורס כמובן שעליו לציין איזה סוג קורס זה: בחירה/ סמינר/ חובה.
  במימוש זה אנחנו נראה DP נוסף והוא – **Factory** המאפשר יצירה של סוגים שונים של קורסים מבלי לחשוף את הגיון היצירה, אנחנו נשלח באמצעות הadmin לבנאי רק את המילה המתאימה (בחירה/ סמינר/ חובה) והמימוש יעשה מאחורי הקלעים.

- AsminUser
  ניתן לראות כי ממשק זה ממומש עי הPractitioner והLecturer מאחר ורק להם יש הרשאות מנהל.
  ממשק זה מהווה DP נוסף והוא **Decorator** , המגדיר התנהגות מסוימת של הפעילים תוך השמת פונקציונליות נוספת לפעיל המוגדר כ'admin'. המחלקה AdminDecorator מממשת גם היא את ה AdminUser ובעצם מאפשרת מעטפת של הרשאות מנהל ליצירת קורסים. במימוש המחלקה ישנה בדיקה האם הuser שנרשם כadmin הוא אכן מסוג מרצה או מתרגל.
  נוכל לראות דוגמה של יצירת קורס עי מנהל בmain:

```
Practitioner practitioner = new Practitioner(8, "Alice Wonderland");
AdminUser adminPractitioner = new AdminDecorator(practitioner);
Course course1 = adminPractitioner.addCourse("Introduction to Java
Programming", 2, "Seminar");
```

סטודנט יכול להירשם לקורס וגם להסיר הרשמה במידת הצורך, גם פה מסתתר DP נוסף **Observer** – לכל סטודנט יש רשימת המתנה של קורסים אליהם ניסה להירשם אך לא היה מקום בקורס, וכך בעצם הסטודנטים רשומים כ"צופים" לקורסים אשר נמצאים ברשימת ההמתנה שלהם, ואילו הקורס אליו ממתנים מבצע notify לאותם הסטודנטים ברגע שסטודנט אחר מסיר את הרשמתו מהקורס הנ"ל. כך בעצם המחלקה CourseIml מרחיבה את האובייקט Observable ובהתאם המחלקה Student מממשת את הממשק Observer .

- Course
ממשק נוסף אשר ממומש עי המחלקה CourseImI, שאותו מרחיבים שלושה סוגי קורסים שונים: SeminarCourseImpl, ElectiveCourseImpl, and CompulsoryCourseImpl
גם על ידי מימוש זה אנחנו יכולים לראות DP נוסף והוא **Singleton**
עיצוב המבטיח מופע יחיד של קורס מאותו הסוג, עקביות ומונע כפילות של אובייקט מסוג ׳קורס׳. אנחנו יכולים לראות את זה עי המחלקה CourseFactory אפשר מסתירה את הבנאי שלה בprivate ומספקת בנאי חלופי בשם GetInstance שרק הוא אחראי ליצירת instance של האובייקט קורס ובמימושו גם מונפק לכל קורס מספר סידורי ייחודי. כמו כן גם הבנאי של CourseImI מוסתר עי private מה שמבטיח שיצירת האובייקט ׳קורס׳ תתקיים רק באמצעות ה GetInstance .

עד כה ספרנו 5 תבניות עיצוב שונות:
Façade, Factory , Observer, Decorator, Singleton .