**הסברים ליחידת UiLibrary :**

**תבנית עיצוב אשר מומשה: State**

**Context** - אצלנו המחלקה היא UiLibrayContext , מחזיקה פוינטר למשתנה מסוג UiLibraryState .

קיים רק context אחד בכל התוכנית שאותו מעבירים בין הסטייט השונים, נמצאת בה הפונקציה run שרצה בלופ אינסופי כדי להשתמש ב-handleInput ו-display .

**State** - אצלנו המחלקה היא UiLibraryState , מחלקה אבסטרקטית שכל הסטייטים יורשים ממנה.

**concreteStates** - אצלנו המחלקות הם :

StudentRegisterState, EntryDisplayState, StudentUnregisteredCourseListState,ListDisplayState,LoginState,StudentListPerCourseState,SecreteryRegisteredListState,CourseListPerStudentState,SecreteryHomeScreenState,

TerminalTestingState,TechnicianHomeScreenState,StudentRegisteredCourseListState,

StudentListHomeScreenState,TerminalListUnlockState

כל המחלקות האלו יורשות מUiLibraryState וממשות את הפונקציות האבסטרקיות שלה.

ישנם גם concreteStates שיורשים מ ListDisplayState שבה יש מימוש רק של handleInput וdisplay ושאר הפונקציות אבסטרקטיות והמחלקות שיורשות מהם צריכות לממש אותם.

בנוסף יש גם את EntryDisplayState שגם ממומש שם display ו handleInput ושאר המחלקות שיורשות ממנה ממשות את הפונקציות האבסטרקיות שיש בה.

הסברים ליחידת **DbProject**:

תבנית עיצוב אשר מומשה: **Abstuct Factory**

* - Abstruct Factory אצלנו המחלקה היא IStorage משמשת כמחלקה אבסטרקטית עם שתי פונקציות אבסטרקטיות get ו set.
* ConcreteFactory - אצלנו המחלקה היא FileStorage מומשות שמה שתי הפונקציות get ו-set , ופונקציה נוספת:

1. generateUniqeId - מכין id יחודי.

* AbstructProduct - אצלנו המחלקה היא IdObj ממנה יורשים ה-products האחרים.
* products - אצלנו המחלקות הם : Student,Course,Terminal,SeasonConfig,SystemState,Bidding

\* IdObjFactory - מחזיק רפרנס למשתנה מסוג IStorage וניתן להעביר אליו אובייקטים שיורשים מIStorage , לכן דרכו ניתן לפנות לFileStorage ולבצע מה שצריך.

\*NonRefField- מחלקה שמחזיקה רפרנס לאובייקט מסוג IdObj, קיימת בשביל שדות של טיפוסים פרימיטיבים.

\*RefListField - מחלקה שמחזיקה רפרנס לאובייקט מסוג idObj, קיימת בשביל רשימה של טיפוסים לא פרימייטיבים.

המחלקות של ה-products בעצם מכילים משתנים מסוג RefListField ו NonRefField בהתאם.

הסברים ליחידת **Lexer-Parser** :

**תבניות עיצוב אשר מומשו:**

**1. Builder**

**2. Composite**

מחלקות אשר מיצגות את התבניות :

**1. Expression - Composite**

---------------------------------

בתבנית עיצוב זו, מימשנו ביטוי אריתמטי בצורת עץ, אנו מאפשרים התייחסות לאובייקטים כאובייקטים עצמאיים או באופן אחיד לצרוף של אובייקטים.

חלוקת התפקידים :

----------------------

\* Component - מחלקת Expression משמשת אותנו כמחלקה האבסטרקטית.

\* Composite - מחלקת BinaryOperator משמשת אותנו בתור האובייקט המרוכב,

מצד אחד - אנו יכולים לבנות בעזרתה עוד רכיבי Expression נוספים, שיכולים להיות עוד רכיבי BinaryOperator כמו Mult, Minus, Plus, Divison.

מצד שני - אנו יכולים להחזיק באמצעותה עלים Leafs, בתחתית העץ.

\* Leaf - מחלקת Number משמשת אותנו במחלקת העלה, היא מחזיקה מספר בודד.

בצורה זו, אנו יכולים ליצור ביטוי רקורסיבי אשר מסוגל לחשב את עצמו .

**2. Lexer-Parser - Builder**

---------------------------------

בתבנית עיצוב זו, מימשנו יצירה של אובייקטים (Expressions) באמצעות מידע שטחי, קלט פשוט לצורך העניין.

חלוקת תפקידים:

-------------------

\* Director - מחלקת נועדה לקחת data בתור מידע שטחי, ולהעבירו אל מחלקת ה - Parser לצורך עיבוד ובניית האובייקטים - Expressions .

\* Builder - מחלקת Parser מקבלת vector של אסימונים ממחלקת ה- Lexer,

ומשם היא ממשיכה לעיבוד הנתונים בשביל יצירת ביטוי רקורסיבי (Composite) אשר בסופו של דבר, ייצור Expression אחד המסוגל לחשב את עצמו.

\* Product - כאשר ה- Parser רוצה להחזיר ביטוי קונקרטי, הוא משתמש בפונקציה : GetExpression,

ומחזיר ביטוי מסוג Expression - אשר משחק בתפקיד Product.