קבוצה: G1

מגישים: דוא"ל:

יאנה אוסטרובסקי [yanasn12@gmail.com](mailto:yanasn12@gmail.com)

ליבנת מנשה [livnatmn@gmail.com](mailto:livnatmn@gmail.com)

עדי אליהו [adidushet@gmail.com](mailto:adidushet@gmail.com)

דורון וייצמן [doronw91@gmail.com](mailto:doronw91@gmail.com)

משה סבן [moshir321@gmail.com](mailto:moshir321@gmail.com)

1. כאשר ניגשנו לתהליך הוספת נושא של ספר למאגר הנושאים במערכת עמדה בפנינו דילמה כיצד לממש את הדבר, בהתאם לדרישות כל נושא בהכרח חייב להשתייך לתחום מסוים.

הדילמה שעמדה בפנינו היא האם לאפשר הוספת נושא חדש למאגר כאובייקט בפני עצמו ולא כתכונה שמוגדרת בעת הוספה של תחום חדש, ראשית חשבנו להוסיף את הנושא החדש כאובייקט בפני עצמו ולא כתכונה שמוגדרת בעת הוספה של תחום חדש, רצינו לממש זאת באמצעות הכלה חלשה (Aggregation) בה קיים מערך נושאים ובעת הוספת נושא חדש מתווסף מצביע לתחום המתאים.

הגענו להבנה שהנושא לא יכול לעמוד בפני עצמו (חייב להיות משויך לתחום מסוים) רצינו לממש זאת בעזרת הכלה חזקה (Composition) בה נפעיל בתוך הConstructor- בנייה של אובייקט מסוג נושא חדש.

מימוש זה עדיף מכיוון שהוא נכון יותר מבחינה הגיונית, מקל על חיפוש ספר באמצעות נושא או תחום,

כמו כן שיטה זו מקצרת זמן ריצה ומקלה על הוספת/הסרת נושאים וספרים.

1. א.

אחד העקרונות של Object Oriented Analysis & Design שבו השתמשנו הוא

ירושה (Generalization).

עיקרון זה מתיר להשתמש בתכונות של מחלקה מסוימת במחלקה אחרת

ולהוסיף לה מתודות/תכונות שיהיו ייחודיות רק לה (למחלקה היורשת).

עיקרון זה בא לידי ביטוי בכמה מקומות ב-Class Diagram:

* המחלקה **Person** שכל משתמש במערכת יורש ממנה. מחלקה זו מייצגת את התכונות שיש לשמור לכל אדם במערכת.
* המחלקות **Worker** ו- **PayingUser** יורשות מ- **User** כיוון שתכונותיו (userName, password,..) שייכות גם הן למחלקות.
* המחלקות **Librarian** ו- **LibraryManager** יורשות מ- **Worker** כיוון שלשניהם קיימים פעולות משותפות, על מנת למנוע כפילויות בחרנו ליצור ישות נוספת אבסטרקטית שתבצע את הפעולות המשותפות של הספרן והמנהל.
* המחלקות **LibrarymanagerUI** ו-**LibrarianGeneralUI** יורשות **מWorkerGeneralUI** מאותה הסיבה הנ''ל, ה-GUI יהיה זהה עם מספר תוספות שונות לספרן ולמנהל בהתאמה.
* המחלקה **PayingUserGeneralUI** יורשת מ **UserGeneralUI**, **payingUser** הוא משתמש מסוג user אך יש לו פעולות נוספת שהוא יכול לבצע.

קיימת בנוסף הורשה מ- AbstractClient ו- AbstractServer למחלקות ClientController, ServerController בהתאמה.

כידוע יש ליצור מופע של המחלקה האבסטרקטית רק לאחר מימוש המחלקה.

לכן עלינו לבצע את ההורשה ולהשתמש בתכונות של המחלקות ולהגדיר את הנתונים הייחודים עבור התקשורת בפרויקט שלנו.

ב.

פולימורפיזם זו תכונה של [שפות תכנות](http://he.wikipedia.org/wiki/%D7%A9%D7%A4%D7%AA_%D7%AA%D7%9B%D7%A0%D7%95%D7%AA) מונחות עצמים המאפשרת לטפל ב[ערכים](http://he.wikipedia.org/wiki/%D7%A2%D7%A8%D7%9A_(%D7%9E%D7%93%D7%A2%D7%99_%D7%94%D7%9E%D7%97%D7%A9%D7%91)) מ[טיפוסים](http://he.wikipedia.org/wiki/%D7%98%D7%99%D7%A4%D7%95%D7%A1_%D7%A0%D7%AA%D7%95%D7%A0%D7%99%D7%9D) שונים בעזרת [ממשק](http://he.wikipedia.org/w/index.php?title=%D7%9E%D7%9E%D7%A9%D7%A7_%D7%AA%D7%95%D7%9B%D7%A0%D7%94&action=edit&redlink=1) אחיד.

במערכת שאנו ממשים לא ראינו לנכון צורך להשתמש בעיקרון זה, לא קיימים אובייקטים מטיפוסים שונים שצריך לטפל בהם באמצעות ממשק אחד.

1. א.

1.

היתרונות של מודל ה-UML הם:

* מאפשר למדל את המערכת בצורה נוחה וידידותית למשתמש.
* הuse case מציג את הפעולות האפשריות של מתפעלי המערכת.
* הclass diagram מאפשר למתכנת לראות את הקשרים בין המחלקות וכך לחלק אותן

וליצור קשרים ביניהן באופן נכון.

* הsequence מתאר באופן מפורט תהליכים במערכת וכיצד הם מתבצעים (אילו מתודות וכו').
* הactivity מציג בצורה ברורה את הלוגיקה מאחורי התהליכים במערכת.
* משתמש בשפה אוניברסלית- מתכנת בכל שפה מונחית עצמים יכול להבין ולמדל בעזרתו.
* ניתן לזהות בעזרתו קשיים שיכולים להיות בשלב יותר מאוחר וכך אולי למנוע אותם. דבר זה כמובן יחסוך לנו גם זמן ועלויות.

כאשר התחלנו לממש את מודל ה- UML הפרדנו בין כל ישות לפעולות שהיא יכולה לבצע, בשלב מסוים ראינו כי יש לנו כפילות במספר פעולות, זאת אומרת- כמה ישויות שמבצעות את אותה הפעולה, לכן החלטנו להוסיף ישות אבסטרקטית – Worker שתבצע את הפעולות המשותפות להם, וכאן החלטנו להשתמש בעקרון ההורשה של תכנות מונחה עצמים וליצור ישות נוספת, בכדי להפחית את רמת הסיבוכיות ולמנוע יתירות ושיכפול של קטעי קוד.

2.

הקשיים בהם נתקלנו בעת יצירת דיאגרמות הUML היו למשל:

* יצירת הדיאגרמות השונות לקחו זמן נכבד, דבר שאילץ אותנו לחלק את העבודה

בין חברי הצוות.

לדוגמה בעת יצירת הsequence diagram הבנו שבכדי להספיק אותם

ולעבור עליהם באופן מעמיק, נצטרך לחלק את העבודה בין כולנו.

כל חבר צוות היה אחראי על sequence של actor מסוים (user, librarian…)

כך כל sequence קיבל יותר תשומת לב.

בזכות החלוקה נחסך לנו הרבה זמן, שהותיר לנו זמן להיפגש כולנו ולעבור ביחד על הדיאגרמות לאחר מכן.

* כל סוג דיאגרמה בUML מציג היבט אחר של המערכת.

דבר זה גרם לחוסר עקביות חלקי באופן בו מידלנו תהליכים.

למשל בעת יצירת הsequence diagram הבנו שהתכנון בflow לא היה נכון ונאלצנו לתקן.

הבנו שע"מ לפתור בעיה זו היה עלינו למדל דיאגרמות שונות של אותו תהליך במקביל.

ב. לא, שימוש במתודולוגיית UML אינה נותנת מענה מלא לצרכי תהליך ה-Design שביצענו בפרויקט.

* תרשימי ה-UML לא נותנים מענה לממשק הגרפי. לא ניתן לראות את מבנה החלון ואת תכולתו ועיצובו הפנימי כפי שיוצג למשתמש.
* אין דיאגרמה בה ניתן לתאר את מבנה מסד הנתונים במערכת, דבר שיכל להקל ביצירת המחלקות הדרושות לפעולת המערכת.

לדוגמא בפרויקט שלנו - כאשר רצינו להציג דו"ח ואת הנתונים שמוצגים בו, לא ידענו כיצד הדו"ח יהיה בנוי, מאיזה טבלאות עלינו להביא את המידע ואיך המידע יוצג בתוכו.