**Design Patterns – מבחן סיכום**

רועי היקר,

כיוון שאינני עוסקת בתכנות במסגרת תפקידי,

באישורו של איתי מרצה הקורס נדרשתי לזהות את ה pattern בו נשתמש למימוש ולא לבצע את המימוש עצמו.

**חלק א'**

1. למימוש משחק החלליות נשתמש ב **State Pattern** , לחללית יש מספר מצבים במהלך המשחק

כל שלב יכיר את ה state האחרים ויבצע את הפעולות הנדרשות בהתאם למצב הנוכחי ומה שנדרש כדי לעבור למצב הבא ( שינוי counterים של נקודות/ פגיעות) ויעביר ל state הבא עד לסיום המשחק.

1. למימוש עץ המספרים ופעולות הסכום ובדיקה האם העץ מכיל מספרים זוגיים בלבד נשתמש ב **Composite Pattern** אשר מאפשר יחס של הכלה, כל קומפוננטה (אבא) בעץ תוכל להכיל בנים מאותו הסוג. בנוסף ע"י שימוש ב pattern זה נוכל לממש פעולות על כל הילדים של אותו האב

(קומפוננטה), הפעולות הנדרשות הן סכימה של כל הבנים כולל המספר של האבא, ופונקציה נוספת אשר תחזיק משתנה בוליאני ותבצע בדיקה האם כל הרכיבים בעץ מחזיקים מספרים זוגיים ע"י רקורסיה.

1. למימוש שליטה/ בקרה במכונית נשתמש בשני patterns : **Proxy** ו- **Facade**.

נרצה לרכז את כל הפונקציונליות במקום אחד, ולחשוף/ לתת גישה רק לחלק מהפונקציות.

כמו בדוגמה שעשינו בכיתה עם המערכת לניהול טיסות.

ה Facade יחזיק אובייקט DAO וכך יאגד את כל הפונקציונליות שלו (התחלת נסיעה, סיום נסיעה, הגברת מהירות, האטה, פניה ימינה, פניה שמאלה, מצב דלק, הצגת מיקום) הFacade יבצע את הקריאה לפונקציות מה DAO,

ה Proxy יחזיק מופע של facade וכל proxy יקרא לפונקציות הרלוונטיות מתוכו, בתרגיל הנתון יקרא ל הצגת מצב דלק והצגת מיקום.

1. בפתרון תרגיל זה נשתמש במספר patterns :

**Visitor** – מאפשר לבצע את הפעולות על המערך

נבנה מבנה נתונים המחזיק מערך של מספרים שלמים, ונתונים על המערך כמפורט בתרגיל.

מבנה הנתונים יהיה visitable כלומר נוכל "לבקר" בו (לבצע עליו שינויים), בהתאם לפונקציונליות שתידרש.

ניצור 2 מחלקות visitor שכל אחת מהן תקבל את המערך ותבצע את הפונקציונליות הנדרשת,

מיון המערך או מכפילה כל אחד מאיברי המערך ב2.

**Strategy** -

כדי לבצע את הפעולות על המערך בהתאם לקלט מהמשתמש נשתמש ב Strategy Pattern

המאפשר לנו לשנות את הפונקציונליות של האובייקט(מבנה הנתונים) בכל פעם נבדוק את הקלט ונשנה את הפונקציונליות למיון/הכפלה בהתאם.

**Prototype**

כדי להדפיס את איברי המערך ולקבל אותם אחד אחד מבלי האפשרות לשנות אותם נרצה לשלוח שכפול של האובייקט ולא את האובייקט עצמו שניתן לשינוי. נעשה זאת ע"י מימוש של prototype ונשלח clone של מבנה הנתונים, ונעבור על איברי המערך אחד אחד ונדפיס.

1. ניהול לוגים במערכת נממש באמצעות **Chain Of Responsibility**, כל Log Type יהיה חולייה ב "שרשרת" וכל חולייה תחזיק משתנה NEXT שהוא המצביע לחולייה הבאה בשרשרת, כל אחת מהחוליות לא תדע אם היא האחרונה, כל חוליה/ לוג יבצע את הפעולה הנדרשת ממנו בהתאם למה שנדרש(info ידפיס הודעה למסך, eroor ידפיס למסך ויכתוב לקובץ וכו'), במידה וסיים יצא, במידה ולא יעביר את הטיפול לחולייה הבאה בשרשרת (Next) אם היא לא null אם כן יבצע יציאה.

בmain נבצע את הקישור בין החוליות בשרשרת ונאתחל את משתנה ה Next בהצבעה למחלקות הלוגים.

1. לאחסון של רשימה ארוכה של מספרים הניתנים ע"י בסיס החזקה והמונה נממש באמצעות **Flyweight Pattern** אשר מאפשר לחסוך מקום בזיכרון.
2. **Mediator – לא למדנו**
3. למימוש אימון בחדר כושר המורכב מסדר פעולות קבוע נשתמש ב **Template Method**

המימוש עבור כל אחת מהפעולות : החלפת בגדי אימון, שתית מים, ביצוע תרגיל כושר ומקלחת יכול להתבצע באופן שונה אך סדר הפעולות הנ"ל יישאר קבוע.

ניצור מחלקה אשר תחזיק פונקציית run שתבצע הרצה של אימון בחדר כושר לפי סדר הפעולות הקבוע. ובנוסף תחזיק הצהרה אבסטרקטית לכל אחת מפעולות אלו.

נממש מחלקות בסיס עבור כל סוג אימון שנרצה לבצע המבצעות override לפעולות האימון לפי אותו הסוג לדוגמא עבור אימון TRX ואימון קיקבוקס נממש שתי מחלקות בסיס שיכילו את הפעולות החלפת בגדי אימון, שתית מים ומקלחת, ובנוסף ניצור מחלקות שונות עבור תרגילי האימונים אשר יממשו בכל פעם סוג תרגיל אחר בכל אחד מסוגי האימונים.

כמו בדוגמה שעשינו בכיתה עם פעולות ההתחברות ל DB, מימשנו שתי מחלקות בסיס להתחברות לשני סוגי DB שונים אורקל ומונגו המממשות את יצירת החיבות, פתיחת החיבור וסגירתו ומימשנו מחלקות שונות עבור פקודות שרצינו להריץ.

1. **Abs Factory – לא למדנו**

1. למימוש בניית החלון באופן גמיש ודינאמי נשתמש ב **Decorator Pattern**,

תבנית ה decorator תאפשר לנו להוסיף לחלון הבסיסי את מרכיבי החלון בצורה דינאמית

ובקומבינציות שונות ללא צורך ביצירה של מופע חדש של החלון.

עבור כל רכיב שנרצה להוסיף נממש מחלקה אשר מוסיפה את הרכיב החדש לחלון שרוצים.

כמו בדוגמה של התרגיל עם הפיצה שעשינו בכיתה ובכל פעם הוספנו תוספות, מה main

נשלח את החלון שלנו לפונקציה אשר תוסיף רכיב ואם נרצה להוסיף עוד רכיב אז נשלח את

החלון המשודרג ונוסיף לו עוד רכיב מבלי ליצור מופע חדש של חלון.

1. לבניית סימולציה של תנועה של אובייקט במרחב נשתמש ב**Pattern Memento**

נממש את הפונקציונליות הנדרשת עבור הזזת האובייקט המכיל מספר בהתאם לקלט המתאים

הזזה למעלה, מטה, ימינה, שמאלה, הגדלה של המספר והקטנתו ויציאה מהסימולציה.

על מנת לממש את הפונקציונליות של שמירת המצב הקיים והאפשרות לחזור חזרה למצב האחרון שנשמר נממש באמצעות Memento אשר ישמור רשימה של כל המצבים בהם בוצע save וידע לבצע את התזוזה בין המצבים שנשמרו,

בכל פעם שיתבצע save (S ) נבדוק אם המצביע שלנו הוא בסוף הרשימה אם כן, נבצע שמירה של המצב הקיים ע"י הוספה לרשימה, אם המצביע לא בסוף הרשימה ( כלומר ביצענו שחזור של המצבים ע"י L ) אז נמחק את כל המצבים שנשמרו מהמקום הבא ברשימה ועד סופה, ובסוף נבצע הוספה של המצב הנוכחי לרשימה לאחר המצב בו המצביע שלנו נמצא.

מימוש החזרה לאחור יבוצע ע"י בדיקה האם קיים יותר מאיבר אחד ברשימה והאיבר הקודם למצביע הוא לא ריק, אם כן המצביע יזוז איבר אחד לאחור.

בנוסף, לאחר כל אחת מהפעולות שמימשנו אם האובייקט יצא מגבולות המטריצה, נחזור למצב האחרון שנשמר אם יש כזה, אחרת נצא מהסימולציה.

1. למימוש המעבדה לבניית מחשבים בסדר פעולות קבוע ( הוספת מרכיבים ) נשתמש ב

**Builder Pattern**, ניצור מחלקה של בניית מחשב שתכיל פונקציות set לפי שדות אובייקט המחשב, ופונקציית build שתיצור מחשב חדש על פי סדר הפעולות (הערכים שביצענו להם set)

הבא : בניית מארז, הוספת לוח אם, הוספת מעבד, הוספת כרטיס גרפי, הוספת זכרון והרצת בדיקה.

בנוסף נבנה מחלקות הבונות מחשב לפי "מתכון" עבור סוג המחשב : מחשב משחקים, מחשב משרדי ומחשב סלון.

נשתמש גם ב **Factory pattern** על מנת ליצור את סוג המחשב בהתאם לקלט שנקבל, ניצור מחלקת ComputerFactory שתכיל בתוכה פונקציה ליצירת מחשב ותחזיר אובייקט מחשב מהסוג הרצוי.

בתוך הפונקציה נממש בדיקה לקלט שהקבל וע"י switch case נבצע יצירה של המחשב הרצוי ע"י יצירת אובייקט לפי מחלקת המחשב המתאימה לפי הסוג וקריאה לפונקציה הbuild. ובסוף נחזיר את המחשב שיצרנו.

**חלק ב' – שאלות מחשבה**

1. נחזיק שני מצביעים/ אינדקסים למערך, הראשון p1 יתחיל מתא במיקום 0 והשני p2 מהתא במיקום 1,

נבצע ריצה עד שהערכים בתא במיקום של המצביעים יהיו שונים ואז בהכרח נמצא את המספר שאינו מופיע פעמיים.

בכל פעם נבצע בדיקה אם הערכים בתאים שווים אם כן נגדיל ( נקדם )כל אחד מהמצביעים ב2.

במידה ולא ( כלומר הערכים בתאים שונים ) אז המספר שאינו מופיע פעמיים הוא הערך במיקום עליו מצביע p1.

בהנחה ש n גודל המערך נבצע לכל היותר n/2 פעולות השוואה.

ניצור מבנה מסוג hashmap בו המפתח יהיה המילה הממוינת וערך count המחזיק כמה פעמים המילה מופיעה.

נעבור על הרשימה ועבור כל מילה ברשימה נבצע מיון של האותיות לפי מיון כלשהו שנקבע מראש

נבדוק האם התוצאה קיימת ב hashmap.

אם כן נעדכן את ערך ה count ונוסיף לו 1.

אם לא, נוסיף את המילה לאחר מיון ל hashmap ונשים ערך 1 ב count.

נדפיס את כל ה [k,v] של ה hashmap .