**תרגול 4 OOP – 1.11**

תבנית עיצוב Decorator (מקשט):

* הגדרה: תבנית עיצוב של מקשט מוסיפה פונקציונליות נוספת לאובייקט
* מבנה:
  + בבסיס יש לנו מחלקה אבסטרקטית או ממשק שמגדיר באילו פעולות אנו רוצים לתמוך (API)
    - כלומר ראשית נכריז על הבסיס
  + יש לנו ConcreteComponent שיורש ממנו וממש את הפעולות הנדרשות
  + דקורטור מחזיק את המופע של המרכיב הקונקרטי ומממש את הפעולה הנדרשת – אופציונלי (אם אין הרבה לא צריך את זה ומספיק רק את הדקורטור הקונקרטי)
  + ConcreteDecorator – הפעולה הנדרשת עם הפונקציונליות הנוספת שנרצה
* הערות:
  + הרכיב – component – בדר"כ הוא ממשק (או מחלקה אבסטרקטית)
  + גם הדקורטור וגם הרכיב הקונקטרי ממשים את הממשק
  + הדקורטור מחזיק בתוכו אובייקט של הרכיב כאשר אנו ממשים את המתודות הנדרשות אנו משתמשים במופע של הרכיב שיש לנו ומוסיפים פונקציונליות – כלומר כותבים תוספות וקוראים למתודה הממומשת
* דוגמא – חנות פיצה: לכל פיצה שלקוח ירצה נחזיר את המחיר של הפיצה (לפי מחיר כל תוספת שמכילה)
  + נגדיר מחלקה לכל תוספת אפשרית.
  + את התוספות נלביש אחת על השנייה בעזרת מקשט שיחבר תוספת לפיצה נתונה.
  + התוספות הן ה-decorator והפיצה היא ה-ConcreteComponent.
  + בדקורטור נחזיק אובייקט של פיצה, נבדוק מהמחיר שלו ונוסיף את המחיר של התוספת הנוכחית שנרצה להוסיף.
* יתרונות עיקריים של דקורטור:
  + דקורטור בא ביחד עם עיקרון single responsibility – כל מודול אחראי על פונקציונליות אחת
  + יכול ליצור קומבינציות של מחלקות מבלי ליצור מחלקה חדשה
  + לא צריך להסתבך עם ירושה
  + ניתן להוסיף פונקציונליות בזמן ריצה – בצורה דינאמית לפי קלט המשתמש
* חסרונות עיקריים של דקורטור:
  + ברגע שיצרנו ערימת שכבות אי אפשר להוציא משהו מהאמצע
  + כשאנו יוצרים אובייקט אנו למעשה קוראים להרבה קונסטרקטורים (יכול להשפיע על זמן ריצה)
* עיקרון הopen-closed – לא נרצה לפתוח בכל פעם שיש לנו משהו להוסיף מחלקה או אובייקט שכבר סיימנו