תכנות ותכן מונחה עצמים – תרגיל בית 4

מגישים:

אלדד וינר – 304901069

ספיר מלכה – 205794001

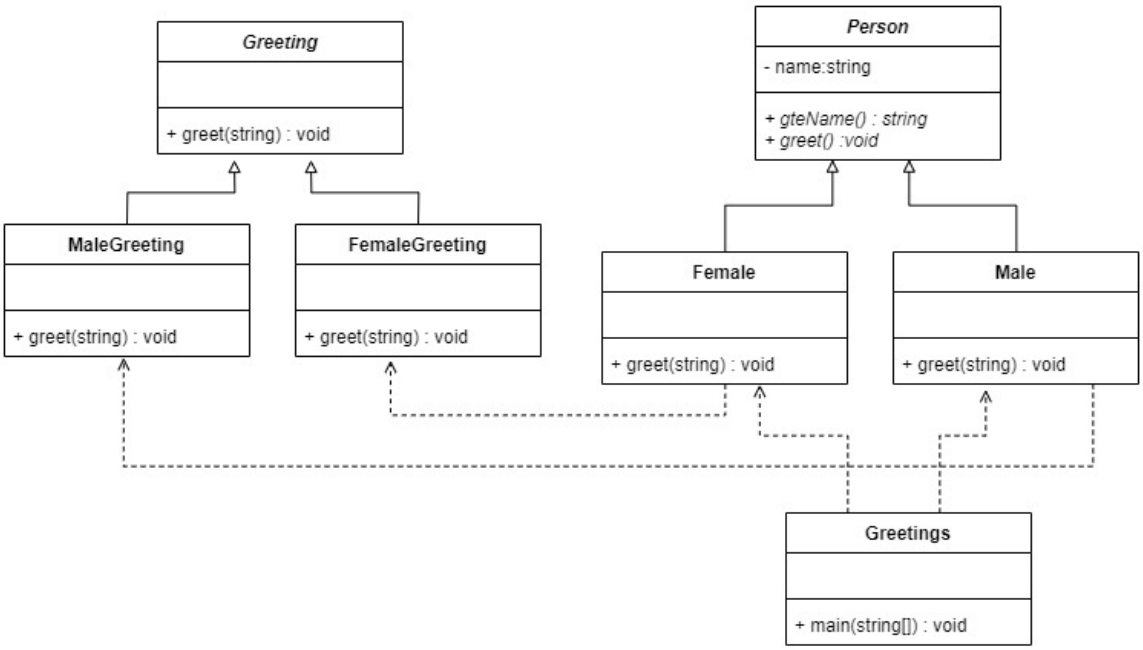
שאלה 1:

א. התוכנית תדפיס:

```
Hello Mr. Danny
Hello Ms. Danna
```

ב. ה-design-pattern שממומש כאן הוא:Factory Method . והבעיה שבאים לפתור כאן היא שנרצה לברך Person ללא צורך בבדיקה מה הsubtype שלו, כלומר ללא ידיעה מראש אלו תת מחלקות שלPerson נייצר נוכל ללא צורך בבדיקות נוספות לברך לשלום בצורה נכונה, בנוסף נוכל להוסיף עוד אופציות שונות לברכות ללא צורך בשינוי הקוד.

ג. Class diagram:

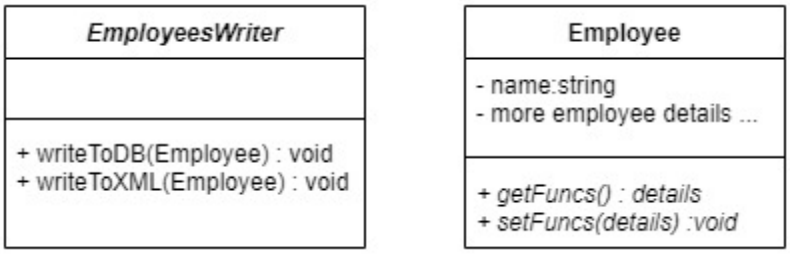


ניתן לראות שיש מחלקה אבסטרקטית בשם Person, המחלקה אבסטרקטית כי יש בה מתודה אבסטרקטית בשם greet() שתפקידה לברך את האדם. את המחלקה ממשים שני הsubtypes: Male וFemale, כאשר כאן כבר ידוע מין האדם אותו יש לברך. בנוסף, יש לנו מחלקה אבסטרקטית בשם greeting שמברכת ללא תלות במין האדם המבורך. את המלקה ממשים שני הsubtypes: FemaleGreeting וMaleGreeting שמתייחסות למין האדם המבורך. מכיוון שMale וFemale יודעים מהו מין האדם המבורך, הם מייצרים אובייקט ברכה בהתאם למין הנכון ומברכים אותו בהתאם.

שאלה 2:

א. המימוש המוצע פוגע בעקרון ה single-responsibility מכיוון שהמחלקה employee צריכה גם להחזיק את פרטי העובד וגם לעדכן אותם בבסיסי הנתונים - שני תפקידים שונים לאותה המחלקה. ע"מ לפתור את הבעיה נייצר מחלקה נוספת שכאשר נרצה לעדכן בסיס נתונים כלשהו, נשלח למתודת העדכון הרלוונטית את פרטי העובד והיא תעדכן את בסיס הנתונים.

תרשים:

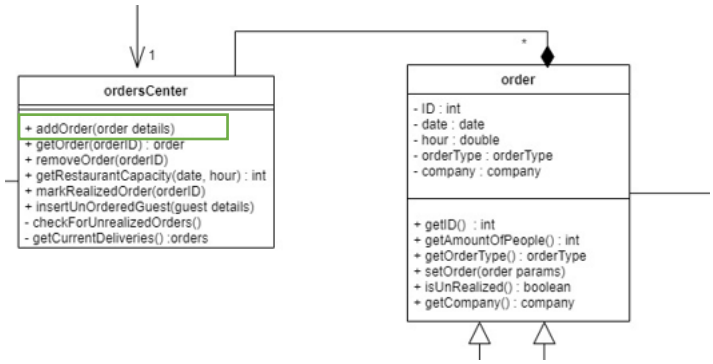


ב. העקרון שנפגע הוא עקרון ההחלפה של ליסקוב מכיוון שריבוע הוא לא true-subtype של מלבן, מכיוון שהמפרט של ריבוע מחליש את המפרט של מלבן (דורש אורך וגובה שווים).

ג. עקרון הפתיחות/סגירות מדבר על יצירת קוד שניתן להרחיבו בקלות ואין צורך לשנות את הקוד הקיים על מנת לעשות כך. עקרון היוצר של GRASP מדבר על כך שמחלקה שמשתמשת באובייקט ויש לה את הידע לגביו-היא האחראית לייצר אותו ולהחזיק בו. בFactory-Method מכיוון שיש לנו concerteCreators שיורשים מcreator יחיד וכל אחד מהם **יוצר לפי המאפיינים והשימוש** שלו בconcreteProduct שיורשים מProduct יחיד, **ניתן להוסיף בקלות עוד** concerteCreator שמשתמש בconcreteProduct (קיים או חדש) בלי לשנות את הקוד הקיים. ראינו כי היוצר של concreteProduct הוא מי שמשתמש בו ויש לו את המידע הדרוש (עקרון היוצר), ושניתן בקלות להוסיף concerteCreator בלי לשנות את הקוד הקיים (פתיחות/סגירות).

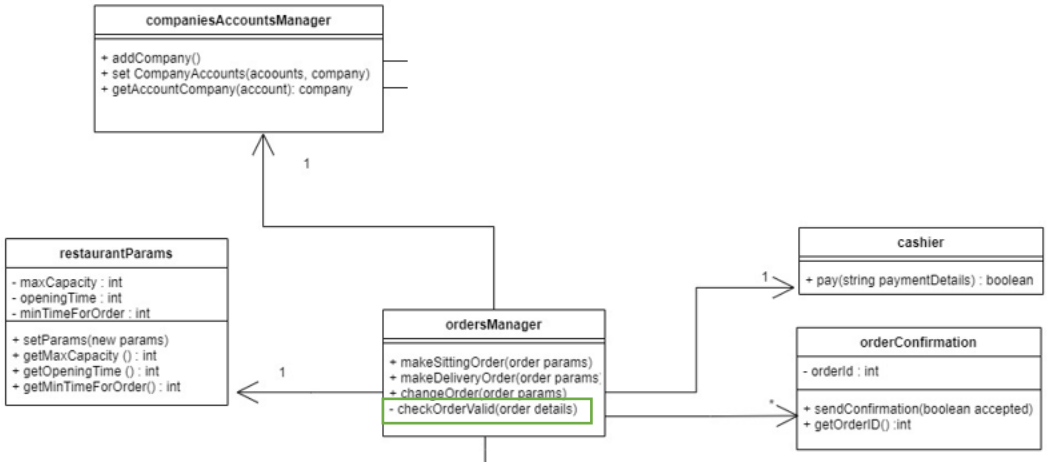
ד. עבור Creator:

המחלקה orderCenter משתמשת ומנהלת את ההזמנות הפעילות ולכן היא יוצרת אותם (המתודה addOrder יוצרת הזמנה ומוסיפה אותה למאגר):



עבור InformationExpert:

מכיוון שהמחלקה ordersManager מחזיקה את האמצעים ומכך את המידע לבדיקת חוקיות של הזמנה (קופה, מאגר נתונים של לקוחות היי-טק והגדרות המסעדה) היא האחראית לבדוק האם ההזמנה חוקית:



ה. בשורה השניה, אנחנו מיצרים אובייקט מסוג border עם תכונות ספציפיות (כלומר התנהגות של Factory-Method), ובנוסף, אנחנו יוצרים את האובייקט דרך ה-borderFactory ולא ע"י ביצוע new, ולכן זהו שימוש בעקרון Abstract Factory.

ו. ממומשת כאן וריאציה של singleton עבור אפליקציה, כלומר כל אפליקציה תוכל להגדיר security-manager אחד, ורק איתו ניתן לעבוד, בנוסף המתודה סטטית ולכן היא מתודה של המחלקה ולא של האובייקט כלומר אין לכל אובייקט של security אחד משלו, אלא אחד לכולם.

ז. Abstract Factory הוא Factory שיש לו גם ממשק ליצירת הconcreteCreators עצמם ולא צריך ליצור אותם בצורה ישירה דרך new.

ח. Facade מספק ממשק פשוט למערכת מורכבת, ולכן דוגמה קלאסית לFacade היא קומפיילר שמורכב מהמון תתי בלוקים והמון שלבים והתניות, אך מבחינת המשתמש יש לו רק מתודה אחת דרכה הוא מקמפל ומריץ את כל המערכת המורכבת הזאת.

שאלה 3:

- **Singleton**:
מכיוון שלכל המערכת קיים רק `ColorGenerator` אחד, ללא תלות במספר לוחות המודעות, יצרנו אותו בsingleton כאשר הc'tor שלו הוא private, וקבלת instance של האובייקט היא רק דרך מתודה סטטית שבודקת אם האובייקט קיים, ורק אם הוא לא קיים יוצר חדש.
- **Observer**:
צבע הלוחות תלוי בצבע ה`ColorGenerator` ויש לעדכן אותם כאשר צבעו משתנה, לכן יצרנו Subject בשם `ColorSubject` אבסטרקטי שמכיל בתוכו: רשימת האובייקטים אותם יש לעדכן, מתודת הוספת מאזין, מתודת הורדת מאזין ומתודה אבסטרקטית לעדכון המאזינים. `ColorGenerator` יורש מ `ColorSubject`, ממש את מתודת העדכון האבסטרקטית, מבצע את הלוגיקה שלו ומעדכן את המאזינים. בנוסף יצרנו ממשק למאזין `ColorObserver` המכיל בתוכו מתודת update וכל לוח מממש את הממשק של `ColorObserver` ע"י מימוש מתודת update שמגדירה התנהגות בעת קבלת עדכון.
- **Strategy**:
יש ל`ColorGenerator` מספר אפשרויות שונות לסדר בו הוא מעדכן את המאזינים שלו כאשר מתבצע אצלו שינוי. ע"מ שיהיה ניתן לשנות בצורה דינאמית את אלגוריתם העדכון, יצרנו ממשק עדכון בשם `UpdateStrategy` וכל אלגוריתם עדכון אפשרי ממש את הממשק. `ColorGenerator` מחזיק אובייקט מסוג זה, ומתודת עדכון לאובייקט כך שבכל זמן ניתן לשנות את אלגוריתם העדכון של `ColorGenerator`