מבחן דוט נט 15.05.19

מיכאל סקטקוב
308301936

<u>חלק 1 - שאלות הגיון</u>

1.

1.1. מחלקות Dog, Cat, Bird יורשות ממחלקת animal. מחלקת SiamiCat יורשת ממחלקת Cat

1.2. כדאי להפוך את Animal למחלקה אבסטרקטית מאחר ואיננה מייצגת מספיק מידע כדיי לייצר אובייקט. כלומר מחלקת cat מספקת לי מספיק מידע על החיה כדיי שאדע לזהותה גם מבלי לדעת אם האובייקט משתייך למחלקת SiamiCat או לא אך לאובייקט שכל הידוע עליו הוא שהוא משתייך למחלקת Animal לא יהיה מספיק מאפיינים ודאיים (למשל האם יש פרווה או נוצות, מספר רגליים, כנפיים, סוגי מזון) כדיי שיהיה אפשר לעשות איתו משהו.

2. לא ניתן לרשת מיותר ממחלקה אחת בדוט נט. הסיבה לכך, לדעתי, היא שמחלקה יכולה להיות לא אבסטרקטית ובכך למעשה לגרום ליצירה של עץ ירושה לא תקין מאחר ואי אפשר לדעת אם שתי המחלקות (או יותר) מגיעות לבסוף לאותו השורש ומטרתן דומה למשל מחלקה אחת יכולה להיות אובייקט ועץ הירושה שלה יוביל למחלקת Object אך מחלקה שניה תייצג שגיאה ותגיעה במעלה העץ שלה למחלקת Exception שאיננה קשורה בשום נקודה לObject. מחלקה שתירש משתיהן לא תוכל להתקיים מאחר והשורשים שלה יגיעו משני מקורות שונים לחלוטין.

3. בדוט נט ניתן לרשת מיותר מאינטרפייס אחד

4. במחלקה לא אבסטרקטית לא ניתן להצהיר על מתודה אבסטרקטית

5. במחלקה אבסטרקטית ניתן להצהיר על פונקציה לא אבסטרקטית

6. באינטרפייס ניתן לרשום רק פונקציות אבסטרקטיות

7. לא ניתן להשתמש ב access modifier מתודה הנמצאת בתוך האינטרפייס

8. ההבדלים בין ארבעתם הם למעשה במיקומים שמהם כל אחד מאפשר לגשת למתודה או אובייקט שאליו הם מוצמדים. כאשר אחד מהם מוצמד למתודה הוא יהיה תקף למתודה זו בלבד אך הצמדה למחלקה היא למעשה התחייבות שכל המתודות במחלקה זו יהיו עם הגבלה זהה או מחמירה אף יותר. Private היא המחמירה ביותר ולמעשה אומרת שרק מתודות בתוך אותה מחלקה יוכלו לגשת אליה. אמנם אי אפשר לקבוע אם protected או internal מחמירה יותר אך אזורי ההגבלה שלהם מתנגשים ולכן מחלקה המוצמדת לאחד לא תכיל מתודה מהשני, protected מאפשר לגשת למתודות במחלקה מתוך כל פרוייקט שנרצה אך רק בתוך מחלקות במורד עץ הירושה שלה ו-internal מאפשר גישה רק מתוך הפרוייקט שבו מוגדרת אותה מחלקה אך איננו מחייב לעשות זאת מתוך מחלקה שיורשת ממנה. Public הוא החופשי מכולם ומאפשר גישה למתודות מכל מקום כלומר מכל פרוייקט ומכל מחלקה שנרצה נוכל לגשת למתודות שנמצאות בתוך מחלקה שהיא public (בתנאי שגם המתודה public כמובן)

9. הצהרה על פונקציה כvirtual אינה מחייבת מימוש ביורשים מאחר שהצהרה זו בניגוד להצהרת abstract מחייבת מימוש בהצהרה הראשונית ורק מודיעה על כך שאולי במורד עץ הירושה שלה אותה המחלקה אחד מהיורשים יחליט לשנותה ואז גם כל יורשיו של זה שהחליף יקבלו את הפונקציה החדשה באותו השם של הקודמת.

10. מימוש מתודה וירטואלית אצל הבן היא Override כלומר פעולה דומה להתעלמות מהכתוב **<u>בגוף</u>** המתודה המקורית (פרמטרים וטיפוס החזרה לא יכולים להשתנות כאן). פעולת overload לעומת זאת היא שימוש בשם יחיד של פונקציה אך בכל פעם שימוש בפרמטרים אחרים או החלפת הטיפוס המוחזר.

11. רק הגדרה של שדה readonly אפשרית בזמן יצירתו.

12. לשדה readonly ניתן לתת ערך בזמן פעולת הבנאי

13. במקרה של מחלקה סטטית לא ניתן יהיה לייצר יותר ממופע אחד אשר יווצר אוטומטית עם עליית התוכנית, כלומר הקוד יכול לפנות לכל מתודה המופיע במחלקה כזו ללא יצירת מופע שלו (במסגרת הגבלות הגישה המצורפים למתודות כמובן) לעומת זאת במקרה של מחלקה לא סטטית יווצר מופע בתחילת התוכנית המכיל רק שדות ומתודות סטטים מהמחלקה הלא סטטית ובכל פעם שיווצר מופע חדש של מחלקה זו במהלך הרצת התוכנית הוא יכיל רק את השדות והמתודות הלא סטטיות.
14.
15.
16.
17.
18.
19.
    19.1. אפליקציית חייגן - stack
    19.2. בקשות למערכת - queue
    19.3. אוסף רשומות - list
    19.4. מאגר לקוחות - dictionary
20. שינוי המימוש נקרא overloading במקום overwriting כי ניתן לבצע שינויים בטיפוס המוחזר ובסוג המחלקה, בנוסף ניתן לבצע יותר משינוי אחד כזה עבור מחלקה אחת למשל אחד עבור פעולה בין שני מופעים מאותה המחלקה ואחד עבור מופע יחיד ומספר אינטגרי.
21.
    21.1. פונקצית ToString - פונקציה וירטואלית אשר במקורה מוסרת את מיקום המתודה שקראה לה בתוך הפרוייקט בו נמצאת המתודה, נהוג להחליף פונקציה זו לפונקציה המייצרת מחרוזת שמתארת את המופע של המתודה.
    21.2. פונקצית GetType - מחזירה את הסוג של האובייקט שקרא לה (למשל int, float. string)
    21.3. פונקצית Equals - מחזירה תשובת נכון\ לא נכון לשאלה אם האובייקט שנשלח שווה לאובייקט שקרא לה
    21.4. פונקציית GetHashCode - מחזירה את מספר ההאש שהוקנה לתוכנית על ידי התוכנית כחלק מיצירתו. מספר זה הוא כמו סוג של ת.ז הקיים לכל אובייקט.

<u>חלק 2 - שאלות קודם קיים</u>

1. הקוד הוא 18618181.8181818
2.
3. בקובץ MyUniqueList.cs

<u>חלק 3 - שאלות הגיון בסיס נתונים</u>

1. שאילתת stored procedure היא שאילתה המאוחסנת בשרת תחת שם מסוים (לרוב שם המתאר את פעולתה), תוכניות אשר פונות לבסיס נתונים זה יכולות לפנות לשאילתה בשמה ולספק לה את המידע הנדרש כדיי לקבל תשובה ממנה במקום לשלוח את השאילתה המלאה לשרת. שימוש בstored procedure מאפשר הפרדה בין עבודה על השרת, למשל שיפור שאילתות כדיי שיבצעו את עבודתן בצורה יותר טובה או החלפה לשרת חדש שגם הוא מכיל שאילתות אלו עם לוגיקה פנימית חדשה, במקום מצב שבו שינוי של כל דבר בשרת יהיה כרוך גם בשינויים גדולים בתוכנית (פחות רצוי כשמדובר בתוכנה שנמצאת כבר בידיהם של לקוחות).

2. בדומה לשאלה הקודמת כדי שלא יהיה צורך לבצע קומפילציה מחדש לתוכנה במקרה של שינוי כתובת השרת רושמים את הכתובת ב app.config ומפנים מהתוכנית אליו ובכך ניתן פשוט לשנות את הכתובת שרשומה שם ובכך להימנע מקומפילציה מחודשת של התוכנית (אחרי הקומפילציה ניתן לבצע זאת ע"י שימוש בעורך טקסט פשוט)
3. במקרה זה מומלץ להשתמש ב SQLITE
4. במקרה זה מומלץ להשתמש ב FIREBASE
5. במקרה זה מומלץ להשתמש ב MSSQL
6. בסיס נתונים MSSQL תומך בפורמט זה
7.
   a. יחס 1:1 מייצג מצב שבו לכל רשומה בטבלה אחת יש רק רשומה אחת שמתאימה לה בטבלה השניה ולהפך לדוגמא,
   b. יחס n:1 מייצג מצב שבו לכל רשומה בטבלה אחת יש רק רשומה אחת שמתאימה לו בטבלה השניה אך בטבלה השניה לכל רשומה יש יותר מרשומה אחת מתאימה בטבלה הראשונה לדוגמא, שתי טבלאות המכילות את כל עובדי החברה וטבלה של מחלקות בחברה - לכל עובד יש מחלקה אחת אליה הוא שייך אך בכל מחלקה יש יותר מעובד אחד.
   c. יחס n:m מייצג מצב שבו עבור כל רשומה בטבלה אחת יש מספר התאמות בטבלה שניה ולהפך לדוגמא, טבלה המכילה הזמנות מחנות וטבלה המכילה מוצרים בחנות - כל הזמנה יכולה להכיל יותר ממוצר אחד (למשל קומקום ומגהץ) וגם יותר מהזמנה אחת יכולה לבקש מוצר מסויים (למשל שתי הזמנות המכילות קומקום)

<u>חלק 4 - קוד C#</u>

שאלה 2 בקובץ Rational.cs