ב"ה

**תשובות למבחן – 15.5.2019**

# חלק 1 – שאלות הגיון

1. המחלקות
   1. להלן הסדר של הירושה
      1. אבא: Animal
      2. ילדים: Dog, Cat, Bird
      3. נכדים: SiamiCat יורש מCat, Buldog יורש מDog.
   2. כדאי להפוך את Animal לאבסטרקטי, כי לא נייצר ממנו אינסטנס.
2. לא ניתן לרשת יותר ממחלקה אחת, משום שאם נירש יותר מאחת, אז פונקציות דומות בתוך המחלקה היורשת לא נדע ממי לירש.
3. כן, ניתן לרשת יותר מאינרפייס אחד.
4. לא ניתן להצהיר על מתודה לא אבסטרקטית בתוך מחלקה אבסטרקטית.
5. כן, במחלקה אבסטרקטית, ניתן לכתוב פונקציה לא אבסטרקטית. היא תהיה פונקציה פנימית שמטפלת בנושאים פנימיים.
6. לא , באינטרפייס כל הפונקציות אבסטקטיות.
7. כן, כל המתודות באינטרפייס הם public. לא ניתן לכתוב access modifier באינטרפייס.
8. להלן ההסבר שלהם:
   1. Internal – רק באותו פרויקט במחלקה אחרת או יורשת, אך לא בפרויקט אחר.
   2. Protected – רק למחלקה וליורשים שלה, גם אם זה בפרויקט אחר.
   3. Private – לאותה מחלקה בלבד, לא ליורשים, לא לפרויקטים אחרים.
   4. Public – לאותה מחלקה, למחלקות אחרות בין אם הם יורשים באותו פרויקט ובין אם הם בפרויקט אחר.
9. פונקציה שמוצהרת כ-virtual לא תהיה חייבת להיות ממומשת ביורשים. לעומת זאת, חובה לממש את כל הפונקציות האבסטרקטיות.
10. מימוש של virtual זה באמצעות override. שתי פונקציות זהות בשמות אך שונות בפרמטרים זה overload. שתי פונצקיות זהות אך שונות בטיפוס הערך המוחזר זה לא override ולא overload, זה exception שהקומפיילר לא יאפשר את זה.
11. Const הוא לא ניתן לשינוי, אחרי שמגדירים אותו פעם אחת. לכן, לא ניתן לשנות בזמן ההגדרה. לעומת זאת, readonly ניתן להגדרה בכל פעם שמגדירים את הערך. זה יכול להתקבל בהגדרת השדה או בבנאי.
12. רק עבור readonly.
13. ההבדל הוא שמחלקה סטטית לא ניתן לייצר אינסטנס שלה, ומחלקה לא סטטית שמכילה שדות סטטיים, היא בעצם 2 מחלקות אחת סטטית ואחת לגמרי לא סטטית.
14. בנאי סטטי מופעל בעת הפעלת מידע סטטי או פונקציה סטטית. בבנאי כדאי לכתוב ערכי השדות (השמות).
15. דוגמא לproperty: public int Student\_ID {get; set;}
16. השימוש בprop חוסך זמן וזכרון. הוא גם עוטף שדה קיים בעזרת GET, SET וגם מייצר DATA (backing field). כן, ניתן לעשות public get; private set;
17. המימוש של הProperty הוא באמצעות getter וגם setter. כלומר יאפשר לקבל את הערך באמצעות return או לשנות אותו באמצעות value = … מה ששמת לו.
18. כן, ניתן לשנות קובץ exe באמצעות assembly מחדש של הקובץ . הexe כתוב בשפת CIL, מי שיודע כיצד לכתוב בה יכול לשנות את הקוד באמצעות פירוק בILDASM ואחרי שסיימת להרכיב את הקוד מחדש באמצעות ILASM.
19. מקרים:
    1. שיחות אחרונות – תשתמש בQUEUE
    2. בקשות לפי הסדר – תשתמש בSTACK
    3. אוסף רשומות – List
    4. מאגר לקוחות – Dictionary
20. שינוי מימוש אופרטור יהיה overloading כדי לאפשר כמה סוגים של מימושים לכל אופרטור.
21. פונקציות של object:
    1. ToString – הדפסת הסוג של המחלקה, נממש אותם כשנרצה להדפיס במחלקה שלנו נתונים או שדות שנרצה.
    2. GetType – מחזיר שם הnamespace והמחלקה שממנה הגיע המשתנה. נעשה override כשנרצה לתת שם לTYPE.
    3. Equals – משווה כתובות בזכרון, נממש כשנרצה להשוות ערכים אחרים במחלקה כשמשווים בין מופעים של מחלקות.
    4. GetHashCode – זיהוי חד ערכי של האוביקט, כמו מספר זהות, נממש כשנרצה לשלוט במספרים שחוזרים.
22. לא רוצים שתבצע שינוי בGETTYPE. שלא נאבד את השם האמיתי שאמור לחזור.
23. מחלקת IComparable
24. מחלקת IComparer
25. חריגות
    1. לגבי הקוד של הCatch יבוצע רק במידה והאקספשיין שייך אליו, נמצא באותו אקספשיין שהcatch מוגדר אליו.
    2. Finally תמיד יבוצע.

# חלק 2 – שאלות קודים קיימים

1. הצופן הוא 18618181.8181818
2. נשים את התשובה בGIT.

# חלק 3 – שאלות בסיס נתונים

1. Stored procedure היא פונקציה שמוגדרת למימוש בשרת בסיס הנתונים. השאילתא מבקשת מהשרת להפעיל פונקציה. כדאי להשתמש בה מכמה סיבות:
   1. אבטחה, פחות אפשרויות להזיק לשרת.
   2. מהיר יותר , טוב עבור שאילתות על מסדים גדולים. וגם שומר קוד "מקומפל".
   3. איש DBA לא צריך להכיר C#, שזה שפה אחרת בשבילו.
   4. אם תיקנתי שאילתא, זה שקוף ללקוח.
2. App.config יהיה יותר נוח לבן אדם לשנות משם את הconnection string בהתאם למה שקיים אצלו, בלי צורך לקמפל מחדש .
3. מומלץ SQLite.
4. Firebase
5. MSSQL
6. תמיכה בJSON – FIREBASE
7. יחסים:
   1. 1:1 – כל לקוח יכול לקבל הזמנה אחת וכל הזמנה מתאימה ללקוח אחד.
   2. 1:n – לכל הזמנה יש מספר מוצרים אבל כל מוצר נמצא רק בהזמנה אחת.
   3. n:m – לכל הזמנה יש מספר מוצרים וגם כל מוצר קיים במספר הזמנות.
8. INNER JOIN
9. LEFT JOIN
10. FULL OUTER JOIN
11. RIGHT JOIN
12. CROSS JOIN
13. POCO הינה מחלקה רק עם מידע, ללא התנהגות ופונקציונאליות. כדאי להשתמש בה כשרוצים לשמור מידע או לסדר שמירת מידע. ראשי תיבות Plain old C# Objects, כדאי לשים בו גם HashCode, וגם ==, =!   
    כדאי בבסיסי נתונים לממש GetHashCode כי כך אתה מוודא תעודת זהות לכל נתון, שונה משאר הנתונים. וגם Equals חשוב משום שאם 2 אוביקטים יגיעו עם אותו HashCode , נבצע השוואה לפי Equals.
14. המחלקה DAO ניגשת למסד הנתונים, שולפת את כל הנתונים ומחזירה אותם ארוזים בLIST. לא הגיוני להדפיס בתוכה או לצייר לחלון.