חלק 1 – שאלות היגיון:

* 1. DOG,BIRD,CAT יורשים מ ANIMAL .

SIAMICAT יורש מ CAT. BULDOG יורש מDOG.

* 1. כן, לא ניצור מופע מANIMAL אלא רק מידע/מתודות מסוימות שנממש במחלקות היורשות.

2. לא ניתן בדוט נט לרשת יותר ממחלקה אחת. כדי שלא יהיו בעיות עם ירושות מרובות. למשל אם מחלקות B,C יורשות ממחלקות A ומחלקהD יורשת גם מB וגם מC, ויש מחלקות שניתנות לדריסה בA שc,B יממשו אך D לא תממש, אז **המתודה של איזו מחלקה Bתירש?**

3. כן , יתן לשרת יותר מאינטרפייס אחד .

4. לא ניתן לבצע זאת, נצטרך לשנות את המחלקה לאבסטרקטית.

5. כן, ניתן לבצע זאת.

6. כן

7. כל המתודות בINTERFACE הן PUBLIC ולא ניתן לתת ACCESS MODIGIERS בתוך INTERFACE.

8. INTERNAL – MODIFIER שניתן להשתמש בו בכל הפרוייקט הנתון.

PROTECTED – MODIFIER שנותן אפשרות לגשת לאותו שדה רק במחלקה יורשת ובאותה המחלקה.

PRIVATE – MODIFIER שמונע גישה למחלקות אחרות, ניתן לגשת רק מאותה מחלקה, גם לא יורשת!

PUBLIC – MODIFIER שנותן אפשרות לגשת אליו מכל מחלקה שהיא.

9. VIRTUAL לא חובה לממש בבנים היורשים. את הפונקציות האבסטרקטיות חובה לממש בבנים.

10. מתודה וירטואלית אשר ממומשת בבן נקרא OVRRIDE – נדרוס אותה ונשתמש בבן.

שתי פונק' זהות בשם אך לא בפרמטרים זו העמסת פונק' – OVERLOAD.

לא ניתן לשכפל מתודה/פונק' – חייב להיות הבדל בין השם/פרמטרים המתקבלים.

11. ג. גם וגם.

12. ב – READONLY

13. מחלקה סטטית – מכילה רק שדות סטטיים, לא מייצרת מופעים, בנאי חייב להיות סטטי, אם יורשים , היורש צריך להיות סטטי גם.

מחלקה לא סטטית – מחלקה שמייצרת מופעים, יכולה להחזיק כל סוגי מידע שהם.

14. בנאי סטטי מופעל אוטומטית לפני שהומפע הראשון נוצר, בפנים ניתן לאתחל שדות סטטיים או להפעיל פעולה כלשהי פעם אחת יחידה.

15. private int speed;

public int Speed

{

get

{

return this.speed;

}

private set

{

Speed = value;

}

}

16. PROPERTY חוסך כתיבת קוד לפונקצ' כמו GET/SET שרוצים לקבל או לאחסן מידע. ניתן לגשת לשדות PRIVATE דרך PROPERTIES. לא ניתן לקבוע שני MODIFIERIS לGET ו ל SET כמו שציינת.

17. ימומש ע"י יצירת BACKING FIELD , ייצור שדה מאחורי הקלעים של אותו שדה מידע שינתן.

18. ניתן לבצע זאת , ע"י ILDASM .

19. – LIFO METHOD – STACK

- FIFO- QUEUE

- DICTIONARY

- LIST

20. לא דורסים את המתודה אלא יש עדיין "העמסה" של המתודה, ממשיכים להשתמש בשתי המתודות במחלקה .

21. TOSTRING – מחלקה שניתן לדרוס , תהיה OVERRIDE כדי שנוכל לשנות את התוכן שתדפיס במחלקות היורשים.

GETTYPE- נשתמש כדי לקבל את סוג ה"שדה"/"אובייקט" כו' שנרצה .

EQUALS – השוואה בין אובייקטים – בין השדה במחלקה העכשווית (THIS) לבין אובייקט אחר. ניתן לבצע OVERRIDE כאשר נשתמש במחלקת בן לשם השוואה אחרת .

GETHASHCODE – לכל שדה/"מידע" שהגדרנו יש HASHCODE – קוד UNIQE אך ורק לו.

22. כל מחלקה יורשת ממחלקת OBJECT , TOSTRING תדפיס את שדות המידע במחלקה בד"כ , נרצה לשנות אותה למען מידע נכון יותר לגבי המחלקה שתירש ולכן נדרוס אותה ,הגיוני שתהיה VIRTUAL .

GETTYPE – לא צריך לדרוס באמת , בכל פעם שנקרא ל GETTYPE נבצע זאת על איזה שדה שנרצה כדי לברר מה הטיפוס שלו, לא נצטרך לדרוס את הפונק' ממחלקת האב..

23. צריך לממש את האינטרפייס ICOMPARABLE

24. נצטרך לממש אינטרפייס – ICOMPARER

25. לא , כי לפמים האקספשן שנצפה לו יהיה שונה, כלומר נקבל NULLEXCEPTION ומה שציפינו לו יהיה ARGUMENTEXCEPTION לכן הקוד ב CATCH לא יעבוד.

FINALLY יבוצע.

חלק3:

1. SP מייצר מקום בזיכרון למידע מסויים וניתן להשתמש בו כדי לאחסן שאילתות כלשהן.

למשל ניתן לאחסן בSP שאילתה לINSERT מידע מסוים ולאחר מכן להשתמש בו דרך .NET כדי לאחסן מידע במאגר נתונים כלשהו.

1. כדאי להשתמש בו כי כך הלקוח יוכל לשנות את פרטי הקונפיג ולקנפג איך שהוא רוצה. לגבי CONNECTION STRING ניתן לשנותו תמיד דרך הCONFIG ולכן נח יותר.
2. SQLITE
3. FIREBASE
4. MSSQL
5. MSSQL
6. 1:1 – לכל לקוח יש הזמנה אחת , כלומר אחד לאחד.

N:1 - לכמה לקוחות יש את אותו המוצר

N:m – MANY TO MANY , לכמה מוצרים יש כמה הזמנות

13. POCO מחלקה שמחזיקה מידע בלבד עם GETERS AND SETERS .

כדאי לממש את EQUALS כדי שנוכל לבצע השוואה בין מידע שנקבל מהבסיס נתונים.

GET HASHCODE כדאי לממש מפני שניתן לקבל את אותו המס' הסודי של אותו אובייקט וניתן לבצע פעולות בבסיס נתונים בעזרת המס' הסודי .

אם לא נממש אותם יכול להיות שניתקל בכל מיני בעיות

14. DAO קיים כדי לקשר בין המחלקה לבין הבסיס נתונים – DATA ACCESS OBJECT .

לא הגיוני מפני שהמחלקה קיימת אך ורק לתת גישה למסד הנתונים.