

## עיצוב מונחה עצמים

### מבחן סיום - מועד א' תשס"ה

- יש לענות על 20 מתוך 22 השאלות. רק 20 התשובות הראשונות יבדקו.
- יש לסמן בדיוק תשובה אחת לכל שאלה. סימון לא ברור או כפול ייפסל.
- משקל כל שאלה הוא 5 נקודות.
- השאלות מתייחסות לחומר הלימוד כפי שנלמד בכיתה.
- משך המבחן שעתיים וחצי.
- אין להשתמש בחומר עזר.

בהצלחה!

מספר ת.ז.: \_\_\_\_\_

- אלו מהתכונות הבאות אינן משותפות לסביבות ההרצה של JVM ושל .NET?
  - שחרור זיכרון אוטומטי
  - טעינה דינמית של מחלקות לזיכרון
  - תמיכה ב-Reflection
  - תמיכה ב-Exceptions
  - יכולות להריץ קוד מקומפל (Bytecode או IL בהתאמה) מגרסאות קודמות וגם מגרסאות עתידיות של סביבת ההרצה
- אלו מהתכונות הבאות הן יכולת מובנה של COM?
  - מימוש כמה ממשקים ע"י מחלקה אחת
  - Automatic Garbage Collection
  - Exceptions
  - ירושה בין ממשקים
  - Reflection
- אלו מהמשפטים הבאים לגבי הקשר בין Sequence Diagrams ובין Collaboration Diagrams נכון?
  - אחד מסוגי הדיאגרמות מציג תהליכים הקורים בזמן ריצה, והשני הוא תיאור סטטי
  - באחד מסוגי הדיאגרמות ניתן להביע תנאים ולולאות, אבל בשני לא
  - שני סוגי הדיאגרמות מציגות את אותו מידע בדיוק, אבל בתצוגות שונות
  - באחד מסוגי הדיאגרמות ניתן להראות העברת פרמטרים בקריאות למתודות, אך בשני לא
  - אף תשובה אינה נכונה
- אלו מהעקרונות העיצוב הבאים מתייחס לחבילות (packages), ולא למחלקות (classes)?
  - Liskov Substitution Principle
  - Law of Demeter
  - Interface Segregation Principle
  - Acyclic Dependencies Principle
  - Dependency Inversion Principle
- מעבד תמלילים מייצג את מבנה הנתונים של מסמך הנערך בו ע"י ממשק הנקרא Glyph, ממנו יורשים לפי תבנית העיצוב Composite אלמנטים פשוטים המופיעים במסמך כגון Image, Character, Row, Table, Page. רוצים להוסיף למעבד התמלילים תמיכה בהערות ובקישורים, כך שניתן לקשר לכל אלמנט גרפי (בין אם פשוט ובין אם מורכב) הערה, קישור או שניהם. מה היתרון של שימוש בתבנית decorator לשם כך, לעומת הוספת הערות וקישורים לממשק Glyph?
  - יציבות: לא נדרש לשנות קוד של מחלקות קיימות, אלא רק לכתוב מחלקות חדשות
  - חיסכון בזיכרון: כיוון שרק אחוז קטן מאד מהאלמנטים במסמך ממוצע כולל הערה או קישור, השימוש ב-decorator יקטין את כמות הזיכרון שמשמך צורך
  - תמיכה לאחור: לא יהיה צורך לתחזק קוד נפרד על מנת לקרוא את הפורמט של מסמכים ישנים (שנוצרו לפני הוספת ההערות והקישורים) וחדשים
  - נדרש לקודד רק במקום אחד איך הערה / קישור מוצגים על המסך, נשמרים וכו' – בניגוד לצורך לממש אותן ע"י כל יורש של Glyph בנפרד, או להוסיף מחלקות נוספות להיררכית הירושה שלו
  - כל התשובות נכונות

6. במעבד התמלילים המתואר בשאלה הקודמת, מומש מנגון Copy & Paste כללי ע"י שכפול (clone) של אובייקט Glyph בפעולת Copy, וחיבורו לאובייקט המכיל אותו במבנה הנתונים של המסמך בפעולת Paste. אלו מהטענות הבאות לגבי המימוש הזה נכונה?
- מימוש כזה יכול לשמש להעתקת אלמנטים פשוטים, אבל לא מורכבים (composites)
  - מימוש כזה לא יכול לתמוך בהעתקה נכונה של decorators המשולבים במבנה הנתונים
  - הוספת מנגון זה דורשת את הוספת המתודה clone() לממשק Glyph, ואת מימושה בכל המחלקות היורשות ממנה, מימוש שעלול להיות לא טריוויאלי
  - מימוש כזה לא יכול לעבוד אם חלק מהמחלקות המממשות אלמנטים גראפיים במסמך (למשל Character) משתמשות ב-Flyweight
  - מימוש כזה יכול לעבוד למימוש Cut & Paste, בו אובייקט משנה את מקומו במבנה הנתונים במסמך, אבל לא למימוש Copy & Paste, כלומר ליצירת עותקים של אובייקט במסמך
7. במעבד התמלילים מהשאלה הקודמת מומש Spell Checker בעזרת תבנית העיצוב Visitor. למה עיצוב זה עדיף על הוספת מתודה בשם checkSpelling() לממשק Glyph(), ומימוש בדיקת האיות במחלקות היורשות מ-Glyph?
- עיצוב זה מקל על שני אנשים שונים לפתח את קוד בדיקת האיות במקביל לקידוד המחלקות היורשות מ-Glyph, בלי להפריע זה לזה
  - בדיקת איות הוא אלגוריתם שדורש זיכרון, ושימוש ב-Visitor נמנע לחלוטין מהקצאת זיכרון לצורך זה עד שהמשתמש מבקש להפעיל אותו
  - עיצוב זה לא דורש את שינוי הממשק Glyph() וכל יורשיו בכל פעם בו נוסף אלגוריתם חדש לתוכנית (למשל בדיקת תחביר, ספירת מילים וכדומה)
  - עיצוב זה לא דורש מימוש כלשהו של בדיקת איות במחלקות המממשות אלמנטים גראפיים במסמך שאינם אותיות ומילים (למשל יורשים של Glyph כמו Circle, Image וכדומה).
  - כל התשובות נכונות
8. אלו מהמשפטים הבאים לא מתאר עיקרון נכון של כתיבה ושימוש בבדיקות יחידה (Unit Tests)?
- בדיקות יחידה צריכות לרוץ ולעבור במלואן כחלק מתהליך הבנייה (build) של המערכת
  - בדיקות יחידה מבוצעות במקום בדיקות לכלל המערכת, כי הן מוכיחות שכל רכיב במערכת תקין
  - בדיקות יחידה צריכות לדעת לדווח אוטומטית אם הן עברו או נכשלו, ללא התערבות אנושית
  - בדיקות יחידה צריכות להיכתב ע"י המתכנת/ת שכותב/ת את הקוד, ולא איש בדיקות אחר
  - בדיקות יחידה צריכות להיכתב לפני או במהלך כתיבת הקוד שהן בודקות, ולא רק אחריו
9. בתוכנית המשתמשת בתבנית העיצוב Command להפעלת כל פקודות המשתמש בה, רוצים להוסיף תמיכה במאקרו: המשתמש יכול להגדיר רצף של פעולות כמאקרו ולתת לו שם, ומאותו רגע מאקרו זה זמין כפקודה רגילה. נדרש לאפשר למאקרו להפעיל מאקרו אחר כחלק ממנו, ונדרש לתמוך ב-Undo למאקרו. באיזו תבנית כדאי להשתמש כדי לממש דרישות אלו?
- Chain of Responsibility
  - Composite
  - Observer
  - Mediator
  - Builder
10. לאילו מבין זוגות התבניות הבאות יש מימוש זהה בקוד?
- State ו-Façade
  - Mediator ו-Observer
  - Template Method ו-Factory Method
  - Builder ו-Abstract Factory
  - Visitor ו-Iterator
11. מתי עדיף להשתמש ב-Class Adapter על פני שימוש ב-Object Adapter?
- אם נדרש למנוע תלות קומפילציה (dependency) בין המחלקה הנעטפת ל-adapter
  - אם המחלקה שאת הממשק שלה עוטפים נכתבה בשפת תכנות אחרת
  - אם שפת התכנות בה עובדים לא תומכת ב-Multiple Inheritance או Multiple Interfaces
  - אם הגדרות המתודות בממשק של המחלקה שעוטפים ובממשק הרצוי תואמות לחלוטין
  - האף תשובה לא נכונה

12. אלו מהחוקים הבאים אינו נובע מעיקרון ה-Liskov Substitution Principle?
- מחלקה יורשת ממחלקה אחרת חייבת לקיים את ה-Invariant של המחלקה שירשה
  - למרות שמחלקה B יורשת ממחלקה A, לא נכון שהמחלקה <B> list יורשת מ-<A> list
  - אם המחלקה B יורשת מהמחלקה A וב-A מוגדרת המתודה foo(A a), אז למחלקה B אסור להגדיר מחדש את חתימת המתודה כך: foo(B b)
  - אם המחלקה B יורשת מהמחלקה A וב-A המתודה foo() הוגדרה כ-public, אז למחלקה B אסור להגדיר את foo() בתוכה כ-protected או private
  - אם המחלקה B יורשת מהמחלקה A וב-A המתודה foo() יש post-condition, אז למחלקה B מותר רק להוריד תנאים מאותו post-condition, ואסור לה להוסיף אליו תנאים נוספים

13. המתודה Commit() במחלקה DatabaseManager כוללת קוד שעובד מול שרת בסיס נתונים ומערכת הקבצים. כיוון שקוד זה עלול להיכשל, כל הפונקציה עטופה בבלוק try..catch, והקוד בבלוק ה-catch הוא:
- ```
catch { throw new CommitException("Commit failed"); }
```
- בהנחה שקוד הפונקציה כולל הקצאת משאבים ושינוי אובייקטים, אלו מהטענות הבאות נכונה?
- קוד זה מפר את עיקרון Exception Neutrality
  - קוד זה מפר את עיקרון Strong Exception Safety
  - קוד זה מפר את עיקרון Weak Exception Safety
  - תשובות ב' ו-ג' נכונות
  - כל התשובות נכונות

14. אלו מהבאים אינו אחד השימושים של כלי Static Code Analysis?
- גילוי באגים בקוד, ובחלק מן הכלים מתן הצעות לתיקון אוטומטי שלהם
  - המלצה על מקומות בקוד הדורשים עיצוב מחדש (refactoring)
  - הקלטה בזמן ריצה של זמן ריצת רכיבים בתוכנית, כדי לאתר צווארי בקבוק
  - גילוי חריגות מסגנון קידוד אחיד (coding standard), סגנון תיעוד וכדומה
  - גילוי בעיות פוטנציאליות של security, portability ונושאים אחרים בקוד

15. תוכנית Java מממשת מנגנון plugins באופן הבא: בעליית התוכנית היא קוראת מקובץ את שמות כל מחלקות ה-Plugins, אז היא טוענת כל מחלקה ע"י קריאה למתודה הסטטית Class.forName(), ואז היא יוצרת אובייקט אחד מכל מחלקה, שמשמש כ-prototype ליצירת אובייקטים נוספים מסוגו. מוצע מימוש חליפי בו במקום לשמור prototypes, התוכנית תשמור את המחלקות שנטענו כרשימת אובייקטים מהמחלקה Class, ותייצר אובייקטים חדשים דרכם. אלו מהטענות הבאות שגויה?
- המימוש החליפי מונע את הצורך לממש מתודת clone() במחלקות ה-Plugin
  - המימוש החליפי חסכוני יותר מבחינת זיכרון
  - המימוש החליפי מחייב שכל מחלקות ה-Plugin יירשו מממשק מסוים (למשל Plugin interface)
  - המימוש החליפי מאפשר לקרוא ל-constructor שונה לפי הצורך בכל יצירת אובייקט
  - מימוש חליפי זה ניתן ליישום גם בתוכנית הכתובה בסביבת .NET.

16. אלו מהטענות הבאות נכונה?
- שילוב בין התבניות Composite ו-Decorator מחייב הידור מחדש של כל ה-Decorators בכל פעם שמוסיפים מחלקה יורשת חדשה למבנה הנתונים שמומש ע"י ה-Composite
  - במימוש של תבנית העיצוב Flyweight, אין הכרח להשתמש ב-Factory משותף – התבנית יכולה להשיג את מטרתה גם אם כל אובייקט ייצור לעצמו עותק נפרד של ה-Flyweight הנחוץ לו
  - אחד היתרונות של תבנית העיצוב Iterator הוא שהמימוש של Iterator ספציפי עבור מבנה נתונים מסוים לא מכיר את המימוש הפנימי שלו
  - Iterator עשוי להיות שמיש או לא שמיש, אחרי שמשתנים נתונים במבנה הנתונים לאחר שה-Iterator נוצר. הדבר תלוי במימוש הספציפי של מבנה הנתונים ושל אותו Iterator.
  - חיסרון מרכזי של התבנית Visitor היא שהפונקציה accept() מחזירה void, ולכן אין למעשה לאלגוריתם מסוים (יורש של הממשק Visitor) שום דרך להחזיר תוצאת חישוב למי שהריץ אותו

17. מבין כל ה-Creational Design Patterns, מתי התבנית העדיפה לשימוש היא Builder?  
 א. כשנוצר אובייקט מורכב, וניתן לבצע הרבה אופטימיזציות עליו לאחר שהוא הוגדר במלואו  
 ב. כשיש רק סיכוי קלוש שבעתיד יחול שינוי בסוג האובייקטים שיש ליצור  
 ג. כשהשיקול המרכזי בבחירה הוא הקטנה ככל שניתן של כמות הזיכרון שהתוכנית צורכת  
 ד. כשנדרשת יכולת להחליף בנקודה אחת בקוד בין משפחות של מחלקות המשמשות ליצירה  
 ה. כשהשיקול המרכזי בבחירה הוא הקטנה ככל שניתן של כמות הקוד שיש לכתוב
18. מבין הפעולות הבאות הנתמכות ע"י מנגנון ה-RTTI של שפת ++C, איזו פעולה אינה ניתנת לביצוע בזמן  $O(1)$ , כלומר בלי תלות במספר המחלקות או עומק ההיררכיה של המחלקות המעורבות בה?  
 א. קבלת שם הטיפוס (כ-string) של אובייקט נתון  
 ב. בדיקה האם שני אובייקטים הם מאותו טיפוס בדיוק  
 ג. ביצוע dynamic\_cast של אובייקט למחלקה אחרת כלשהי  
 ד. ביצוע static\_cast של אובייקט למחלקה אחרת כלשהי  
 ה. אף תשובה אינה נכונה
19. מה מהבאים לא ניתן לבצע בעזרת Reflection בלבד, כלומר בלי Aspects או Dynamic Proxies?  
 א. איפוס כל השדות של אובייקט לברירות המחדל של שפת Java (null, false, 0)  
 ב. יצירת של אובייקט ממחלקה, כך ששם המחלקה נודע לתוכנית רק בזמן ריצה  
 ג. הדפסה למסך של שם המחלקה של אובייקט, ושמות כל הממשקים שהוא מממש  
 ד. הוספת קוד שייקרא לפני כל קריאה למתודה במחלקה נתונה, בלי לשנות את הקוד שלה  
 ה. שמירה לקובץ של כל המידע השמור באובייקט מסוים באופן שניתן לקריאה אחר כך, כולל כל המידע באובייקטים שאובייקט זה מצביע עליהם
20. אלו מהדברים הבאים לא ניתן לעשות ע"י מנגנון ה-Introductions של AspectJ?  
 א. לשנות את המחלקה ממנה המחלקה יורשת (extends), בלי לשנות את קוד המקור שלה  
 ב. למחוק מתודה קיימת ממחלקה, בלי לשנות את קוד המקור שלה  
 ג. להוסיף למחלקה ממשק נוסף שהיא מממשת (implements), בלי לשנות את קוד המקור שלה  
 ד. להוסיף למחלקה מתודה או constructor חדש, בלי לשנות את קוד המקור שלה  
 ה. להוסיף למחלקה שדה (data field) חדש, בלי לשנות את קוד המקור שלה
21. אלו מהבאים אינו אחד מן העקרונות של Extreme Programming?  
 א. Continuous Integration  
 ב. Planning Game  
 ג. Design by Contract  
 ד. On-Site Customer  
 ה. Pair Programming
22. תוכנית המשתמשת בתבנית העיצוב Strategy קוראת מקובץ בזמן האתחול שלה את האלגוריתם בו עליה להשתמש. אם לא הוגדר דבר בקובץ זה, התוכנית תעשה שימוש באסטרטגיה טריוויאלית שלא עושה דבר, כלומר מוגדרת כך: `{ } void run()`.  
`class DefaultStrategy extends Strategy { void run() { } }`  
 מחלקה זו היא דוגמא לשימוש בתבנית:  
 א. Null Object Pattern  
 ב. Decorator  
 ג. Proxy  
 ד. Adapter  
 ה. Façade