עיצוב מונחה עצמים

מבחן סיום - מועד א' תשס"ג

- 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.: _____

1. מטרתה העיקרית של תבנית העיצוב Flyweight היא:
   א. לחסוך זיכרון, תוך שמירת כל הגמישות הקיימת.
   ב. להוסיף גמישות ויכולות חדשות למערכת.
   ג. לשפר את זמן הריצה של המערכת.
   ד. להקל על עבודה במקביל של אנשים שונים על אותה מערכת.
   ה. כל התשובות נכונות.

2. אלו מהטענות הבאות לגבי צורת העבודה הנכונה עם Unit Testing נכונה?
   א. כשמשנים קוד (בין אם ממשק או מימוש של מחלקה), אין צורך לשנות גם את ה-unit tests הבודקים אותו כדי לבדוק את השינויים
   ב. אדם שכותב קוד צריך להיות אותו האדם שכותב את ה-unit tests עבורו
   ג. כתיבת unit tests לא צריכה להתבצע ע"י מתכנתים, אלא ע"י אנשי בדיקות
   ד. המונח Unit מתייחס לחבילה (package): יש להגדיר סט בדיקות עבור כל חבילה
   ה. המונח Unit מתייחס למחלקה (class): יש להגדיר סט בדיקות עבור כל מחלקה

3. אלו מהתבניות הבאות היא היעילה ביותר מבחינת זמן ריצה וצריכת זיכרון:
   א. Prototype
   ב. Singleton
   ג. Factory Method
   ד. Abstract Factory
   ה. Builder

4. אלו מהטענות הבאות איננה נכונה?
   א. מטרת התבנית Visitor היא להפריד בין מבנה נתונים לאלגוריתמים מורכבים הפועלים עליו.
   ב. מטרת התבנית Adapter היא לאפשר שימוש במימוש (implementation) קיים למרות שהממשק (interface) דרכו ניגשים למימוש זה אינו מתאים למקום בו יש להשתמש בו.
   ג. מטרת התבנית Mediator היא להגדיר קשרי יחיד-לרבים בין אובייקטים, כך ששינוי ביחיד יעדכן את כל האובייקטים הקשורים אליו.
   ד. מטרת התבנית State היא לאפשר לאובייקט להחליף בזמן ריצה בין מספר מצבים לוגיים, הנבדלים ביניהם הן בקוד שיש להריץ בכל אחד מהם והן בצריכת הזיכרון שלהם.
   ה. מטרת התבנית Flyweight היא לשתף אובייקטים על מנת לתמוך באופן יעיל וחסכוני במספר גדול של אובייקטים קטנים, הכוללים data members הנמצאים בשימוש נדיר.

5. אלו מהבאים הוא דוגמא למצב בו נכון לזרוק Exception?
   א. מחלקה המממשת FlyweightFactory מקבלת בקשה למצוא אובייקט Flyweight, מחזירה אותו אם הוא קיים או זורקת Exception אם הוא לא, כדי שהקורא ייצור את האובייקט בעצמו
   ב. פונקציה המחפשת האם אובייקט נמצא ברשימה מחזירה את האובייקט אם הוא נמצא בה, או זורקת Exception אם הוא לא
   ג. פונקציה הכותבת מידע לרשת נכשלת בגלל שגיאת חומרה המנתקת את קו התקשורת
   ד. פונקציה במחלקה Graph מחזירה מיון טופולוגי של הגרף, או זורקת Exception אם יש בו מעגל
   ה. כל התשובות נכונות

6. עבור אלו מהדרישות הבאות לא ניתן להיעזר בתבנית העיצוב Proxy?
   א. ניהול מנגנון Copy-on-Write על אובייקט מועתק
   ב. ניהול Reference Count של אובייקט לשם שחרור זיכרון אוטומטי
   ג. ניהול נעילות (למשל ע"י Semaphore) על אובייקט המשותף לכמה תהליכים מקבילים
   ד. עבודה עם אובייקט הנמצא על מחשב אחר באופן שקוף למשתמש בו
   ה. ניהול רשימת מאזינים על אובייקט, שצריכים לקבל הודעה על שינויים בו

7. אילו מהטענות הבאות איננה נכונה?
   א. התבנית Flyweight עלולה לגרום לבזבוז זיכרון אם מתבצעת חלוקה שגויה בין המידע המשותף והלא-משותף במבנה הנתונים המשתמש ב-Flyweight.
   ב. תוכנית מאפשרת למשתמש לבחור בזמן ריצה אחד מארבעה אלגוריתמים לביצוע פעולה; כל אלגוריתם צורך זיכרון רב במהלך ריצתו. התבנית Strategy מתאימה כאן כדי להחליף אלגוריתמים בזמן ריצה, אבל עדיין יהיה צורך להקצות את הזיכרון הנדרש ע"י כל ארבעת האלגוריתמים לכל אורך ריצת התוכנית.
   ג. התבנית Prototype היא חסכונית יותר מבחינת מספר המחלקות בתוכנית לעומת תבניות היצירה Builder ו-Abstract Factory.
   ד. מהדר בעל אופטימיזציה מתאימה יכול להביא לכך שאם מחלקה מגדירה Template Method אך אף מחלקה יורשת אינה משנה (override) מתודה זו, אז לא תהיה שום עלות מבחינת זיכרון או ביצועים לשימוש בתבנית זו לעומת מצב בו המחלקה לא הייתה משתמשת בתבנית זו.
   ה. התבנית Observer מאפשרת לאובייקטים ממחלקות שונות להירשם כמאזינים על אובייקט, כל עוד כל המחלקות השונות הללו מימשו את ה-interface שמגדיר ה-Observer.

8. התבנית Visitor כוללת בכל אלגוריתם (visitor) מתודה לכל מחלקה קונקרטית של מבנה הנתונים עליו האלגוריתם עובד, ומתודות ()accept בתוך מבנה הנתונים הקוראות בכל פעם למתודה המתאימה. מוצע מימוש אלטרנטיבי בו אין מתודות ()accept, וה-Visitor כולל מתודת ()visit אחת המזהה את טיפוס האובייקט שקיבלה ע"י RTTI ופועלת בהתאם. אלו מהטענות הבאות נכונה?
   א. המימוש האלטרנטיבי יהיה מהיר יותר
   ב. המימוש האלטרנטיבי דורש פחות קוד בכל אלגוריתם שצריך לעבור על מבנה הנתונים
   ג. במימוש האלטרנטיבי קל יותר להוסיף מחלקה חדשה למבנה הנתונים, בפרט אם יש כבר מספר אלגוריתמים (visitors) שמומשו
   ד. המימוש האלטרנטיבי לא מאפשר לעבור על מבנה הנתונים ע"י Iterator עליו, תוך הפעלת האלגוריתם על האיבר הנוכחי בכל איטרציה
   ה. אף תשובה אינה נכונה

9. איזה מבנה בקרה לא ניתן להציג ב-UML Sequence Diagram?
   א. פונקציה הקוראת לפונקציה של אותו אובייקט.
   ב. פונקציה הקוראת לפונקציה של אובייקט אחר.
   ג. תנאי (if…then).
   ד. לולאה (for…).
   ה. אף תשובה אינה נכונה.

10. אלו מהטענות הבאות לגבי תבנית העיצוב Command איננה נכונה?
   א. באפליקציה בה יש גישה לפקודות דרך תפריט ראשי, ניתן להגדיר מספר פקודות בתפריט המפעילות אותו command class באופנים שונים ע"י prototype command שונה לכל פקודה.
   ב. ניתן לממש פקודות המופעלות בתנאי כלשהו, המוגדר ע"י המשתמש בזמן ריצה, ע"י יצירת מחלקה שהיא Decorator Command היורשת מ-Command, וכוללת את התנאי שיש לבדוק.
   ג. ניתן לממש יכולת לשמור מספר גרסאות של מסמך בעריכה ע"י הוספת תמיכה ב-serialization של אובייקטים המייצגים פקודות (יורשים מ-Command), ושמירת גרסה חדשה ע"י שמירת אוסף הפקודות המגדיר את השינויים מן הגרסה הקודמת.
   ד. ניתן לממש Macros (כלומר, יצירה בזמן ריצה של פקודה המפעילה אוסף פקודות אחרות) ע"י יצירת מחלקה שהיא Composite Command היורשת מ-Command.
   ה. הדרך היעילה לשלב בין פקודות שניתן לבטל (לבצע עליהן undo) ופקודות שאינן מאפשרות זאת, היא ע"י שימוש בתבנית Chain of Responsibility והפנית בקשה לביטול פקודה שאינה תומכת בביטול לפקודה הבאה אחריה בתור הפקודות שבוצעו.

11. איזו מהתבניות הבאות לא ניתן לשלב בקוד של מחלקה קיימת בלי לשנות את ה-interface שלה?
    א. Flyweight
    ב. Strategy
    ג. State
    ד. Visitor
    ה. אף תשובה אינה נכונה.

12. איזו מהטענות הבאות לגבי מנגנון ה-Dynamic Proxies של שפת Java איננה נכונה?
    א. קריאה לפונקציה על dynamic proxy instance המממש ממשק X היא איטית יותר מקריאה לאותה פונקציה על אובייקט הנוצר ממחלקה רגילה (המוגדרת בקוד) המממשת ממשק זה.
    ב. ממשק שיוצרים dynamic proxy class התומך בו לא יכול לכלול פונקציות המקבלות או מחזירות פרמטרים מהטיפוסים הפרימיטיביים (int, double, boolean), כיוון שמנגנון ה-Reflection יכול לקבל ולהחזיר רק אובייקטים ואינו תומך בטיפוסים הפרימיטיביים.
    ג. מחלקה שהיא dynamic proxy class יכולה לממש כמה ממשקים (interfaces), אך אינה יכולה לרשת ("extends") ממחלקה אחרת.
    ד. הפונקציה invoke() יכולה להיות בכתיבה נכונה Exception Neutral כלפי האובייקט הנעטף
    ה. לפונקציה invoke() מותר לקרוא כחלק מן המימוש שלה למימוש של הפונקציה שהופעלה באובייקט הנעטף, ולהעביר לו את הפרמטרים שהועברו לה או פרמטרים אחרים כלשהם

13. אלו מההגדרות הבאות של Aspect-Oriented Programming איננה נכונה?
    א. Join Point היא נקודה מוגדרת היטב בזמן הריצה של התוכנית
    ב. Pointcut היא הגדרה של קבוצה של join points
    ג. Advice מגדירה קטע קוד שצריך לרוץ באירועים מסוימים, את ה-pointcut שמגדיר את אוסף אירועים אלו, ומתי יחסית ל-pointcut על הקוד לרוץ
    ד. Aspect הוא מודול תוכנה המקבץ בתוכו pointcuts, advices, methods and fields
    ה. Introduction הוא מודול תוכנה אבסטרקטי, המאפשר להגדיר abstract pointcuts ולהרחיב אותם אחר כך ל-aspects ע"י ירושה ממנו

14. בתבנית Chain of Responsibility, אובייקט מקבל הודעה ויכול לטפל בה, או להעביר אותה לבא אחריו אם אינו יודע לעשות זאת. נניח ואנו רוצים לחסוך את הגדרת ה-interface המשותף והקוד המטפל בהעברה הלאה, ואנו מעלים את ההודעה כ-exception במקום כקריאה למתודה. מעבר לשימוש שגוי עקרונית ב-Exceptions, איזה חיסרון עיקרי יש למימוש כזה?
    א. הוא איטי הרבה יותר
    ב. לא ניתן להעביר פרמטרים ב-exception, ואילו בקריאה למתודה כן
    ג. מספר הרמות שההודעה יכולה לעבור עד שתטופל הוא חסום ומוגבל מראש
    ד. רק מחלקה המממשת interface מסוים תוכל להעלות את ה-exception
    ה. לא ניתן להגדיר בתוכנית יותר מסוג הודעה אחד שניתן להעביר באופן זה

15. אלו מהבאים איננו הבדל קיים בין COM Component לבין Java class?
    א. רכיב COM יכול להיכתב במגוון גדול יותר של שפות תכנות
    ב. ניתן להשתמש ברכיב COM ממגוון גדול יותר של שפות תכנות
    ג. רכיב COM אינו Garbage Collected
    ד. רכיב COM רץ כ-native compiled code (במקום בתוך Virtual Machine)
    ה. ניתן להשתמש ברכיב COM במגוון גדול יותר של מערכות הפעלה

16. מה לא ניתן לבצע ב-Java Reflection?
    א. הפעלת מתודה של אובייקט בהינתן האובייקט ושם המתודה כ-string
    ב. יצירת אובייקט בהינתן שם המחלקה שלו כ-string
    ג. שינוי של שדה (data member) באובייקט בהינתן האובייקט ושם השדה כ-string
    ד. הוספת שדה חדש למחלקה בהינתן שם המחלקה ושם השדה כ-strings
    ה. הדפסה למסך של רשימת ה-exceptions שמתודה נתונה הגדירה שהיא עלולה לזרוק

17. באלו מהמקרים הבאים נכון להשתמש ב-assertion?
א. כדי לבדוק תקינות קלט מהמקלדת.
ב. כדי לבדוק תקינות של מבנה הודעה שהגיעה דרך הרשת.
ג. כדי לבדוק שמכפלה של שני מספרים שידוע כי הם חיוביים היא חיובית.
ד. כדי לגרום לזריקת exception, שחלק אחר בקוד אחראי לטפל בו.
ה. אף תשובה אינה נכונה.

18. מתי כדאי לתכנת אוסף מחלקות המבצעות יחד משימה מסוימת (למשל XML Parsing) כרכיב (component)?
א. על מנת למכור אותן כמוצר עצמאי המבצע את אותה משימה
ב. על מנת לשלב אותן בקלות בסביבת פיתוח (IDE) כלשהי, כגון Delphi
ג. על מנת לחסוך במחלקות אלו קידוד של שירותים המסופקים ע"י ה-Framework שבתוכו הרכיב ייכתב לכל הרכיבים בו, כמו למשל ניהול זיכרון
ד. אם השימוש היחידי במחלקות אלו נדרש במסגרת Framework ספציפי, ונוח או יעיל יותר להגדיר מחלקות אלו כך שיירשו מ-interfaces קיימים בו
ה. כל התשובות נכונות

19. אלו מהדברים הבאים לא ניתן ליישם ע"י Aspects, כפי שמומשו ב-AspectJ?
א. קבלת שגיאת קומפילציה אם יש אפשרות שהקוד ב-package הנקרא com.acme.abc יוצר אובייקטים מהמחלקה C
ב. הגדרת מחלקה כממשת את הממשק Serializable, בלי לכתוב זאת בקוד שלה
ג. הגדרה שכל exception שעולה מכל מתודה של מחלקה מסוימת "ייבלע", במקום לעלות למתודה שקראה לה
ד. החלפה מלאה של המימוש של מתודה מסוימת במחלקה – כלומר, הרצת קוד אחר במקום הקוד של המתודה המקורית – בלי לשנות את הקוד של המחלקה וללא ירושה
ה. הוספת שדה (data member) למחלקה, כך שהשדה לא עובד בירושה ליורשים של אותה מחלקה

20. אלו מן הטענות הבאות נכונה?
א. מסגרות (Frameworks) מאפשרות הן Design Reuse והן Code Reuse, אך השימוש בכל מסגרת מוגבל לשפת התכנות בו היא מומשה.
ב. רכיבים (Components) מוגבלים הן לשפת התכנות והן לאילוצי ה-Framework בתוכו נכתבו.
ג. רכיבי COM מוגבלים לסביבת Windows, ואילו רכיבי JavaBeans מוגבלים לשפת Java.
ד. תבניות עיצוב (Design patterns) תומכות ב-Design Reuse אבל לא ב-Code Reuse.
ה. כל התשובות נכונות

21. אלו מהשילובים הבאים בין Creational Design Patterns לא יתקיים במציאות?
א. Builder המשתמש ב-Abstract Factory כדי לייצר אובייקטים
ב. Builder המשתמש ב-Template Methods כדי לייצר אובייקטים
ג. Abstract Factory המשתמש ב-Prototypes כדי לייצר אובייקטים
ד. Abstract Factory שהוא Singleton ובנוי מאוסף של Template Methods
ה. Abstract Factory המשתמש ב-Builder כדי לייצר אובייקטים

22. המימוש של פונקציה מסוימת מבטיח שאם קורה במהלך ריצתה Exception אז הוא תמיד יעלה לפונקציה שקראה לה, ושלא ייווצר resource leak עקב העלאת ה-Exception. עם זאת, אין הבטחה כי האובייקט עליו נקראה הפונקציה יקיים את ה-Invariant של המחלקה שלו אחרי עליית השגיאה. פונקציה זו אינה מקיימת:
א. Strong Exception Safety
ב. Weak Exception Safety
ג. Exception Neutrality
ד. תשובות א' ו-ב' נכונות
ה. כל התשובות נכונות