תכנות ותכן מונחה עצמים
046271

דו"ח הגשה – תרגיל בית רטוב 1

מתן צחור 208936989

פז וולף 206555138

שאלה 1

א. ממומש בקובץ $Shape.java$.

ב. המתודה $clone()$ לא זורקת חריגה מכיוון והמחלקה $Shape$ מממשת את $Cloneable$, ומכילה משתנים פשוטים בלבד. לכן המתודה $clone()$ מסוגלת להעתיק את משתני המחלקה בצורה פשוטה ללא אפשרות שלא תצליח.

ג. ממומש בקובץ $ImpossibleSizeException.java$.

ד. המחלקה $Color$ היא $unmutable$, כיוון ואין לה אף מתודת $mutator$, כלומר לא ניתן לשנות את ערכיה לאחר יצירת מופע של המחלקה. מכיוון ש-$Color$ היא $unmutable$, אין חשש בחשיפת המשתנה הפנימי $color$, כיוון ומשתמש חיצוני לא יוכל לשנותו מבחוץ, גם אם יקבל רפרנס עבורו.

ה. ממומש בקובץ $Shape.java$.

שאלה 2

א. ממומש בקובץ $LocationChangingShape.java$.

ב. ממומש בקבצים $LocationChangingRectangle.java$ ו-$LocationChangingOval.java$.

ג. ממומש בקבצים $LocationChangingRoundedRectangle.java$ ו-
$LocationChangingNumberedOval.java$

ד. ממומש בקובץ $AngleChangingSector.java$

ה. התכן שהציע הסטודנט בעייתי מכיוון שעיגול הוא לא $True\ Subtype$ של אליפסה. למשל, אם נסתכל על המתודה $setSize(Dimension\ dimension)$, עבור מחלקת עיגול נצטרך לדרוש ש-$dimension$ מקיים: $dimension.x == dimension.y$, אך דרישה כזו תחליש את מפרט המתודה ולכן לא ייתכן שעיגול הוא $True\ Subtype$ של אליפסה.

ו. הצעת הסטודנט בעייתית מכיוון שבמצב כזה - לאובייקט LocationChangingOval רגיל יהיה אפשר להעלים חלק ממנו באמצעות מתודות של ה AngleChangingSector דבר שנוגד את המפרט.

ז. היתרונות בהצעה של הסטודנטית הם שלא משתמשים הרבה ב interface $Animatable$ ובמתודות שלה, ולכן כדאי להסירה, ובנוסף הסרתה מפשטת את הירושות בין הclassים שבקוד.
החסרונות של ההצעה היא שצריך לכתוב את אותו הקוד הרבה פעמים (מקטין code reuse), ושהסרת Animatable עלולה להביא לחוסר נוחות – בעזרת הAnimatable הזזת הצורות נעשית בעזרת אותה מתודה אצל כולן.

שאלה 3

א. ממומש בקובץ $Animator.java$.
ב. ממומש בקובץ $Animator.java$.
ג. ממומש בקובץ $Animator.java$.
ד. על מנת להציג ולעדכן את הצורות השונות במבנה ביצענו איטרציה על $Container$ המכיל אובייקטים מטיפוסים שונים. היינו חייבים להחזיק ב-$Container$ אובייקטים מטיפוסים שונים מכיוון שכל צורה שהתוכנה תומכת בה ממומשת ע"י טיפוס שונה. המנגנון ש-$Java$ תומכת בו שאפשר לנו לבצע זאת הוא $Subtyping$ ו-$Interface$. $Interface$ אפשר לנו לממש מתודות של תנועה לצורות שונות שאינן יורשות בהכרח מאותו אב קדמון, ומנגנון ה-$Subtyping$ אפשר לנו להתייחס לכל אובייקט הממומש את ה-$Interface$ שלנו או שיורש מ-$Shape$ כאל טיפוס זהה. בכך יכולנו להכיל את האובייקטים יחד באותו $Container$, לעבור עליהם באיטרציה ולהפעיל מתודות שהאובייקטים חולקים לתנועה וציור.
ה. ממומש בקובץ $Animator.java$.
ו. המחלקה Animator יורשת מ Jframe כי זוהי מחלקה שמאפשרת יצירת חלון GUI שבו ניתן להוסיף ולשלוט באובייקטים (widget) על המסך. ניתן לראות זאת בשורת ההכרזה (Animator extends Jframe), ובכל דבר שמתחיל בJ כמו JMenu ו-Jitem לדוגמא. ניתן לראות שJframe מספקת לנו פיתרון קל ומוכן ליצירת הAnimator עם כל הפיצ'רים שאנחנו צריכים. המחלקה Animator מממשת את ActionListener מכיוון שזהו ממשק שמאפשר הפעלת תגובה במקרה של לחיצה על כפתור או menuitem בGUI – דבר שאנו רוצים בAnimator. ניתן לראות מימוש זה בכל דבר שמופיע בו המילה action בקוד.
ז. הסטודנט לא צריך לשנות הרבה בקוד. כל שעליו לעשות זה לשנות את הטיפוס של $shapes$ ל-$LinkedList$, וכן את הקריאה לבנאי.
זאת מתאפשר מכיוון ו-$LinkedList$ ו-$ArrayList$ מממשים את אותו ממשק - $List$ ובכך כל הפעולות שהפעלנו על $ArrayList$ אפשריות וחוקיות עבור $LinkedList$.