# שאלה 1

1. רצ"ב קוד.
2. במפרט של המתודה clone() במחלקה Shape לא נזרקת אף חריגה מכיוון שהמתודה מטפלת בחריגה הזאת (במידה ותיזרק מהמתודה clone של מחלקת האב – Object) ולא "זורקת" אותה למתודה הקוראת. בפועל החריגה לא אמורה להיזרק ע"י המחלקת האב, מכיוון שמחלקת  
   Shape (ולכן גם כל ה-subtypes שלה) מממשת את הממשק Cloneable.
3. רצ"ב קובץ ImpossibleSizeException.java.
4. המחלקה java.awt.Color היא immutable, מכיוון שאין אפילו מתודה אחת במחלקה שהיא mutator – ניתן לראות את זה גם מהמפרטים וגם מהמימוש. מסיבה זאת, ניתן לממש את המתודות setColor() ו-getColor() כמו שהם מומשו: במקרה של setColor – ע"י השמה של ה-reference המתקבל כפרמטר לתכונה של האובייקט, ובקרה של getColor ע"י החזרת ההפניה למשתנה color של האובייקט, מכיוון שלמשתמש אין דרך לשנות את האובייקט ממחלקת Color ולהשפיע על אובייקט ממחלקת Shape בעקיפין. זאת בניגוד למתודות setLocation ו-getLocation, שמכיוון שמחלקת Point היא mutable, יש צורך ליצור העתק לאובייקט מסוג Point.
5. רצ"ב קוד.

# שאלה 2

1. רצ"ב קוד
2. רצ"ב קוד
3. רצ"ב קוד
4. רצ"ב קוד
5. רצ"ב קוד
6. אנחנו לא מסכימים אם הצעתו של הסטודנט, מכיוון שבמקרה כזה LocationChangingCircle לא יהיה true subtype של LocationChangingOval, מאחר ולמשל, המפרט של מתודה LocationChangingCircle.setSize תצטרך לדרוש שהגובה והרוחב יהיו זהים, וכך יהפוך למפרט חלש יותר ממחלקת האב. לכן עקרון Liskov לא יתקיים.
7. אנחנו לא מסכימים אם הצעתו של הסטודנט. ההתנהגות של LocationChangingOval לא צריכה להיות תלויה איכשהו בהתנהגות של AngleChangingSector. למשך LocationChangingOval לא יוכל להשתמש במתודה step של המופע של AngleChangingSector, מכיוון שמדבור במפרט ספציפי שונה. עוד דוגמה – אם המפרט הספציפי של draw של AngleChangingSector איכשהו ישתנה, זה לא אמור להשפיע על ה-draw של LocationChangingOval. אז גם במקרה הזה LocationChangingOval לא יוכל להשתמש במתודה draw של המופע של AngleChangingSector.

# שאלה 3

1. רצ"ב קוד
2. רצ"ב קוד
3. רצ"ב קוד