**046271 - תכנות ותכן מונחה עצמים**  
**תרגיל בית 0**

**מגישים -**   
 נתי ג'וסף – 302305396

שחף הלר – 307979906

**שאלה 1**

Java Programming Style Conventions

1. **Naming Conventions –**

* **Packages –**

Packages will we written with all lowercase letters.

* **Types –**

Types will be written with mixed casing, where the start of a new word will be uppercase.

* **Variables –**

Variables will be written with mixed casing, where the start of the name will be lowercase, and every word after will start with uppercase.

* **Constants –**

Constants will be written in all uppercase letters.

* **Associated constants –**

Should be prefixed with the common type,  
E.g. COLOR\_RED, COLOR\_BLUE etc.

* **Abbreviations –**

Abbreviations and acronyms will begin with an uppercase letter, followed by all lowercase.

* **Methods –**

Methods, like variables, will be written with mixed casing, where the start of the method will be lowercase, and every word after will start with uppercase.

* **Private variables –**

Private class variables should have an underscore prefix.

* **ALL NAMES SHOULD BE WRITTEN IN ENGLISH –**

A name should only be written in another language, if it seems to be readable,  
and a proper translation to English reduces readability.

* **The Name of the object should not be mentioned in the method -**

E.g. – line.getLineLength() is bad.

* **Get/Set –**

The terms get\set must be used when accessing an attribute directly.

* **Is prefix for Boolean variable and methods –**

Boolean variables and methods should start with the prefix ‘is’.

Other allowed prefixes are ‘was’ and ‘can’ and ‘should’.

* **User interface components –**

User interface components should be suffixed by the element type.

* **Plural form –**

When declaring names of a collection, the plural form of the object should be used.

* **Complement names –**

Complements names must be used for complement entities,  
E.g. get/set, add/remove, create/destroy, etc.

* **Abbreviations –**

Abbreviations should be avoided when possible.

Unless the abbreviation is more known than the actual name,  
E.g. Html, cpu, etc.

* **Exceptions -**

Exceptions should be suffix with ‘Exception’.

1. **Files –**

* **Classes –**

Should be declared in individual files with the file name matching the class name.

Private classes can be declared as inner classes and reside in the file of the class they belong to.

* **File Content –**
  + Lines should be kept shorter than 80 columns.
  + When a line needs to be broken, it should be made obvious and be kept readable.

1. **Statements –**

* **Package –**

The package statement must be the first statement of the package.

This does not include comments to describe the file.

* **Import –** 
  + The import statements must follow the package statement.
  + Most fundamental packages first.
  + Related packages should be written together, with one blank line between groups.
* **Classes –** 
  + Classes must always be listed explicitly.
  + Class and interface declarations should be organized in the following manner:

1. Class/interface documentation.
2. Class or interface statement.
3. Class (static) variables in the order public, protected, private.
4. Instance variables in the order public, protected, private.
5. Constructors
6. Methods.

* **Methods –** 
  + Modifiers should be written in the following order:
  + <access> static abstract synchronized <unusual> final native.
  + <access> MUST always be first.
* **Type conversions –**

Should always be written explicitly.

* **Variables –**

If applicable, variables should be initialized where they are declared.

If not possible, they SHOULD NOT be initialized with a dummy value.

* **Class variables –**

Should never be declared public.

* **Arrays –**

Arrays should be declared with their brackets next to the type, not the name.

* **Loops –** 
  + Only loops control statements should be included in the for()

construction.

* + Loop variables should be initialized immediately before the loop.
  + Avoid do-while.
* **Conditionals-**
  + Complex conditional expressions must be avoided.

Instead, introduce temporary variables.

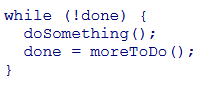
* + The ‘happy’ case should be put in the ‘if’-part and the exception in the ‘else’-part.
  + The conditional should be put on a separate line.
  + Executable statements in conditionals must be avoided.
* **Miscellaneous –** 
  + Magic numbers should be avoided.

Numbers other than 0/1 can be considered declared as named constants instead.

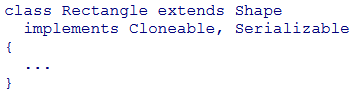
* + Floats should always be written with a decimal point and at least one decimal before and after the point.
  + Static variables and methods must always be referred to through the class name.

1. **Layout and Comments –**

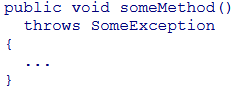
* **Layout –** 
  + Basic indentation should be 4 (Tab).
  + Block layout should be illustrated as in the example –



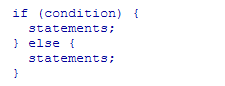
* + Class and interface should have the following form:



* + Method definitions should have the following form:



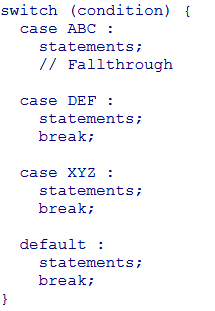
* + If-else statements should have the following form



* + For statements should have the following form



* + The switch statement should have the following form



* + Try-catch statements should be written as the if-else statmeets.
  + Single statement if-else can be written without brackets.

For and while CANNOT.

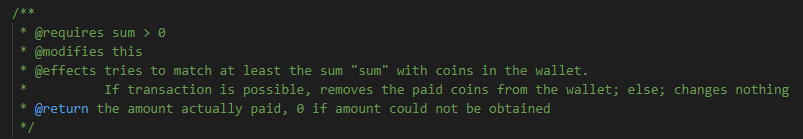
* **White space –** 
  + Operators should be surrounded by a space character.
  + Java reserved words should be followed by a white space.
  + Commas should be followed by a white space.
  + Colons should be followed by a white space.
  + Semicolons should be followed by a white space.
* **Logical blocks –**

Logical units within a block should be separated by one black line (including comments).

* Statements should be aligned wherever this increases readability.

1. Comments-

* All comments should be in English.
* Javadoc comments should have the following form:



* There should be a space after the comment identifier.
* Use // for all non-Javadoc comments, including multi-line comments.
* Comments should be indented relative to their position in the code.
* All public class and public and protected functions within public classes should be documented using the Java documentation conventions.
* All class members should be properly documented.
* Overridden methods must be tagged with @Override.

1. **Best Practices –**

* Classes should be final unless they are explicitly designed for inheritance.
* Accessibility of classes and members should be minimized.
* Mutable members must not be exposed through a public API without explicit notice.
* Mutable arguments must be COPIED when received by public methods.
* Valid argument range of public methods must be documented and validated with IllegalArgumentException.
* The possible value range of the return values must be properly documented.
* The equals() method must be overridden if different objects should report as equals according to the logical model.
* When equals() is overridden, hashCode must be overridden as well.
* Utility classes (with static methods only), should have a private constructor.
* Checked exceptions must be properly handled.
* The toString() method should always be overridden.
* Code must never rely on the toString method.
* Methods that return a collection should never return null. Should return an empty collection instead.
* Level of nesting should be kept as low as possible.

**שאלה 2**

**שאלה 3**

**ב1.** אין באמת צורך לשנות את המתודה.

הדבר היחיד שצריך לעשות הוא לוודא שהארנק (רשימת מטבעות) שלנו ממוינת בסדר עולה לפני תחילת המתודה.  
  
כלומר, ניתן ליצור מתודת מעטפת, אשר מסדרת את הארנק, ורק אז קוראת למתודה המקורית.

**ב2.** כל מימוש שיקיים את המתודה העדכנית, יקיים גם את המתודה המקורית.  
הדבר ההפוך לא נכון.  
ע"כ – המפרט החדש חזק יותר.

דרך אחרת לנסח את התשובה היא – המפרט החדש מקבל אותם קלטים, אך יותר ספציפי לגבי הפלט, וע"כ חזק יותר.

**ג1.** כעת צריך לממש לחלוטין את המתודה ע"י האלגוריתם שניתן לנו (פה, ובאלגו,).

המימוש החדש יהיה רקורסיבי, לכל מטבע ינתן משקל שהוא הערך שלה (שכן אנחנו חייבים הגיע למשקל\סכום כסף מדויק),

וערכו של מטבע באלגוריתם יהיה אחד (אין חשיבות לסוג מטבע, אבל צריכים כמה שיותר מטבעות).

לא נראה פה את נכונות האלגוריתם.

**ג2.** אי אפשר להשוות בין המפרטים,

מכיוון שמימוש שיקיים אחד מהם, לא בהכרח יקיים את האחר.

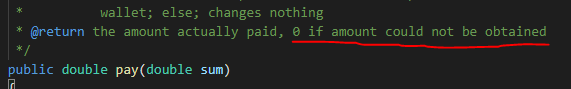
הפלטים שלהם שונים, ואינם מקרה פרטי של אחד או האחר.

**ד1.** אין צורך לשנות את המתודה. בעצם הדרישה הזאת לא מוסיפה שום מידע חדש, שכן צורת המימוש הנוכחית של המתודה מבטיחה לנו שבהינתן שהסכום גדול ממה שיש בארנק, נחזיר גם ככה 0.

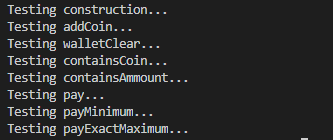
ניתן להוריד את החלק בקוד שמחזיר 0 עבור סכומים שגדולים ממה שניתן לשלם, אך הקוד יעבוד בין אם נוריד את החלק הזה או לא.

**ד2.**  המפרט המחודש מקבל פחות קלטים, אך הפלט שלהם זהה, וע"כ המפרט חדש חלש יותר.

**ד3.** צריך להוריד את החלק המסומן מהקוד



**פלט תכנית הבדיקה:**



**שאלה 4**

**פלט תכנית בדיקה**

