**שבוע 1 – מבוא לעצמים ולג'אווה**

* 1. **מנגנונים בתכנות מונחה עצמים – עצמים ומחלקות:**
* דן ומירי – מבוא לפרקים 1 ו-2:
  + עצמים ומחלקות, קצת בסיס לJAVA, סביבת עבודה ובניית chatterbot – שבוע 1
  + איקס עיגול – 7X7 וצריך רצף של 5 – אינקפסולציה שבוע 2
    - חלוקת הקוד לקפסולות
* עצם, מחלקה, שדות ושיטות:
  + עצם – OBJECT – עולם וירטואלי מורכב מעצמים. למשל במשחק מחשב כל אובייקט שנמצא הוא הוא עצם, וישנם אף עצמים שאחראים על תזוזות של דברים וכדומה.
    - בעולם של אנה – היא עצם, כל עץ או דבר אחר בעולם הוא עצם ואף מזג האוויר הוא עצם.
    - בין העצמים יש רשת של קשרים
    - תכנות מונחה עצמים הוא התורה של בניית מבנים כאלו
  + מחלקה – CLASS – תבנית לעצמים מאותו סוג בעלי אותה התנהגות, תכונות ויכולות. כל עצם משויך למחלקה, והוא מופע שלה (instance). מחלקה מגדירה אלו תכונות (attributes) יהיו למופעים שלה.
    - בעולם של אנה – אנה ושתי חברותיה הן עצמים של אותה מחלקה.
  + תכונות – ATTRIBUTES – כל מחלקה מגדירה את התכונות של המופעים שלה
    - למשל שם
  + שיטות – METHODS - כל מחלקה מגדירה גם את הפעולות שהמופעים שלה יכולים לבצע.
    - למשל לקפוץ
  + בג'אווה ריצת התוכנה היא רצף של שיטות שמפעילות זו את זו.
  + סיכום – כל עצם הוא מופע של מחלקה כלשהי שמגדירה אלו תכונות ושיטות יהיו לעצמים שלה. הריצה של תוכנה מונחית עצמים היא האינטראקציה בין העצמים
  + ביטויים:
    - Data member – משתנה מחלקה, תכונה, שדה
    - Member – איבר, אוסף של שדות ושיטות
    - Method – שיטה, מתודה, פונקציה
    - ערכי השדות של העצם השתנו = המצב של העצם השתנה
* הצגת מחלקה ובנאים:
  + כל המחלקות JAVA יושבים בקבצים מהצורה – name.java
  + מה שמגדיר את המחלקה הוא שם המחלקה class Name, ומתחת נראה את משתני המחלקה.
  + הגדרת משתני המחלקה הם למעשה הגדרת התכונות
  + בהגדרת משתנה צריך לכתוב את טיפוס המשתנה בתחילת השורה לפני השם שלו.
  + לאחר מכן, מוגדרות המתודות של המחלקה –
    - לכל שיטה יש שם, היא יכולה לקבל פרמטרים (נצטרך לכתוב את טיפוס הפרמטר ואת שמו) וכן שיטה יכולה להחזיר ערכים וצריך לציין איזה סוג טיפוס יהיה ערך ההחזרה (ללא ערך – void)
    - אם נכתוב בתחילת שורת ההצהרה של המתודה את המילה public זה אומר שאפשר לקרוא למתודה גם מחוץ למחלקה.
  + יצירת אובייקט של מחלקה:
    - כדי ליצור אובייקט אנו צריכים להשתמש במתודת הבנאי constructor – דרך לקחת את המחלקה וליצור ממנה אובייקט ולתת ערכים שיהיו לאובייקט הספציפי.
    - תכונות קונסטרקטורים בJAVA:
      * שמם יהיה כשם המחלקה
      * לא מחזירים שום דבר
      * בדומה למתודות הם יכולים לקבל פרמטרים
    - איך משתמשים בקונסטרקטור ליצירת אובייקטים של המחלקה?
      * ClassName obj = new ClassName(args)
      * נשתמש בכך מחוץ למחלקה
      * כעת obj מצביע לאובייקט שהוא חלק מהמחלקה
    - על אובייקט שיצרנו אנו להפעיל את השיטות שיש במחלקה ע"י obj.methodName()
  + הדפסה בJAVA – System.out.println(String)
  + זיכרון של התוכנה –
    - לכל מחלקה קיים עותק אחד של הזיכרון בJAVA
    - לכל אובייקט שאנו מגדירים במחלקה אנו צריכים להקצות זיכרון משל עצמו
    - כל אובייקט שייך בדיוק למחלקה אחת – כשניצור אובייקט צריך להיות מקושר למחלקה אחת.
* רפרנסים ופרימיטיבים:
  + כל משתנה יכול להיות:
    - הפנייה לאובייקט מסוים - reference
    - משתנה פרימיטיבי – לא אובייקטים של מחלקות בג'אווה – primitive
  + פרימיטיבים:
    - מחזיקים את זוגי המידע הכי בסיסיים שיש – int (מספרים שלמים), double (שברים עשרוניים), char (תווים בודדים בשפה), boolean (אמת או שקר) וכו'
      * מחרוזת מיוצגת ע"י מחלקה בג'אווה
  + רפרנסים:
    - מצביע לאובייקט – איזשהו חץ שמצביע לאובייקט קונקרטי
    - לכל רפרנס יש טיפוס – שם המחלקה
  + קונטנט:
    - בשורת הגדרת משתנה של ג'אווה אנו רואים שני חלקים המופרדים בסימן =
      * החלק השמאלי ClassName obj מגדיר רפרנס לאובייקט מטיפוס המחלקה
      * החלק הימני הוא יצירת אובייקט של מחלקה – יצירת קונטנט – new ClassName();
  + מצביעים לאובייקט –
    - אם יצרנו אובייקט obj1 ולאחר מכן נכתוב בשורה חדשה ClassName obj2 = obj1 אז המצביע החדש יצביע לאותו האובייקט כמו הראשון, כלומר אם נבצע שינוי עבור אחד מהם נוכל לראות את השינוי גם באובייקט השני
    - אם יצרנו obj1 ולאחר מכן נכתוב בשורה חדשה obj1 = new ClassName() – נקרא שוב לקונסטרקטור – למעשה המצביע יעבור להצביע על אובייקט חדש בזיכרון.
      * לאובייקט הראשון שיצרנו אין כעת מצביע כלומר אין גישה אליו ולכן אין צורך בו – garbage collector "שומר" את האובייקט הראשון ובאיזשהו שלב ישחרר את הזיכרון שלו.
* By reference vs. by value:
  + יש שני סוגי טיפוסים בגאווה – פרימיטיביים ורפרנסים לאובייקטים
  + כאשר אנו שולחים פרימיטיב למתודה אנו למעשה שולחים עותק של אותו המשתנה, לכן אם נשנה את ערך המשתנה במתודה לא תהיה השפעה על המשתנה עצמו בתוכנית כי השינוי יתבצע על העותק.
    - כלומר אי אפשר לשנות פרימיטיב בתוך מתודה
  + אם אנו שולחים רפרנס למתודה אנו למעשה שולחים עותק של המצביע (כלומר עותק של תא הזיכרון של המצביע אליו מועתק הערך שנמצא בתא, אך במקרה של רפרנס זוהי כתובת של האובייקט אליו מצביע), אולם במקרה זה כאשר נבצע שינוי במתודה הוא יתבצע על האובייקט עצמו ונוכל לראות זאת גם מחוץ למתודה (כי השינוי לא התבצע על המצביע עצמו).
  + אם אנו שולחים רפרנס למתודה אך משנים את האובייקט אליו מצביע ומבצעים עליו פעולות, נקבל כי רק העותק של המצביע יצביע לאובייקט אחר ולכן מחוץ למתודה המצביע מצביע לאובייקט המקורי. כן נוכל לראות את השינויים אם יש לנו דרך אחרת לגשת לאובייקט ששונה.
  + סיכום:
    - פרימיטיבייים נשלחים by value – הערך נשלח אבל הכתובת לא
    - אובייקטים מועברים by rreference – מה שעבור זה מצביע לאובייקט ולא האובייקט
    - הרפרנס עצמו הוא רק עותק ולכן אם נשנה אותו זה לא ישתקף בחוץ
* null – למעשה כלום, היעדרות של אובייקט.
  + כשאנו מגדירים מצביע לאובייקט הוא יכול להצביע לאובייקט ממשי קיים או לא להצביע לכלום – נגדירו להצביע לnull.
    - במקרה זה לא ניתן לגשת לאיבר של האובייקט – נקבל שגיאת זמן ריצה NullPointerException
  + כשאנו מגדירים מצביע חדש ולא אומרים לו למה להצביע הוא יצביע באופן דיפולטיבי לכלום, כלומר ערכו יהיה null.
* Strings and Mutability:
  + מחלקת string היא המחלקה הכי נפוצה בג'אווה
  + אפשר להשתמש בסימן = כדי להגדיר אותה – לא צריך קונסטרקטור
    - String s = “” מקביל ל-String s = new String()
  + מתודות של המחלקה:
    - Length() – מחזירה את אורך המחרוזת
    - charAt() – מחזירה את התו במקום שנבקש
  + לSTRING יש תכונה שהיא immutable – לא ניתנת לשינוי
    - ברגע שיצרנו מחרוזת לא ניתן לשנות את ערכי המחרוזת
  + כשאנו כותבים s = s + ”a” אנו לא משנים את s אלא יוצרים מחרוזת חדשה.
  1. **ג'אווה סטרים: ChatterBot:**
* main, IO, import:
  + main היא הפונקציה הראשית של ג'אווה דרכה התוכנית מורצת וההרצה הנגמרת בדר"כ כשאנו מגיעים לסוף הפונקציה
  + את הגדרת main אנו כותבים כך:

Public static void main(string[] args) {}

* + בג'אווה תמיד יש לנו שני שלבים – הקמפול ואז הרצה.
    - הקומפיילר מוודא שכל מה שכתבנו נכון תחבירית.
  + פקודה הדפסה – System.out.println()
    - אם נכתוב רק print לא תהיה ירידת שורה בסוף ההדפסה
  + קבלת קלט – בעזרת מחלקה Scanner
    - התחביר ליצירת אובייקט קבלת קלט –

Scanner scanner = new Scanner(System.in);

* + - צריך לייבא את מחלקת הקלט מהספריה java.util ולכן בתחילת הקובץ נכתוב import java.util.Scanner;
    - קריאת שורה – scanner.nextLine() – מחזיר במחרוזת את השורה הבאה
* class, default constructor, while:
  + מחלקות בג'אווה יכולות להיות באותו קובץ (בדר"כ רק אם הן מחלקות קטנות)
  + אם אנו לא מגדירים בנאי למחלקה יהיה בנאי ברירת המחדל שתיצור ג'אווה מאחורי הקלעים.
    - כאשר נכתוב בנאי (כל בנאי גם אם מקבל ארגומנטים וגם אם לא) הוא ידרוס את בנאי ברירת המחדל.
  + לולאת while – רצה כל עוד איזשהו תנאי מתקיים
    - איך נכתוב? While (<condition>) {}
    - כל עוד התנאי מחזיר אמת הלולאה תמשיך לרוץ
    - == זה בדיקה =זו השמה
* if, else, תיעוד רשמי, הערות:
  + מתודות של מחרוזת:
    - startswith() – האם מתחילה במחרוזת שנכניס בסוגריים כפרמטר – מחזירה אמת או שקר
    - substring() – מקבלת אינדקס התחלה ומחזירה את החלק במחרוזת מאינדקס זה עד הסוף
    - replaceFirst() – מקבלת שתי מחרוזות – מחליפה את המחרוזת הראשונה במופע הראשון שלה במחרוזת שלנו במחרוזת השנייה שניתנה כפרמטר – מייצרת מחרוזת חדשה.
  + הערות – נכתוב // בתחילת השורה
  + ברגע שיש return כל מה שכתוב אחרי שורה זו לא התבצע.
    - למשל אם אנו שמים return ב-if לא צריך את הelse כי אם נגיע לחלק שאחרי הif בהכרח התנאי לא התבצע.
* קבועים:
  + קבוע הוא משהו שנכון לכל הריצה של התוכנית
  + הגדרת קבוע – static final <type> NAME = <value>;
    - Static – לא מקושר למופע אלא לכל המחלקה
    - Final – אי אפשר לשנות את הערך
    - קבוע תמיד באותיות גדולות
  + קבוע הוא שדה של המחלקה אבל הוא לא שדה מופע (לא שייך למופע מיוחד)
  + הגישה לקבוע מחוץ למחלקה היא אחרת – שם המחלקה נקודה שם הקבוע – ClassName.NAME
* המחלקה Random ואתחול שדות:
  + מחלקת RANDOM היא מחלקה שנמצאת בספריית java.util (צריך לייבא בתחילת הקובץ)
  + שיטות של מחלקת RANDOM:
    - nextBoolean() – מחזירה באופן אקראי אמת או שקר
    - nextInt(int) – מחזיר מספר אקראי בין 0 למספר שנקבל כפרמטר
  + הדרך להשתמש באקראיות היא יצירת אובייקט אחד ולהשתמש בו לאורך התוכנית (במקום לייצר כל פעם אובייקט חדש של המחלקה)
* כדי לחיי לחיצת ENTER לאחר הקלדת הקלט נוסיף שורה של scanner.nextLine()
* מערכים:
  + מערך הוא רשימה שאנו מקצים את הגודל שלה מראש.
  + הגדרת מערך –
    - <type>[] array;
      * קיבלנו רפרנס לעצם – מצביע לכלום כל עוד לא ניתן לו ערכים (null)
    - <type>[] array = new <type>[<length>];
    - ניתן לכתוב בסוף השורה בסוגריים מסולסלים את הערכים –
    - <type>[] array = new <type>[2] {a, b};
      * אפשר לוותר על ה-2 שמציין את אורך המערך
      * אפשר לכתוב פשוט <type>[] array = {a, b};
  + גישה לאיבר במערך הוא עם []
  + array.length – מחזיר את אורך המערך, כמה איברים יש בו
  + איברי המערך מאונדקסים מ-0 עד array.length – 1
* ירידת שורה באמצע פקודה היא חספרת משמעות בג'אווה – הפקודה מסתיימת כשנגיע ל;
* This:
  + אם יש לנו פרמטר שיש לו אותו שם כמו אחד השדות של המחלקה, בתוך השיטה הפרמטר יסתיר את השדה.
  + נוכל לגשת לשדה בעזרת המילה השמורה this – מילה זו היא רפרנס לאובייקט עצמו (למופע הנוכחי)
  + כלומר this.field ייגש לשדה של המופע הנוכחי במקום לפרמטר
* לולאות for:
  + מערכים –
    - אפשר לשנות ערכים של מערך
    - כשאנו שולחים מערך אנו למעשה שולחים by reference
    - לכן אם נבצע שינוי של המערך הוא ישפיע גם על האובייקט או המתודה אליו נשלח וגם בחוץ.
  + אם נרצה למנוע שינוי של מערך בתוך מחלקה – בבנאי לא נעשה השמה של הרפרנס למערך אלא ניצור מערך חדש באותו אורך ונעתיק את כל איברי המערך למקום המתאים.
  + תחביר של FOR – for (int I = 0 ; I < <num>; i=i+1);
    - בהגדרה 3 חלקים המופרדים ב-;
      * חלק ראשון מורץ פעם אחת ובו מוגדרת משתנה
      * חלק שני הוא תנאי של הלולאה – הלולאה תרוץ כל עוד התנאי מתקיים – התנאי ייבדק לפני כל איטרציה של הלולאה
      * חלק שלישי – מתבצע בסוף כל איטרציה
* קונוונציות:
  + לשמור על אינדנטציה
  + סוגריים מסולסלים – הפותח באותה שורה של מי שפתח אותם (לא לרדם שורה לפני פתיחת הסוגריים)
  + שם קבוע באותיות גדולות
  + שם מחלקות ושם הקובץ שלהם – אות ראשונה גדולה
  + שתי מילים או יותר בשמות – בלי קו תחתון, אות ראשונה גדולה במילה השנייה והלאה
  + שדות ושיטות מתחילים באות קטנה
* ייעול עבודה:
  + Ctrl + home – תחילת הקובץ
  + Ctrl + end – סוף הקובץ
  + Home פעם אחת ילך לתחילת השורה (אחרי האינדנטציה) ופעם נוספת לתחילת השורה ממש
  1. **טיפים מצביקה – קומפילציה והרצה:**
* Machine Code, Compilation:
  + הקומפיילר לוקח את הקוד שאנו כתבנו ומתרגם אותו לשפת מכונה כדי שהמחשב יוכל להריץ – שהמעבד ידע לקרוא (בינארי)
  + פעולת הקמפול קוראת פעם אחת, נותן קובץ בשפת מכונה ואז מריץ אותו בכל פעם.
* jvm and bytecode:
  + בג'אווה יש לנו עוד שלב באמצע.
  + לג'אווה יש קומפיילר – JAVAC – שמתרגם את הקובץ של ג'אווה לשפה שנקראת bytecode.
    - זוהי שפה יותר פשוטה מג'אווה – פקודות יחסית קצרות ויותר מצומצמות
  + לאחר שקיבלנו קובץ של bytecode ניתן אותו לתוכנה JVM והיא מתרגמת אותה ישירות לשפת מכונה – לכל פקודה בbytecode יש לה תרגום לפקודה בשפת מכונה
    - הJVM הוא כמו מילון
  + למה צריך את זה?
    - ישנם מעבדים שונים – לא כולם מדברים באותה שפת מכונה.
    - כל יצרן יותר מילון JVM שידע לתרגם לשפת המכונה המתאימה לו.
    - התהליך של התרגום לbytecode הוא יקר יותר ונצטרך לעשות אותו רק פעם אחת.
* הרצת קוד ג'אווה:
  + קמפול:
    - ראשית צריך לוודא שאנו באותה תיקייה שבה נמצאים הקבצים
    - הפעלת הקומפיילר – JAVAC file.java
    - לאחר הפעלת הקומפיילר ייווצר לנו קובץ bytecode בתיקייה – סיומת .class
  + הרצה:
    - נכתוב java file (לא צריך סיומת class)
* קימפול והרצת מספר קבצים:
  + נתחיל בלקמפל את כל אחד מהקבצים בנפרד
    - אפשר באותה שורה – javac f1.java f2.java …
  + בשביל להריץ אנו צריכים להריץ רק את הקובץ שבו נמצא הmain שלנו
  + ישנה דרך נוספת לקמפל ביחד את כל הקבצים – javac \*.java – יקמפל את כל קבצי הג'אווה בתיקייה
    - \*.java אומר לו לחפש את כל הקבצים עם סיומת java
  + כתיבת שתי השורות יחד – קמפול והרצה:

Javac \*.java && java mainFile

* + - יקמפל ויריץ באותה פקוד05D4