שיעור שלישי- ג'אווה

טיפ: ניתן לסמן חתיכת קוד הכוללת הערות וכדומה, נסמן ונעטוף ב- /*-------**/ לבסוף נייצא את כל קטעי הקוד המסומנים לקובץ אחד הנקרא "עזרה", לרוב משמש בארגונים גדולים בהם יש הרבה ידיים שעובדות על הקוד.

מחלקת string

בשפות תכנות רבות string מהווה משתנה בסיסי, בג'אווה String הינה מחלקה לכל דבר, מחלקה שנכתבה כחלק מממשק הפיתוח הבסיסי של ג'אווה
מידע מפורט כאן: http://developer.android.com/reference/java/lang/String.html

יצירת אובייקט של המחלקה String ללא הכנסת ערך:

```
String st = new String("");
```

יצירת אובייקט של המחלקה String עם הכנסת ערך לאובייקט

```
String st = new String("maccabi tel aviv ");
```

קבלת אורך המחרוזת באמצעות השיטה length

```
System.out.println( " the length of the  string is " + st.length() );
```

יצירת אובייקט מחרוזת לא באמצעות **new**

השימוש במחרוזות בתוכניות נפוץ ביותר, מסיבה זו ג'אווה מאפשרת קיצור ביצירת אובייקט מחרוזת, וניתן ליצור מחרוזת ללא **new** באופן הבא:

```
String testString = "";
String myName = "William"
```

משתנים לוקאליים ומשתנים גלובליים- סקופ

Scope- האזור בתוכנית בו ניתן לגשת למשתנה מסוים. אזור זה נקבע ויכול להשתנות בשתי דרכים:

1) אופן הגדרת הגישה למשתנה (public , private וכו..).

2) מיקום הגדרת המשתנה קובע את אזור "המחיה" של המשתנה. ישנן 4 אפשרויות לאזורי הכרה של משתנים.

א- משתנים שהוגדרו בתוך מחלקה מסוימת- חיים בגבולות המחלקה אך לא מחוצה לה.

ב- פרמטרים- מוכרים בגבולות השיטה (פונקציה)

ג- משתנים מקומיים של השיטה- מוכרים בתוך הפונקציה בלבד.

ד- משתנים גלובליים- הוצהרו בראשי, מוכרים מכל חבי הקוד.

- **אוסף הזבל**- רץ בכל פעם שיוצאם מסקופ ומנקה את כל מה שלא בשימוש יותר. ברגע שמריצים את הפונקציה הפנימית הוא פותח סקופ, כאשר יוצאים מהסקופ הזה מנקה הזבל מוחק אותו ומנקה את הזיכרון. פעולה ששקופה שמשתמש.

כביכול, יותר נוח להצהיר על כל המשתנים גלובליים כדי שתמיד תהיה אליהם גישה אך האמת היא שזה בא על חשבון מקום בזיכרון ומאריך את זמן הרצת התוכנית. כשמדובר במחשבים זה פחות משמעותי אבל כשמדובר באפליקציות וסלולרי זה קריטי.

Switch

כעקרון הוא רץ הדיוק כמו בג'אווה עם הבדל אחד קטן ומשמעותי! הוא רץ רק עם אינטג'רים לעומת ג'אווה סקריפט שקיבל גם מחרוזות.

השיטה זהה בדיוק למצב שבו היינו רושמים if else.

המרה של טיפוס המשתנה- 2 שיטות

Casting- המרה של טיפוס אחד לטיפוס אחר מאותה משפחה. השינוי הוא בכלי הקיבול.

כדי לשים לב שפעולת הcasting תתבצע מטיפוס "קטן" לטיפוס "גדול" ולא להפך בשביל הוודאות שהוא באמת יצליח להכיל את המידע אם לא הוא פשוט יחתוך את מה שלא נכנס.

דוגמא:

Y=(int)x; הקאסטינג הוא המשתנה אליו עושים את ההשמה.

- אני מקבל ערך מספרי שהוא קורדינטה והוא מסוג (int) ואני רוצה לשלוח לגוגל שמקבל רק לונג, נעשה כך: long coard=(long) x;

שימושי במקומות שצריך לעשות המרות לפונקציות שיודעות לקבל משתנה מסוג מסוים. (כבר לא רלוונטי השיקול של חיסכון בזיכרון)

Valueof- המרה של טיפוס אחד לטיפוס אחר שלא מאותה משפחה.

- קבלת סטרינג מהמשתמש: String num='1000';

String num='500';

נרצה למצוא את num1+num2; התוצאה תהיה 1000500;

התוצאה לא רצויה, נרצה את הערך המספרי של התוצאה ולכן נכתוב כך:

Int n1=Integer.valueof(num1);

Int n2=Integer.valueof(num2);

Float.valueof(num1);

- הערה: לא נוכל לבצע המרה מסטרינג לאינט כאשר הערך הינו אותיות ומספרים- טעות.

הדגמה בקובץ: com.java.cls-----Main.java

תבנית- המרה:

type name = type **.** ***valueof*** **(**name1);

קבלת קלט מהמשתמש

המחלקה Scanner מגדירה פעולות קלט שונות, המחלקה כלולה במארז java.util ולכן אם רוצים להשתמש בפעולות המחלקה יש להכניס הצהרה לפני כותרת המחלקה:

import java.util.Scanner ;

עלינו להגדיר עצם של המחלקה ורק אז נוכל להפעיל את אחת מפעולות הקלט המוגדרות על עצם זה. יצירת עצם מסוג Scanner נעשית כך:

Scanner in = new Scanner(System.in);

הסבר:
המילה Scanner בתחילת המשפט מעידה שאנו מגדירים עצם ממחלקה זו
המילה השמורה new יוצרת עצם בדומה לכל יצירת עצם בג'אווה
הצירוף System.in משמעותו שהקלט מתבצע מלוח המקשים
קליטת ערך שלם

int num = in.nextInt();

קליטת ערך ממשי מסוג double

int num = in.nextDouble();

קליטת ערך ממשי מסוג float

in.nextFloat();

קליטת תו

int num = in.nextInt.charAt(0);

ה-next עומד בקונסול ומחכה עד שיוזן אליו איזה קלט מהמשתמש.

הקלט שהתקבל צריך לעבור השמה למשתנה מאותו טיפוס, במידה והסוג הטיפוס אינו זהה יש לבצע המרה.

- אם לא ידוע מה סוג הקלט שהתקבל נוכל להשתמש ב switch עבור כל סוג של טיפוס.

Encapsulation

עקרון האנקפסולציה - הכמסה מהווה עקרון חשוב בתכנות מונחה עצמים.
כאשר משמשים בעצם חייבים להכיר את הפעולות \ שירותים שניתן לבקש מהאובייקט לבצע או מידע שאנו מבקשים לקבל. האופן בו האובייקט מממש את הפעולות נסתר ממבקש השירות.
יצירה מעין "קפסולה" – "כמוסה" של קוד חסינה ל"פגיעות" מבחוץ יוצרת קוד טוב יותר וסיכוי לשגיאות קטן יותר במיוחד בפרויקט תוכנה גדול עליו עובדים מספר אנשים.
פרטי – private - כאשר לפני הגדרת איבר מופיעה המילה private, ניתן לגשת לאיבר רק מתוך המחלקה\תחום בו הוא מוגדר ולא ניתן לגשת אליו ממחלקות אחרות\ תחומים אחרים.
פומבי – public- כאשר לפני הגדרת האיבר מופיעה המילה public, פירושו של דבר שניתן לגשת לאיבר זה מכל המחלקות ביישום\פרויקט.

constracture

בניגוד להגדרת משתנים בסיסיים (int, boolean וכיו"ב) המקצה זיכרון למשתנים ומאפשרת גישה אליהן, מאותו הרגע, ההכרזה על אובייקט איננה מקצה זיכרון אלא רק מגדירה למפרש שם שיכיל בהמשך את הפנייה לעצם מסוג מסויים ללא יצירת העצם. כדי לאפשר גישה לעצם (חברים, שיטות וכו') עלינו להקצות זיכרון לאותו האובייקט ולעדכן את ההפניה לעצם שנוצר והמקום שקיבל בזכרון – פעולה המתבצעת ע"י האופרטור **new**.

מה מבצע האופרטור new?

כאשר משתמשים באופרטור new מתרחשים מספר דברים, מופע חדש של המחלקה שמופיעה במשפט נוצר, זיכרון מוקצה למופע של המחלקה לפי גודל הזיכרון הנדרש, והשם שנתנו לאובייקט מכיל את ההפניה לאובייקט.

יצירה של מופע מחלקה :

```
chair stool = new chair();
```

הקצאת ערכים לאובייקטים.

```
stool.legs = 1;
stool.color ='g';
```

השיטה הבונה- כדי ליצור מופעים של המחלקה אשר מקבלים ערכים למשתני האובייקט אנו מגדירים שיטות בונות במחלקה. השיטה הבונה יוצרת מופעים לפי תבנית המחלקה ומאתחלת את תכונות העצם.
אם המתכנת לא הגדיר שיטה בונה, ג'אווה "מספקת" שיטה בונה ללא ארגומנטים הנקראת *default constructor*, שיטה בונה זו אינה מבצעת דבר, אינה מבצעת השמה של ערכים אבל נוצרת באופן אוטומטי כי לכל מחלקה חייבת להיות לפחות שיטה בונה אחת.

<u>עבודת כיתה:</u>

1. כתבו תוכנה לקליטת מוצרים והדפסת דוח.
2. מחלקה product תכיל: מחיר, שם וכמות.
3. המשתמש מקבל תפריט: א. הוספת מוצר
   ב. הדפסת דוח
   ג. יציאה.
4. יש ליישם הוספת מוצרים שונים:
   א. הוסף שם.
   ב. הוסף כמות.
   ג. הוסף מחיר.
   ד. חזור.
5. למי שמסיים מוקדם: יש להוסיף ואלידציות: א. מחיר וכמות חיוביים.
6. להוציא את כל הפונקציות מהראשי לקלאס חדש ProductList,

לקרוא לפונקציית run מתוך ProductList מה-main.

- *הסבר תשובה 6: צור קלאס חדש* ProductList

נזין בו את שני המשתנים של המערך ושל האינדקס.

הערה: לא צריך גטר וסטר כי הם ממילא בפרייבט והגישה אליהם היא רק באמצעות פונקציות של פרייבט.

לקלאס זה גזרנו את כל מה שקשור לרשימה מרובת אובייקטים מהראשי שכתבנו בשלבים הקודמים.

- **פתרון מורה בקובץ com.java.products --- ProducList.java , Product.java , Main.java**

<u>דוגמא בכיתה לבנאי:</u>

קלאס קפה {

סוג

חזק/ חלש

סוכר

חלב

סוג חלב- String [] milkType={"soya","cow","other"} אח"כ לולאת פור שמזינה למערך.

קונסטרקטור ()

גטס

סטס

{

קלאס משקאות{

קפה [][12]...

1. פונקציה שמאתחלת את המערך בקפה שחור 1 סוכר ללא חלב
2.

}

<u>שיעורי בית:</u>

א. כתבו תוכנית לחישוב שכר

1. כתבו calss בשם Employee משתנים: שם, כתובת, טלפון, כמות שעות, מחיר לשעה
2. כתבו Class בשם EmployeeList משתנים: מערך של employee בגודל 100

פונקציות: -1. הוספת עובד. 2. עריכת עובד. 3. הצגת רשימת עובדים. 4. הצגת דוח שכר.

3. ב Class Main תתבצע הצהרה של אובייקט EmployeeList להרצת הפונקציה run בלבד.

ב. הוסיפו לתרגיל פרודקט שביצענו בכיתה את האפשרות "עריכת פריט"

1. הוסף פריט
2. ערוך פריט- ערוך שם, ערוך מחיר, ערוך כמות.
3. הצגת דוח
4. יציאה

- הקידו לשמור על אינקפסוליישן כל הפונקציות לעריכה ירוצו מתוך קלאס פרודקט.

ג. השלימו את הפרוייקט קופיי

1. אפשרו תפריט- הכן קפה: סוג קפה, כמות סוכר, סוג חלב, חזק/ חלש.
2. הצג סוגי חלב
3. הצג סוגי קפה

- בחירה ב-1 תציג לבסוף את הקפה המבוקש.