ג'אווה- שיעור רביעי

חידוד- לגבי constractur (=בנאי) הבנאי תמיד קיים, בין אם הגדרנו אותו ובין אם לא. אם לא הגדרנו הוא נקרא constractur default (=בנאי ברירת מחדל) הקיום שלו הכרחי לתכנות, בלעדיו לא הייתה הקצאת מקום למשתנה!
לאחר שהגדרנו את הבנאי הוא משתלט על הבנאי ברירת מחדל ומקצה מקום גם למספר משתנים המשויכים לאובייקט (פרמטרים) ומחייב את המשתמש להזין אותם. (סוג של ולידאציה)

- ייצור מערך **אינו** יצירת אובייקט (בניגוד לג'אווה סקריפט!) המערך הוא אופציונלי להזנת אובייקטים והוא מכיל null בכל אחד מתאי הזיכרון. הוא רק נותן הקצאת מקום ל-מספר תאים לא מוגדרים וריקים. עם הגדרת הקונסטרקטור ("ניו") הוא בונה את התאים וערכם כבר לא null וכעת שמו של התא הוא arr[i] וכבר יש לו מקום ממשי.
- בלי "ניו" אין בנאי ולכן לא נוכל להזין דבר לתא כי אין לו לאן להיכנס, לכן חייב להזמין את הבנאי שיבנה את השטח המיועד וזה על ידי "ניו". לתוך האובייקט אפשר להזין ערכים אך ללא הצהרה על אובייקט לא קיימת הקצאת מקום ולכן אין לערכים איפה לשבת ומתקבלת טעות.
- אובר לודינג- היכולת שלנו לבצע העמסה של מספר פונקציות עם שם זהה וערכים שונים במושג זה נכללת גם היכולת להגדיר מספר בנאים שהם שונים. גם פונקציות זהות שאחת מחזירה void ואחת מחזירה int וכו'. (=מושג שיש ללמוד בע"פ)!

Overloading- העמסת שיטות.

שיטות בעלות שם זהה אך עם מספר פרמטרים, או פרמטרים מסוג שונה, יכולות להימצא באותה מחלקה. לעיתים אנו נדרשים לאותה שיטה אך עם מספר פרמטרים או סוג פרמטרים וסדר הופעתם שונה וג'אווה מאשרת מצב זה הנקרא "העמסת שיטות", כי החתימה של השיטות שונה.

דוגמא:

```
public int big(int sum, int tax );
public double big ( double sum , double tax ) ;
```

שתי השיטות מקבלות שני מספרים ומחזירות את הערך הגדול, אולם שיטה אחת מקבלת שני מספרים מטיפוס int, והשיטה השנייה מקבלת שני מספרים מסוג double

- אנו מרבים להשתמש ברעיון של העמסת שיטות בעת כתיבת שיטה בונה, במקרים רבים נכתוב יותר משיטה בונה אחת מצב המאפשר לנו גמישות ביצירה ואתחול של אובייקטים גם שיטה בונה מעתיקה מהווה העמסת שיטה.

מושגים ומונחים חדשים

Catch & try

שיטה המשמשת להגנה על הקוד וגם שליטה טובה יותר על הודעות שגיאה של ג'אווה.

```
Try{

}catch(exception e){

}finaly{

}
```

הסבר:
e- הפרמטר שמחזיק את השגיאה.
Try- עשה את הפעולות הבאות, המופרטות בתוך הבלוק.
Catch- במידה ו try לא יצליח אז עשה את catch.
finaly- הפקודות שבבלוק זה תמיד ירוצו, בין אם הצליח ובין אם לא.
דוגמא:

```
Public Boolean isAnumber(String str);{
Try{
int x= Integer.valueOf(str);
} catch(exception e){
Return false;
}
Return true;
}
```

- throws- סוג נוסף (=לא רלוונטי כרגע)

static = קבוע
הגדרת משתנה כקבוע. התא המוגדר כ- static יהיה קבוע עבור כל אחד מהאובייקטים ויכנס אל כל אובייקט בצורה אוטומטית.
יתרה מכך, הזנת ערך חדש למשתנה המוגדר כקבוע ישפיע אוטומטית על כל האובייקטים.
פונקציית static- היא תהיה קבועה בשביל כולם. באזור הסטטיק נרשמות גם פונקציות ובאזור זה תירשם גם הפונקציה המוגדרת והיא תירשם לכולם.
דוגמא: Integer.valueOf אינטג'ר זה קלאס ואפשר להשתמש בו עבור כל טיפוס.
final
מרגע שהצהרתה על משתנה- הוא לא הולך יותר להשתנות- הופך את הערך לקבוע.
מימושים- בלתי ניתן לשינוי אלא באמצעות שינוי ידני בתוך הקוד
וכן, פיינלים נוצרים בתחילת הקוד ונשמרים בכל האפליקציה, אלא, אם הפינאל מוגדר בסקופ.
וכן ניתן להגיע אליהם מכל מקום כי הם קבועים. כשמכריזים על פינאל בתחילת האפליקציה העלאה הראשונה של הפונקציה איטית יותר אך הריצה שלה מהירה יותר כי הפינאלים- הקבועים כבר עלו ונשארו בזיכרון.
היתרון הגדול הוא בביצועים ובנגישות.
כל הקבועים שלא הולכים להשתנות במהלך האפליקציה יקבלו final static.
מקובל לבנות קלאס שנקרא- פרמטר או קונטאנס ובו נגדיר את כל הקבועים שלנו אם public static או פיינאלים.
כאשר הוא קבוע שניתן לשינוי- סטטיק
קבוע שאינו ניתן לשינוי- פינאל.

שיעורי בית
- לסיים את השיעורים של שיעור שעבר- employee, product, coffe
- בכולם ליישם הגנה מפני שגיאות בהכנסת נתונים (try catch)
- בכולם להוסיף כותרת בפתיחת התוכנה- תוכן הכותרת ימומש במשתנה מסוג פיינאל סטטיק. VER NUMBER 1 (מספר גירסא)
- הוסיפו את האפשרות לחישוב מחיר לפי שער הדולר, יש להוסיף משתנה סטטי dollerRate.