

# תוכנה 1

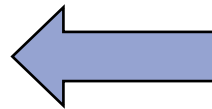
תרגול 4: מחלקות ועצמים  
נעמה מאיר ומתי שמרת

# תכנון תוכנה למערכת בנקאית

## תכנון מערכת תוכנה עוסק במיפוי בין עולם הבעיה ועולם הפתרון

### ■ עולם הפתרון:

- שפת תכנות
- עצמים
- מחלקות
- שירותים
- שדות



### ■ עולם הבעיה:

- בנקים
- לקוחות
- משיכות, הפקדות
- חשבונות
- יתרות

# תכנון מחלקה לייצוג חשבון בנק

- בגישה מכוונת עצמים מייצגים ישויות מעולם הבעיה ע"י ישויות בשפת התכנות
  - כל שם עצם מעולם הבעיה מועמד לייצוג ע"י מחלקה
  - יש להיזהר לא להיצמד בקנאות לעולם האמיתי (דוגמא: פקיד בנק שעושה הכל)
- נתכנן מחלקה לייצוג חשבון בנק
  - נהפוך תיאור המילולי של חשבון בנק לרכיב תוכנה עם מצב פנימי וסט פעולות אפשריות
  - תאור הפעולות יתבטא בחוזה (שיעור הבא) ובמתודות המחלקה

# המחלקה BankAccount

- מייצג חשבון בנק
- באילו פעולות תומך?
- איזה מידע שומר כדי לאפשר את הפעולות?
- האם קיימים תנאים שהם תמיד נכונים?
  - לכל חשבון יש לקוח
  - לכל חשבון יש מספר מזהה חוקי

# שרותי המחלקה

ישנם 3 סוגי שירותים (מתודות, פונקציות, פרוצדורות)

## ■ שאילתות (queries, accessors)

- מחזירות ערך ללא שינוי המצב הפנימי
- כגון: בירור יתרה

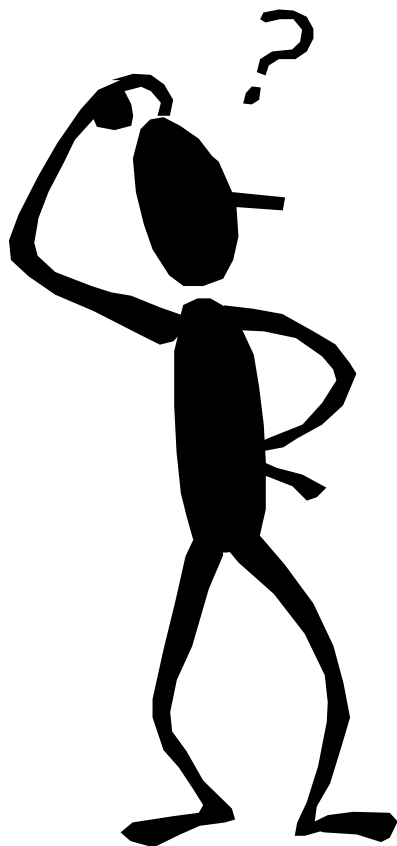
## ■ פקודות (commands, transformers, mutators)

- מבצעות שינוי במצב הפנימי של העצם
- כגון: משיכה, הפקדה

## ■ בנאים (constructors)

- יצירה ואיתחול של עצם חדש
- כגון: יצירת חשבון חדש

# שאלות BankAccount



■ ברור יתרה:

- ארגומנטים?
- מה טיפוס הערך המוחזר?

■ פרטים על החשבון:

- מספר חשבון?
- פרטים על בעל החשבון?
  - ❖ תעודת זהות?
  - ❖ גיל?

# שאלות BankAccount

```
public class BankAccount {  
    public double getBalance() {  
        ???  
    }  
    public long getAccountNumber() {  
        ???  
    }  
    public Customer getOwner () {  
        ???  
    }  
}
```

השאלות הן מצג לעולם החיצון.

שאלות

מצב  
פנימי

```
    ???  
    ???  
    ???
```

# שאלות BankAccount

```
public class BankAccount {  
    public double getBalance() {  
        return balance;  
    }  
  
    public long getAccountNumber() {  
        return accountNumber;  
    }  
  
    public Customer getOwner () {  
        return owner;  
    }  
}
```

שאלות

מצב  
פנימי

```
private double balance;  
private long accountNumber;  
private Customer owner;  
}
```

- מוסכמה: הגישה לשדה field תעשה בעזרת המתודה getField().
- שמירה על מוסכמה זו הכרחית בסביבות GUI Builders ו-JavaBeans



# המצב הפנימי

- המצב הפנימי של עצם מיוצג ע"י נתוניו (שדותיו)
- מאפשר מימוש שאילתות/פקודות
- שדות עצם הם לרוב עם הרשאת גישה פרטית
- במקרה של חשבון בנק:
- מצב פנימי: מכיל מספר חשבון, יתרה ולקוח
- מאיזה טיפוס?

```
public class BankAccount {  
    ...  
    private double balance;  
    private long accoutNumber;  
    private Custome owner;  
}
```

# getter/setter

- יש חשיבות לגישה לנתונים דרך מתודות. מדוע?
- לא כל שדה עם נראות פרטית (`private`) צריך `getter/setter` ציבורי
- יצירה אוטומטית של שרותים אלו עבור כל שדה פוגמת בעקרון הסתרת המידע
- למשל: עבור השדה `balance`
  - האם דרוש `getter`?
  - כן, זהו חלק מהממשק של חשבון בנק
  - האם דרוש `setter`?
  - לא בהכרח, פעולות של משיכה או הפקדה אמנם משפיעות על היתרה, אבל פעולה של שינוי יתרה במנותק מהן אינה חלק מהממשק

# פקודת ה-'להפקיד'

■ המתודה: deposit

■ סכום הכסף המופקד מתווסף ליתרה בחשבון

■ ארגומנטים?

■ ערך מוחזר?

■ מהן הנחות המימוש?

מוסכמה: שמות פקודות  
הם שמות פועל



# פקודת ה-'להפקיד'

```
/**  
 * Make a deposit to the account  
 * Requires ...?  
 * Ensures ...?  
 */  
public void deposit(double amount) {  
    balance += amount;  
}
```

# פקודת ה-'להפקיד'

```
/**  
 * Make a deposit to the account  
 * Requires that amount is non-negative  
 * Ensures that the balance after the operation is the  
 * balance before it plus the amount  
 */  
public void deposit(double amount) {  
    balance += amount;  
}
```

# פקודת ה-'למשוך'

■ המתודה: withdraw

■ סכום הכסף המבוקש יורד מיתרת החשבון.

משיכת יתר (אוברדרפט) אינו אפשרי.

■ ארגומנטים?

■ ערך מוחזר?

■ מהן הנחות המימוש?

# פקודת ה-'למשוך'

```
/**  
 * Withdraw amount from the account  
 * Requires ...?  
 * Ensures ...?  
 */  
public void withdraw(double amount) {  
    balance -= amount;  
}
```

# פקודת ה-'למשוך'

```
/**  
 * Withdraw amount from the account  
 * Requires  $0 < \text{amount} \leq \text{getBalance}()$   
 * Ensures that the balance after the operation is the  
 * balance before it minus the amount  
 */  
public void withdraw(double amount) {  
    balance -= amount;  
}
```



# דיון – העברה בנקאית

מספר חלופות למימוש העברת סכום מחשבון לחשבון:  
■ אפשרות א: מתודה סטטית שתקבל שני חשבונות  
בנק ותבצע ביניהם העברה:

```
/**  
 * Make a transfer of amount from one account to the other  
 * Requires ...?  
 * Ensures ...?  
 */  
public static void transfer(double amount,  
                             BankAccount from,  
                             BankAccount to) {  
    from.withdraw(amount);  
    to.deposit(amount);  
}
```

# דיון – העברה בנקאית

מספר חלופות למימוש העברת סכום מחשבון לחשבון:  
■ אפשרות א: מתודה סטטית שתקבל שני חשבונות  
בנק ותבצע ביניהם העברה:

```
/**  
 * Make a transfer of amount from one account to the other  
 * Requires  $0 < \text{amount} \leq \text{from.getBalance}()$   
 * Ensures that amount has been deducted the 'from' account and added to  
 * the 'to' account  
 */  
public static void transfer(double amount,  
                             BankAccount from,  
                             BankAccount to) {  
    from.withdraw(amount);  
    to.deposit(amount);  
}
```

# דיון – העברה בנקאית

אפשרות ב:

```
/**  
 * Makes a transfer of amount from the current account to  
 * the other one  
 */  
public void transferTo(double amount,  
                        BankAccount other) {  
    other.deposit(amount);  
    balance -= amount;  
}
```

# דיון – העברה בנקאית

אפשרות ג: העמסת withdraw ו/או deposit שיקבלו שני ארגומנטים (סכום והפנייה לחשבון נוסף):

```
/**
 * Make a transfer of amount from other to the current account
 */
public void deposit(double amount, BankAccount other) {
    other.withdraw(amount);
    balance += amount;
}
```

# בנאי

- תפקיד: ליצור עצם חדש ולאתחל אותו באופן כזה שהתנאים שחייבים להתקיים אכן מתקיימים לאחר האיתחול
- בנאי לא אמור לכלול לוגיקה נוספת פרט לכך
- יש דברים שאינם באחריות המחלקה. למשל:
  - מי דואג לתקינות מספרי חשבון? (למשל שיהיו שונים)
  - מי מנהל את מאגר הלקוחות?

# בנאי BankAccount

```
/**
```

```
* Constructs a new account and sets its owner and identifier
```

```
* Requires a valid id and that customer is not null
```

```
*/
```

```
public BankAccount(Customer customer, long id) {  
    accountNumber = id;  
    owner = customer;  
}
```

# final

- חשבון בנק מזוהה חד-חד ערכית עם עצם לא משתנה של `accountNumber`. לכן, נהפוך שדה זה ל-  
**final**

```
final private long accountNumber;
```

**Blank final field:** a final field that hasn't been initialized at creation

- שדה *blank final* יש לאתחל פעם אחת בדיוק, בתוך הבנאי של המחלקה.
- אכיפה ע"י הקומפיילר: במקרה של השמות נוספות למשתנה `final` תהיה שגיאת קומפילציה

# שיעור הבא

■ נשלים את דוגמת הבנק

■ נבדוק את הקשר בין המחלקות השונות במערכת

Bank .

Customer .

BankAccount .

■ נראה דוגמה לשימוש במערכת