


## תכנון תוכנה למערכת בנקאית

תכנון מערכת תוכנה עוסק במיפוי בין עולם הבעיה ועולם הפתרון

- |                       |                      |
|-----------------------|----------------------|
| □ <u>עולם הפתרון:</u> | □ <u>עולם הבעיה:</u> |
| ■ שפת תכנות           | ■ בנקים              |
| ■ עצמים               | ■ לקוחות             |
| ■ מחלקות              | ■ משיכות, הפקדות     |
| ■ מתודות              | ■ חשבונות            |
| ■ שדות                | ■ יתרות              |
- 

## תוכנה I בשפת Java עיצוב מחלקה ותיכון על פי חוזה

תרגול מספר 5  
אורנית דרור ואוהד ברזילי

## המצב הפנימי

- המצב הפנימי של עצם מיוצג ע"י נתוניו (שדותיו)
- שדות עצם הם בד"כ עם הרשאת גישה פרטית
- במקרה של חשבון בנק – היתרה
- מאיזה טיפוס?

```
public class BankAccount {  
    ...  
    private ??? balance;  
}
```

## מחלקה לייצוג חשבון בנק

- בגישה מוכוונת עצמים כל שם עצם מעולם הבעיה הוא מועמד לייצוג ע"י מחלקה
- נתכנן מחלקה, `BankAccount`, לייצוג חשבון בנק
- ננסה להפוך את התאור המילולי והתפיסה האינטואיטיבית שלנו של חשבון בנק לרכיב תוכנה
- תאור הפעולות יתבטא בחוזה ובמתודות המחלקה
- יש להזהר לא להצמד יותר מדי לתאור העולם האמיתי (דוגמא: פקיד בנק שעושה הכל)

## שרותי מחלקה

- ישנם 3 סוגים של מתודות (שרותים, פונקציות):
- פקודות (commands, transformers, mutators)
  - מבצעות שינוי במצב המופשט (abstract state) של העצם
  - כגון: משיכה, הפקדה
  - שאליות (queries, accessors)
  - מחזירות ערך ללא שינוי המצב המופשט
  - כגון: בירור יתרה
  - בנאים (constructors)
  - יצירת עצם חדש
  - כגון: יצירת חשבון חדש

## המצב הפנימי

- המצב הפנימי של עצם מיוצג ע"י נתוניו (שדותיו)
- שדות עצם הם בד"כ עם הרשאת גישה פרטית
- במקרה של חשבון בנק – היתרה
- מאיזה טיפוס?

```
public class BankAccount {  
    ...  
    private double balance;  
}
```

## שאלות BankAccount

- ברור יתרה:
  - ארגומנטים?
  - מה טיפוס הערך המוחזר?
  - תנאי קדם? תנאי בתר?
- פרטים על החשבון:
  - מספר חשבון?
  - פרטים על בעל החשבון?
    - תועדת זהות?
    - גיל?

## חתימה של פקודות

- בד"כ פקודות אינן מחזירות ערך (גם לא ערך שגיאה) וחתימתן היא עם טיפוס ערך מוחזר void
- לפעמים פקודות מחזירות הפנייה לעצם הנוכחי (this)
  - בעד: מאפשר הרכבה של פקודות
  - נגד: מטשטש את ההבחנה בין שאלתה ופקודה

```
x.command1();  
x.command2();  
x.command3(); } x.command1().command2().command3();
```

e.g. new StringBuilder().append("19").append(84).toString();

## setter/getter

- לא כל שדה עם נראות פרטית (private) צריך setter/getter ציבורי
- יצירה 'אוטומטית' של שדות אלו עבור כל שדה פוגמת בעקרון הסתרת המידע
  - ועם זאת, עדיין יש חשיבות לגישה לנתונים דרך מתודות. מדוע?
- למשל: נתבונן בשדה: private double balance;
  - האם דרוש getter? כן, זהו חלק מהמשק 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 מהמשק

## שאלות BankAccount

- בעולם ה"אקדמי" מקובל לגשת לנתון field בעזרת המתודה field()
  - בשפת Java השתרשה המוסכמה כי הגישה לשדה field תעשה בעזרת המתודה getField()
  - שמירה על מוסכמה זו הכרחית בסביבות GUI Builders - JavaBeans
- ```
public class BankAccount {  
    public double getBalance() {  
        return balance;  
    }  
  
    public long getAccountNumber() {  
        return accountNumber;  
    }  
  
    public Customer getOwner() {  
        return owner;  
    }  
  
    private double balance;  
    private long accountNumber;  
    private Customer owner;  
}
```

## פקודת ה-'להפקיד'

```
/**  
 * Makes a deposit to the current account  
 * @pre amount > 0 , "amount is positive"  
 * @post getBalance() == $prev(getBalance()) + amount ,  
 *       "balance updated according to deposit"  
 */  
public void deposit(double amount) {  
    balance += amount;  
}
```

## פקודת ה-'להפקיד'

- המתודה: deposit
- סכום הכסף המופקד מתווסף ליתרה בחשבון
  - ארגומנטים?
  - ערך מוחזר?
  - תנאי קדם?
  - תנאי בתר?

מוסכמה: שמות פקודות הם שמות פועל

## פקודת ה-'למשוך'

```
/**
 * Withdraw amount from the current account
 * @pre amount <= getBalance() , "can't overdraft"
 * @pre 0 < amount , "amount is positive"
 * @post getBalance() == $prev(getBalance()) - amount ,
 *       "balance updated according to withdraw"
 */
public void withdraw(double amount) {
    balance -= amount;
}
```

## פקודת ה-'למשוך'

- המתודה: withdraw
- סכום הכסף המבוקש יורד מיתרת החשבון. אין באפשרותו של הלקוח להיכנס למצב של משיכת יתר ארגומנטים?
- ערך מוחזר?
- תנאי קדם?
- תנאי בתר?
- תנאי קדם לא יבדקו בגוף המתודה – זהו תכנות מתגונן והוא שגוי בכמה היבטים

## דיון – העברה בנקאית

- **אפשרות ב'** – מתודה סטטית (הסבר בהמשך הקורס) שתקבל שני חשבונות בנק ותבצע ביניהם העברה:

```
/**
 * Makes a transfer of amount from one account to the other
 * @pre 0 < amount <= from.getBalance() , "from can't overdraft"
 * @post to.getBalance() == $prev(to.getBalance()) + amount
 * @post from.getBalance() == $prev(from.getBalance()) - amount
 */
public static void transfer(double amount, BankAccount from,
                             BankAccount to) {
    from.withdraw(amount);
    to.deposit(amount);
}
```

## דיון – העברה בנקאית

- נדון במספר חלופות למימוש העברת סכום מחשבון לחשבון
- **אפשרות א'**: העמסת withdraw ו-deposit שיקבלו 2 ארגומנטים: סכום והפנייה לחשבון נוסף. לדוגמא:

```
/**
 * Makes a transfer of amount from other to the current account
 * @pre 0 < amount , "amount is positive"
 * @pre amount <= other.getBalance() , "other can't overdraft"
 * @post getBalance() == $prev(getBalance()) + amount,
 *       "balance updated"
 * @post other.getBalance() == $prev(other.getBalance()) - amount,
 *       "balance of other updated"
 */
public void deposit(double amount, BankAccount other) {
    other.withdraw(amount);
    balance += amount;
}
```

## שמורת BankAccount

```
/**
 * This class represents a bank account
 * @inv getBalance() >= 0,
 *     "can't overdraft"
 * @inv getAccountNumber() > 0 ,
 *     "an account must have an identifier"
 * @inv owner() != null ,
 *     "an account must have owner"
 */
public class BankAccount {
    ...
}
```

## שמורת המחלקה (class invariant)

- צריכה להתקיים "תמיד"
- לפני ואחרי ביצוע כל מתודה ציבורית אחרי הבנאי
- במחלקה חשבון בנק:
- חשבון חייב להיות עם יתרה אי שלילית
- לכל חשבון קיים מספר מזהה במערכת
- לכל חשבון יש בעלים

## בנאי `BankAccount`

```
/**
 * Constructs a new account and sets its owner and
 * identifier
 * @pre id > 0, "account number must be positive"
 * @pre customer != null,
 *      "an account must have an owner"
 * @post getOwner() == customer,
 *      "argument was assigned"
 * @post getAccountNumber() == id,
 *      "argument was assigned"
 */
public BankAccount(Customer customer, long id) {
    accountNumber = id;
    owner = customer;
}
```

תוכנה | בשפת Java  
אוריית דרור ואורד ברזילי

20

## בנאי

- תפקיד הבנאי הוא ליצור עצם חדש ולהביא אותו למצב המקיים את שמורת המחלקה
- בנאי לא אמור לכלול לוגיקה נוספת פרט לכך
- במחלקה `BankAccount`: בנאי ברירת המחדל יוצר עצם שאינו מקיים את השמורה!
- יש דברים שאינם באחריות המחלקה. למשל:
  - מי דואג שמספרי החשבון יהיו תקינים? (למשל שונים זה מזה)
  - מי מנהל את מאגר הלקוחות?
  - הכמסה (encapsulation)

תוכנה | בשפת Java  
אוריית דרור ואורד ברזילי

19

## חזרה מימוש

- תנאי הקדם מיועדים ללקוח ולכן אסור להם להכיל רכיבים שאינם זמינים לו (כגון מתודות או שדות `private`)
- תנאי הבתר ושמורת המחלקה עשויים להכיל טענות "לצורכי פנים" שיומנו ב: `@imp_inv`, `@imp_post` לדוגמא:

```
/**
 * @imp_post $ret == balance ,
 *      "consistency of representation"
 */
public double getBalance() {...}
```

balance הוא שדה `private` ולכן אינו מיועד ללקוחות

תוכנה | בשפת Java  
אוריית דרור ואורד ברזילי

22

## `final`

- מכיוון שחשבון מזוהה חד-חד ערכית עם עצם של `accountNumber` נהפוך שדה זה ל-`final`:

```
final private long accountNumber;
```

- את השדה (`blank final`) יש לאתחל פעם אחת בדיוק, בתוך הבנאי של המחלקה, כפי שאנו אכן עושים
- כעת, מרגע שנוצר עצם, שפת התכנות אוכפת את הצימוד בין העצם והמזוהה שלו

תוכנה | בשפת Java  
אוריית דרור ואורד ברזילי

21

## יצירת תיעוד אוטומטי

עבודה עם javadoc והוספת תגיות חזרה

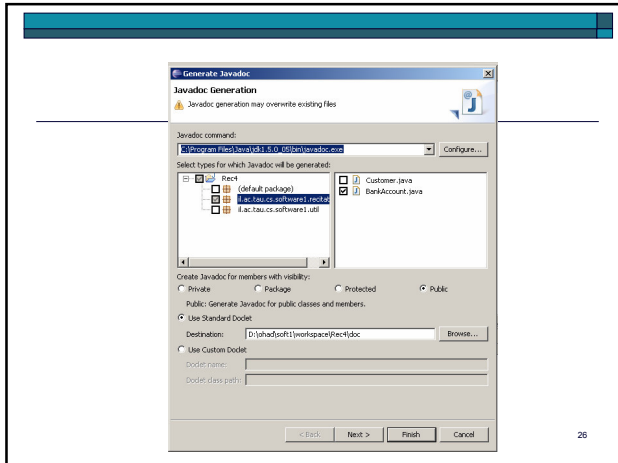
24

## שמורת מימוש של `BankAccount`

```
/**
 * This class represents a bank account
 * @imp_inv getBalance() == balance,
 *      "balance interface is consistent with representation"
 * @imp_inv getOwner() == owner
 *      "owner interface is consistent with representation"
 */
public class BankAccount {
    ...
}
```

תוכנה | בשפת Java  
אוריית דרור ואורד ברזילי

23



## javadoc

כדי לחולל תיעוד אוטומטי עבור הקוד שכתבנו נבחר בסרגל הכלים: Project ->Generate Javadoc...

## javadoc

- סמנו את הקבצים שברצונכם לתעד ב-  public
- בחרו את רמת הגישה אשר ממנה אתם מעוניינים לתעד. לדוגמא:
  - public**: רק מתודות ושדות ציבוריים יופיעו בתיעוד. תיעוד זה מיועד ללקוחות של המחלקה
  - private**: כל המתודות והשדות יכללו בתיעוד. תיעוד זה מיועד למפתחי המחלקה
  - לצורך הגשת התרגילים בקורס זה יש לבחור ב private
- האפשרות Use Standard Doclet תיצור תיעוד אוטומטי HTML במיקום שצויין בשורת ה Destination

תוכנה | בשפת Java  
ארנית דרור ואורד | ברזילי

## javadoc

- נזין (בעזרת הכפתור configure או בשורת המלל) את מיקומה של התוכנית javadoc. תוכנית זו כלולה בחבילת ה JDK (Java SDK) שהורדנו מאתר חברת Sun
- מיקום טיפוס של החבילה הוא ב: C:\Program Files\Java\jdk1.5.x\_xxx\bin\javadoc.exe (ה- xים מסמנים את מספר הגרסה)

תוכנה | בשפת Java  
ארנית דרור ואורד | ברזילי

## הכללת החוזה בתיעוד

- מחולל התיעוד התקני אינו 'מכיר' את התגיות: @pre, @post, @inv, @imp\_inv, @imp\_post
- ניתן להוסיף תגיות אלו על ידי רישום שלהן בתוכנה javadoc באופן הבא:
  - הורידו את הקובץ taglets.jar מאתר הקורס ומקמו אותו בתיקייה לבחירתכם
  - למשל: D:\ohad\soft\taglets.jar
  - בתפריט Generate Javadoc... לחצו פעמיים על Next

תוכנה | בשפת Java  
ארנית דרור ואורד | ברזילי

## javadoc

- לחיצה על Finish תיצור את התיעוד המבוקש
- תיקייה הכוללת את דפי התיעוד תופיע בסייר החבילות בסביבת העבודה
- לחיצה על דפי ה html תפתח אותם בסביבת העבודה

תוכנה | בשפת Java  
ארנית דרור ואורד | ברזילי

## הכללת החוזה בתיעוד

□ הזינו בחלון Extra Javadoc options את הפרטים הבאים:

```
-taglet il.ac.tau.cs.software1.util.PreTaglet
-taglet il.ac.tau.cs.software1.util.PostTaglet
-taglet il.ac.tau.cs.software1.util.InvTaglet
-taglet il.ac.tau.cs.software1.util.ImpPostTaglet
-taglet il.ac.tau.cs.software1.util.ImpInvTaglet

-tagletpath D:\ohad\soft1\taglets.jar
```

□ עדכנו את מיקום הקובץ ה taglets.jar לפי מיקומו במחשב שלכם

□ לסיים ליחצו על Finish

