

תוכנה 1

תרגול מס' 6
מחלקות, עצמים, וקצת חוזים

תזכורת – מופעי מחלקה

- אפשר ליצור מופעים של מחלקה מסוימת (גם: עצמים מטיפוס המחלקה) בעזרת ביטוי `new`.

```
BankAccount account1 = new BankAccount(...);
```

- כל מופע יכול להכיל ערכים שונים של **שדות מופע**.
- בניגוד לשדות סטטיים, אשר שייכים למחלקה

- כל מופע יכול לקרוא ל**שירותי מופע**

- מתוך שירותים אלה יש גישה למשתנה `this`, אשר מצביע על העצם הקורא, וממנו ניתן לגשת לשדות ושירותי מופע נוספים
- בניגוד לשירותים (פונקציות\מתודות) סטטיים, אשר אינם מקושרים למופע ספציפי אלא רק למחלקה

המצב הפנימי של אובייקט

- המצב הפנימי של עצם מיוצג ע"י נתוניו (שדותיו)
- שדות עצם יהיו לרוב עם הרשאת גישה פרטית
- במקרה של חשבון בנק:
- מצב פנימי: מכיל בין היתר שדה לייצוג היתרה
- מאיזה טיפוס?

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



שירותי מופע

ישנם 3 סוגי שירותים (מתודות, פונקציות, פרוצדורות):

- שאילתות (queries, accessors)

- מחזירות ערך ללא שינוי המצב הפנימי

- שאילתות צופות (**observers**): מחזירות פרט מידע הקשור לעצם (למשל, בירור יתרה)

- שאילתות מפיקות (**producers**): מחזירות עצם מאותו טיפוס (למשל, חשבון חיסכון

המקושר לחשבון עובר ושב)

- בד"כ שימושיות עבור עצמים **מקובעים** (immutable) כמו מחרוזות.

- פקודות (commands, transformers, mutators)

- מבצעות שינוי במצב הפנימי של העצם

- כגון: משיכה, הפקדה

- בנאים (constructors)

- יצירת עצם חדש

- כגון: יצירת חשבון חדש

תזכורת - חוזה בין ספק ללקוח

- חוזה בין ספק ללקוח מגדיר עבור כל שרות:
 - תנאי ללקוח - "תנאי קדם" - precondition
 - תנאי לספק - "תנאי אחר" - postcondition.



תנאי קדם (preconditions)

- מגדירים את הנחות הספק - מצבים של התוכנית שבהם מותר לקרוא לשירות
- בד"כ, ההנחות הללו נוגעות רק לקלט שמועבר לשירות.
- תנאי הקדם יכול להיות מורכב ממספר תנאים שעל כולם להתקיים (AND)
- סימון:

@pre

תנאי אחר (postconditions)

- אם תנאי הקדם מתקיים, הספק חייב לקיים את תנאי האחר
- ואם תנאי קדם אינו מתקיים? לא ניתן להניח דבר:
 - אולי השרות יסתיים ללא בעיה
 - אולי השרות יתקע בלולאה אינסופית
 - אולי התוכנית תעוף מייד
 - אולי יוחזר ערך שגוי
- אולי השרות יסתיים ללא בעיה אך התוכנית תעוף / תתקע לאחר מכן ...
- ובכתיב לוגי: תנאי קדם \Leftarrow תנאי אחר,
(תנאי קדם) \Leftarrow !?
- סימון: @post

כיצד נסמן?

- בקורס הנוכחי אנחנו מאפשרים גמישות בתחביר של כתיבת חוזים
- ניתן להשתמש ב:
 - תנאים בוליאניים בג'אווה ($x \geq 0$)
 - תגיות מהסגנון (שנלמד בהרצאה): `@pre`, `@post`, `$prev`, `$ret`, `$implies`
 - ביטויים ונוסחאות מתמטיים ($x \in [0,1]$)
 - שפה חופשית ("`M is a diagonal square matrix`")
 - שילובים של הנ"ל, ועוד
- בכתיבת חוזים חשוב לשמור על
 - התייחסות לכל המקרים שמתאימים לתנאי הקדם בתנאי האחר
 - תמציתיות, בהירות ודיוק! (בייחוד אם משתמשים בשפה טבעית)

שאלות BankAccount

```

public class BankAccount {
    public double getBalance() {
        ...;
    }

    public long getAccountNumber() {
        ...;
    }

    public Customer getOwner () {
        ...;
    }

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

```

שאלות

מצב
פנימי

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

getter/setter

- יש חשיבות לגישה לנתונים דרך מתודות. מדוע?
- לא כל שדה עם נראות פרטית (**private**) צריך **getter/setter** ציבורי

למשל: עבור השדה **balance**

- האם דרוש **?getter**
- כן, זהו חלק מהממשק של חשבון בנק
- האם דרוש **?setter**


```
public void setBalance(double balance) {
    this.balance = balance;
}
```

לא בהכרח, פעולות של משיכה או הפקדה אמנם משפיעות על היתרה, אבל פעולה של שינוי יתרה במנותק מהן אינה חלק מהממשק

פקודות: משיכה והפקדה

```
public void deposit(double amount) {
    balance += amount;
}

/**
 * Withdraw amount from the account
 *
 * @pre 0 < amount <= getBalance()
 * @post getBalance() == $prev(getBalance()) - amount
 */
public void withdraw(double amount) {
    if (amount < 0 || amount > getBalance()) {
        System.out.println("Invalid withdrawal amount: "+
            amount);
        return;
    }
    balance -= amount;
}
```



אפשרות ב': תכנות מתגובני:

דיון – העברה בנקאית

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

```
/**
 * Makes a transfer of amount from one account to the other
 * @pre 0 < amount <= from.getBalance()
 * @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

- אפשרות ב: אחד החשבונות אחראי לפעולה (למשל, מעביר הכסף)

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

אפשר גם להשתמש בהעמסה של withdraw

בנאי

- תפקיד: ליצור עצם חדש ולא תחיל את שדותיו
- בנאי לא אמור לכלול לוגיקה נוספת פרט לכך!
- לאחר האתחול העצם חייב לקיים את **משתמר המחלקה**
- דוגמא למשתמר: מאזן אי-שלילי, בעלים אינו null, מס' חשבון חיובי...
- במחלקה `BankAccount`:
 - בנאי **ברירת המחדל** יוצר עצם שאינו מקיים את המשתמר!
 - נותן ערכי ברירת מחדל לכל השדות, ולכן, למשל, הבעלים הוא null.
- 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  
 * @pre id > 0  
 * @pre customer != null  
 * @post getOwner() == customer  
 * @post getAccountNumber() == id  
 * @post getBalance() == 0  
 */  
public BankAccount(Customer customer, long id) {  
    accountNumber = id;  
    owner = customer;  
}
```

אין ערך החזרה לבנאי!
לא נקרא ל- **new** BankAccount מהבנאי.
אם יש בעיה בקלט אי אפשר להחזיר **null**

העמסת בנאים

```

/**
 * Constructs a new account and sets its owner and identifier
 * @pre id > 0
 * @pre customer != null
 * @pre initialBalance >= 0
 * @post getOwner() == customer
 * @post getAccountNumber() == id
 * @post getBalance() == initialBalance
 */
public BankAccount(Customer customer, long id,
                    double initialBalance) {
    this(customer, id);
    balance = initialBalance;
}

```

תזכורת: העמסה = יצירת מתודה בעלת שם זהה אך עם ארגומנטים שונים. באופן דומה ניתן להגדיר בנאים עם ארגומנטים שונים.

`this()` כאן משמש לא כמשתנה אלא כקריאה לבנאי אחר של אותה מחלקה שיבצע אתחול ראשוני על העצם שאנו מייצרים. ניתן להשתמש בתחביר זה רק מתוך בנאי!

עצמים, מחלקות, נראות ומה שביניהם

המחלקה CurrentClass

```
public class CurrentClass {  
  
    public static void myPublicStaticMethod() {  
        System.out.println("In myPublicStaticMethod");  
    }  
  
    private static void myPrivateStaticMethod() {  
        System.out.println("In myPrivateStaticMethod");  
    }  
  
    public void myPublicMethod() {  
        System.out.print("In myPublicMethod >> ");  
        myPrivateMethod();  
    }  
  
    private void myPrivateMethod() {  
        System.out.println("In myPrivateMethod");  
    }  
}
```

קריאה למתודה פרטית ממתודה פומבית
(גם ההפך זה בסדר)
שקול ל- `this.myPrivateMethod()`

המחלקה OtherClass

```
public class OtherClass {  
    public static void othersPublicStaticMethod() {  
        System.out.println("In othersPublicStaticMethod");  
    }  
  
    private static void othersPrivateStaticMethod() {  
        System.out.println("In othersPrivateStaticMethod");  
    }  
  
    public void othersPublicMethod() {  
        System.out.print("In othersPublicMethod >> ");  
        othersPrivateMethod();  
    }  
  
    private void othersPrivateMethod() {  
        System.out.println("In othersPrivateMethod");  
    }  
}
```

נוסף main ל-CurrentClass

```

public class CurrentClass {
    public static void main(String[] args) {
        CurrentClass.myPublicStaticMethod(); // Prints: In myPublicStaticMethod
        myPublicStaticMethod();             // Prints: In myPublicStaticMethod
        CurrentClass.myPrivateStaticMethod(); // Prints: In myPrivateStaticMethod
        ❌ CurrentClass.myPublicMethod();

        CurrentClass currentClass = new CurrentClass();
        currentClass.myPublicMethod(); // Prints: In myPublicMethod >> In myPrivateMethod
        currentClass.myPrivateMethod(); // Prints: In myPrivateMethod
        currentClass.myPublicStaticMethod(); //Has a warning, Prints: In myPublicStaticMethod

        OtherClass.othersPublicStaticMethod(); // Prints: In othersPublicStaticMethod
        ❌ othersPublicStaticMethod();
        ❌ OtherClass.othersPrivateStaticMethod();

        OtherClass otherClass = new OtherClass();
        otherClass.othersPublicMethod();// Prints: In othersPublicMethod >> In othersPrivateMethod
        ❌ otherClass.othersPrivateMethod();
    }
    ...
}

```

מסקנות

- מתודה סטטית אינה יכולה לקרוא למתודה שאינה סטטית
 - חייבים לציין מיהו העצם שהשירות משויך אליו
 - `myPublicMethod()` לא יעבוד (מתוך מתודה סטטית)
 - `currentClass.myPublicMethod()` כן!
- נראות מגדירה מאיזה **מקום בקוד** ניתן לגשת למתודה
 - נראות פרטית = ניתן לגשת רק מהקוד של אותה מחלקה
 - נראות פומבית = ניתן לגשת מכל מחלקה (אם היא לא באותה חבילה, יש להוסיף הצהרת `import`)
 - נלמד על עוד שני סוגים בהמשך

Instance vs. Class (static) Fields

Instance fields

■ למה?

■ ייצוג פנימי של המופע

■ מתי?

■ מאותחלים עם יצירת האובייקט

■ כמה?

■ אחד לכל מופע

■ מאיפה?

■ נגישים אך ורק ממתודות מופע!
(למה?)

Class (static) fields

■ למה?

■ קבועים

■ ערכים המשותפים לכל מופעי המחלקה

■ מתי?

■ מאותחלים לפי הסדר עם טעינת המחלקה

■ כמה?


■ יש רק 1 בכל התוכנית! (0 לפני טעינת המחלקה)

■ מאיפה?

■ נגישים ממתודות סטטיות ומתודות מופע

```
public class BankAccount {
    public static final String BANK_NAME = "BNP"; //static constant
    private static int lastAccountId = 0; //static field
    private int id;

    public BankAccount() {
        id = ++lastAccountId; // unique ID for every account
    }

    /* static method */
    public static void main(String[] args) {
        System.out.println(lastAccountId);
         System.out.println(id);
        BankAccount account = new BankAccount();
        System.out.println(account.id);
    }

    /* instance method */
    public void printStuff() {
        System.out.println(lastAccountId);
        System.out.println(id);
    }
}
```