תוכנה 1

תרגול 4: מחלקות ועצמים

תכנון תוכנה למערכת בנקאית

תכנון מערכת תוכנה עוסק במיפוי בין עולם הבעיה ועולם הפתרון

עולם הבעיה:
- בנקים
- לקוחות
- משיכות, הפקדות
- חשבונות
- יתרות

עולם הפתרון:
- שפת תכנות
- עצמים
- מחלקות
- שירותים
- שדות

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 ובמתודות המחלקה

המצב הפנימי

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

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


המצב הפנימי

- המצב הפנימי של עצם מיוצג ע"י נתוניו (שדותיו)
- שדות עצם הם 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;
}
```


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)
  - יצירת עצם חדש
  - כגון: יצירת חשבון חדש

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 שאילתה ופקודה

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

e.g.

```
new StringBuilder().append("19").append(84).toString();
```


שאילתות BankAccount

- ברור יתרה:
  - ארגומנטים?
  - מה טיפוס הערך המוחזר?
  - חוזה? (תנאי קדם? תנאי אחר?)
- פרטים על החשבון:
  - מספר חשבון?
  - פרטים על בעל החשבון?
    - תעודת זהות?
    - גיל?

שאילתות BankAccount

```
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;
}
```

שאילתות

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 ציבורי **getter/setter**
- 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?
  - חוזה? (תנאי קדם?, תנאי אחר?)

מוסכמה: שמות פקודות הם שמות פועל

פקודת ה-'להפקיד'

```
/**
 * Makes a deposit to the account
 * @pre  ??????????????????????????????????????
 * @post ??????????????????????????????????????
 */
public void deposit(double amount) {
  balance += amount;
}
```

פקודת ה-'להפקיד'

```
/**
 * Makes a deposit to the account
 * @pre   amount > 0
 * @post ??????????????????????????????????????
 */
public void deposit(double amount) {
  balance += amount;
}
```

בגישת **תכנות לפי חוזה** תנאיי הקדם לא יבדקו בגוף המתודה. אחרת: **תכנות מתגונן**.

פקודת ה-'להפקיד'

```
/**
 * Makes a deposit to the account
 * @pre   amount > 0
 * @post getBalance() == $prev(getBalance()) + 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?
  - חוזה? (תנאי קדם? תנאי אחר?)

פקודת ה-'למשוך'

```
/**
 * Withdraw amount from the account
 * @pre  ??????????????????????????????????????
 * @post ??????????????????????????????????????
 */
public void withdraw(double amount) {
  balance -= amount;
}
```


פקודת ה-'למשוך'

```
/**
 * Withdraw amount from the account
 * @pre   0 < amount <= getBalance()
 * @post ??????????????????????????????????????
 */
public void withdraw(double amount) {
  balance -= amount;
}
```


פקודת ה-'למשוך'

```
/**
 * Withdraw amount from the account
 * @pre   0 < amount <= getBalance()
 * @post getBalance() == $prev(getBalance()) - amount
 */
public void withdraw(double amount) {
  balance -= amount;
}
```

דיון – העברה בנקאית

מספר חלופות למימוש העברת סכום מחשבון לחשבון:

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

```
/**
 * Makes a transfer of amount from other to the current account
 * @pre 0 < amount <= other.getBalance()
 * @post getBalance() == $prev(getBalance()) + amount
 * @post other.getBalance() == $prev(other.getBalance()) – amount
 */
public void deposit(double amount, BankAccount other) {
    other.withdraw(amount);
    balance += amount;
}
```


דיון – העברה בנקאית

- **אפשרות ב**: 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

ניתן לתת למתודות שמות מפורשים יותר, כגון:

```
/**
 * 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;
}
```


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

שמורת BankAccount

```
/**
 * @inv getBalance() >= 0
 * @inv getAccountNumber() > 0
 * @inv getOwner() != null
 */
public class BankAccount {
  ...
}
```


בנאי

- תפקיד: ליצור עצם חדש 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 לתקינות מספרי חשבון? (למשל שיהיו שונים)
  - מי מנהל את מאגר הלקוחות?
  - הכמסה (encapsulation)

BankAccount בנאי

```
/**
 * Constructs a new account and sets its owner and identifier
 * @pre id > 0
 * @pre customer != null
 * @post ????????
 * @post ????????
 * @post ????????
 */
public BankAccount(Customer customer, long id) {
   accountNumber = id;
   owner = customer;
}
```


BankAccount בנאי

```
/**
 * Constructs a new account and sets its owner and identifier
 * @pre  ????????
 * @pre  ????????
 * @post ????????
 * @post ????????
 * @post ????????
 */
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

BankAccount בנאי

```
/**
 * 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;
}
```


BankAccount שמורת מימוש של

```
/**
 * 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 {
   ...
}
```


חוזה מימוש

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

לדוגמא:

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

balance הוא שדה private ולכן אינו מיועד ללקוחות

שיעור הבא

- נשלים את דוגמת הבנק
- נבדוק את הקשר בין המחלקות השונות במערכת
  - Bank
  - Customer
  - BankAccount
- נראה דוגמה לשימוש במערכת