

חוזה בין ספק ללקוח

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



2

תוכנה 1

תרגול 4: חוזים
רובי בוים ומתי שמרת

1

תנאי אחר (postconditions)

- מגדיר את המחויבות של הספק
- 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 יוחזר ערך שגוי
 - אולי השרות יתקע בלולאה אינסופית
 - אולי התוכנית תעוף מייד, אולי השרות יסתיים ללא בעיה אך והתוכנית תעוף / תתקע לאחר מכן

... 4

תנאי קדם (preconditions)

- מגדירים את הנחות הספק
- הנחות הללו מתארות מצבים של התוכנית שבהם מותר לקרוא לשירות
- במקרים פשוטים (ונפוצים), ההנחות הללו נוגעות רק לקלט שמועבר לשירות
- במקרה הכללי ההנחות הללו מתייחסות גם למצב התוכנית, כגון משתנים גלובליים
- תנאי הקדם יכול להיות מורכב ממספר תנאים שעל כולם להתקיים (AND)

3

דוגמא 2 (אותו קוד, חוזה שונה)

```
/*
 * precondition: arr != null
 *
 * postcondition:
 *   If ((arr.length==0) || (arr contains only NaNs))
 *     returns Infinity.
 *   Otherwise, returns the minimal value in arr.
 */
public static double min2(double[] arr) {
    double m = Double.POSITIVE_INFINITY;
    for (double x : arr)
        m = (x < m ? x : m);
    return m;
}
```

בהשווא לחוזה מדוגמא 1:
חוזה מתירני יותר מבחינת הלקוח

6

דוגמא 1

```
/*
 * precondition:
 *   1) arr != null
 *   2) arr.length > 0
 *   3) arr contains only numbers (no NaN or ±infinity)
 *
 * postcondition: Returns the minimal element in arr
 */
public static double min1(double[] arr) {
    double m = Double.POSITIVE_INFINITY;
    for (double x : arr)
        m = (x < m ? x : m);
    return m;
}
```

המימוש אינו בודק את קיומם של תנאי הקדם

מה יקרה אם בקריאה ל- min1 לא יקויימו כל התנאים בתנאי הקדם?
?arr==null
?arr.length==0
?arr מכיל NaN
?arr מכיל Infinity או -Infinity

1

דוגמא 4 (ללא precondition)

```

/*
 * precondition: true
 *
 * postcondition: If ((arr==null) || (arr.length==0))
                 returns NaN
 * Otherwise, if arr contains only NaN - returns Infinity.
 * Otherwise, returns the minimal value in arr, ignoring any NaN.
 */
public static double min4(double[] arr) {
    if (arr == null || arr.length == 0)
        return Double.NaN;

    double m = Double.POSITIVE_INFINITY;
    for (double x : arr)
        m = (x < m ? x : m);

    return m;
}

```

מוכן לכל מקרה

תנאי אחר המגדיר תגובה לכל קלט אפשרי מסבך את הקוד.

8

דוגמא 3 (טיפול שונה ב-NaN)

```

/*
 * precondition: arr != null
 *
 * postcondition: If (arr.length==0) returns Infinity.
 * Otherwise, if arr contains NaN - returns NaN.
 * Otherwise, returns the minimal value in arr.
 */
public static double min3(double[] arr) {
    double m = Double.POSITIVE_INFINITY;

    for (double x : arr) {
        if (Double.isNaN(x))
            return x;
        m = (x < m ? x : m);
    }

    return m;
}

```

השוואה לחזרה מדוגמא 2: טיפול שונה במקרה קצה (קיום ערכי NaN)

7

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)
- ניתכן 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

10

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
- חשבוניות
- יתרות



9

שרותי המחלקה

- ישנם 3 סוגי שירותים (מתודות, פונקציות, פרוצדורות)
 - שאליות (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

12

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

11

שאליות BankAccount

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

השאליות הן מצג לעולם החיצון.

שאליות
מצב
פנימי

שאליות BankAccount



- ברור יתרה:
 - ארגומנטים?
 - מה טיפוס הערך המוחזר?
- פרטים על החשבון:
 - מספר חשבון?
 - פרטים על בעל החשבון?
 - תעודת זהות?
 - גיל?

המצב הפנימי

- המצב הפנימי של עצם מיוצג ע"י נתונים (שדותיו)
 - מאפשר מימוש שאליות/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 accountNumber;
    private Customer owner;
}
```

שאליות BankAccount

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

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

שאליות
מצב
פנימי

פקודת ה-'להפקיד'

- המתודה: deposit
- סכום הכסף המופקד מתווסף ליתרה בחשבון
 - ארגומנטים?
 - ערך מוחזר?
 - מהן הנחות המימוש?

מוסכמה: שמות פקודות הם שמות פועל



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'

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

21

פקודת ה-'להפקיד'

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

20

פקודת ה-'להפקיד'

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

23

פקודת ה-'להפקיד'

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

בגישת תכנות לפי חזרה תנאי הקדם לא יבדוק בגוף המתודה. אחרת: תכנות מתגונן.

22

פקודת ה-'למשוך'

■ המתודה: withdraw

■ סכום הכסף המבוקש יורד מיתרת החשבון.

■ משיכת יתר (אוברדרפט) אינו אפשרי.

■ ארגומנטים?

■ ערך מוחזר?

■ מהן הנחות המימוש?

25

מימוש ה-'להפקיד'

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

24

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;
}
```

33

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 < amount <= 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);
}
```

32

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

35

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;
}
```

34

בנאי BankAccount

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

37

שמורת BankAccount

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

36

BankAccount 'אנן

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

39

BankAccount 'אנן

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

38