

תוכנה 1

תרגול 7: מנשקים, פולימורפיזם ועוד

== vs equals

Point p1 = new Point(1,2)

Point p2 = new Point(1,2)

p1 == p2

p1.equals(p2)

■ מתי נכון להשתמש בכל אחד מהם ?

■ שימו לב, במחלקה שכתבתם בעצמכם יש לכתוב מתודת equals על מנת להשתמש בה.

■ אין להשתמש במתודת "ברירת מחדל" (יוסבר בהמשך הקורס)

נדפד

מנשקים

- מנשק (interface) הוא מבנה תחבירי ב Java המאפשר לחסוך בקוד לקוח
- קוד אשר משתמש במנשק יוכל בזמן ריצה לעבוד עם מגוון מחלקות המממשות את המנשק הזה (ללא צורך בשכפול הקוד עבור כל מחלקה)
- דוגמא: נגן מוזיקה אשר מותאם לעבוד עם קובצי מוזיקה (mp3) ועם קובצי וידאו (mp4)

הגדרת ממשק

```
public interface InterfaceName {  
    public String getString();  
    public void method(int param);  
}
```

שם הממשק

מחלקה המממשת את
הממשק

```
public class Concrete implements InterfaceName {  
    ...  
    @Override  
    public String getString() {...}  
    @Override  
    public void method(int param) {...}  
}
```

Playing Mp3

```
public class MP3Song {  
  
    public void play(){  
        // audio codec calculations,  
        // play the song...  
    }  
  
    // does complicated stuff  
    // related to MP3 format...  
}
```

```
public class Player {  
  
    private boolean repeat;  
    private boolean shuffle;  
  
    public void playSongs(MP3Song[] songs) {  
        do {  
            if (shuffle)  
                Collections.shuffle(Arrays.asList(songs));  
  
            for (MP3Song song : songs)  
                song.play();  
  
        } while (repeat);  
    }  
}
```

Playing VideoClips

```
public class VideoClip {  
  
    public void play(){  
        // video codec calculations,  
        // play the clip ...  
    }  
  
    // does complicated stuff  
    // related to MP4 format ...  
}
```

```
public class Player {  
  
    // same as before...  
  
    public void playVideos(VideoClip[] clips) {  
        do {  
            if (shuffle)  
                Collections.shuffle(Arrays.asList(clips));  
  
            for (VideoClip videoClip : clips)  
                videoClip.play();  
  
        } while (repeat);  
    }  
}
```

שכפול קוד

```
public void playSongs(MP3Song[] songs) {  
    do {  
        if (shuffle)  
            Collections.shuffle(Arrays.asList(songs));  
  
        for (MP3Song song : songs)  
            song.play();  
  
    } while (repeat);  
}
```

למרות ששני השרותים נקראים `play()`
אלו פונקציות שונות!

```
public void playVideos(VideoClip[] clips) {  
    do {  
        if (shuffle)  
            Collections.shuffle(Arrays.asList(clips));  
  
        for (VideoClip videoClip : clips)  
            videoClip.play();  
  
    } while (repeat);  
}
```

נרצה למזג את שני קטעי הקוד

שימוש במנשק

```
public void play (Playable[] items) {  
    do {  
        if (shuffle)  
            Collections.shuffle(Arrays.asList(items));  
  
        for (Playable item : items)  
            item.play();  
  
    } while (repeat);  
}
```

```
public interface Playable {  
    public void play();  
}
```

מימוש המנשק ע"י הספקים

```
public class VideoClip implements Playable {  
  
    @Override  
    public void play() {  
        // render video, play the clip on screen...  
    }  
  
    // does complicated stuff related to video formats...  
}
```

```
public class MP3Song implements Playable {  
  
    @Override  
    public void play(){  
        // audio codec calculations, play the song...  
    }  
  
    // does complicated stuff related to MP3 format...  
}
```

מערכים פולימורפים

```
Playable[] playables = new Playable[3];
```

```
playables[0] = new MP3Song();
```

```
playables[1] = new VideoClip();
```

```
playables[2] = new MP4Song(); // new Playable class
```

```
Player player = new Player();
```

```
// init player...
```

```
player.play(playables);
```

```
public void play (Playable [] items) {  
    do {  
        if (shuffle)  
            Collections.shuffle(Arrays.asList(items));  
  
        for (Playable item : items)  
            item.play();  
  
    } while (repeat);  
}
```

עבור כל איבר במערך
יקרא ה `play()` המתאים

דלדלל: אפרש השפה CALC

השפה CALC

■ ניזכר בשפה CALC משעורי הבית:

■ שפה מבוססת מחסנית

■ בשורה ראשונה: הצהרה על מספר הפרמטרים

■ מספרים: מתווספים לראש המחסנית

■ פקודות: פועלות על המחסנית

■ פרמטרים מוסרים מראש המחסנית

■ ערכי חזרה מתווספים לראש המחסנית

■ דוגמא: פעולת החיבור מסירה שני ערכים מראש

המחסנית ומוסיפה את סכומם לראש המחסנית

args 0
1
2
+

מפרש השפה CALC

- המפרש קורא תוכנית CALC מזרם הקלט:
 - שובר את זרם הקלט לשורות
 - פועל על כל שורה באמצעות `runCommand`
- מה הבעיה עם המתודה `runCommand` של המפרש?
 - נתבונן במימוש אפשרי

runCommand

```
public static double runCommand(String cmd) {  
    switch (cmd) {  
        case "+":  
            push(pop() + pop());  
        break;  
        case "-":  
            push(pop() - pop());  
        break;  
        case "*":  
            push(pop() * pop());  
        break;  
        ...  
        default:  
            push(Double.parseDouble(cmd));  
    }  
  
    return top();  
}
```

לכל Command יש
מקרה משלה ב-
.switch

קשה לתחזק את הקוד.
תוצר מתודה ארוכה
שכל המתכנתים
מעדכנים.

אם לא ראינו פקודה
מוכרת, אנו מצפים
למספר שיש להכניס
למחסנית.

runCommand

```
public static double runCommand(String cmd) {  
    switch (cmd) {  
        case "+":  
            push(pop() + pop());  
            break;  
        case "-":  
            push(pop() - pop());  
            break;  
        case "*":  
            push(pop() * pop());  
            break;  
        ...  
        default:  
            push(Double.parseDouble(cmd));  
    }  
  
    return top();  
}
```

תבנית חוזרת

לכל Command יש שם.

כל Command משנה את
המחסנית.

מנשק Command

```
public interface Command {  
    public String getName();  
    public void interpret(Stack stack);  
}
```

לכל Command יש שם.

```
public interface Stack {  
    public double pop();  
    public double top();  
    public void push(double value);  
    public int size();  
}
```

כל Command משנה את המחסנית.

בצורה דומה נגדיר גם מנשק למחסנית

מימוש פקודות

■ כעת נוכל לממש כל פקודה במחלקה משלה

■ יתרונות

■ המחלקות פשוטות

■ ניתן לשמור מידע נוסף וכן פונקציות עזר נוספות

■ ניתן לפתח מספר פקודות במקביל (מספר מפתחים)

■ חסרונות

■ יש הרבה מחלקות והקוד מפוזר על פני מספר קבצים

דוגמא: חיבור

```
public class PlusCommand implements Command {  
    @Override  
    public String getName() {  
        return "+";  
    }  
  
    @Override  
    public void interpret(Stack stack) {  
        stack.push(stack.pop() + stack.pop());  
    }  
}
```

מימוש מנשקים

ציון מפורש שהמתודה היא מימוש למנשק

החתימה חייבת להיות זהה לזו המצוינת במנשק

דוגמא: חיבור

```
public class PlusCommand implements Command {  
    @Override  
    public String getName() {  
        return "+";  
    }  
  
    @Override  
    public void interpret(Stack stack) {  
        stack.push(stack.pop() + stack.pop());  
    }  
}
```

המחרוזת המתאימה לחיבור

המימוש נשאר זהה

פולימורפיזם

■ כיצד בונים את קוד הלקוח (runCommand)?

■ נחזיק רשימה של פקודות אפשריות:

```
private static Command[] commands = {  
    new PlusCommand(),  
    new MinusCommand(),  
    new MultiplyCommand(), ... };
```

■ ב-runCommand:

■ נעבור על הרשימה ונחפש התאמה למחרוזת הקלט

■ אם מצאנו התאמה, נבצע את הפקודה.

runCommand החדש

```
public static double runCommand(String cmd) {  
    if (cmd == null || cmd.isEmpty() || cmd.startsWith(";"))  
        return top();  
  
    for (Command command : commands)  
        if (command.getName().equals(cmd)) {  
            command.interpret(stack);  
            return top();  
        }  
  
    push(Double.parseDouble(cmd));  
    return top();  
}
```

פולימורפיזם:
בזמן קומפילציה לא ידוע היכן ממומשת
המתודות interpret ו- getName.
רק בזמן ריצה יקבעו המתודות הנכונות,
עפ"י הטיפוס הקונקרטי של העצם!

פקודות לא טריוויאליות

- נדגים כיצד להרחיב את מודל החישוב ע"י הוספת משתנים (או זיכרון) לתוכניות CALC.
- נדגים פקודות הנעזרות בנתונים נוספים שאינם במחסנית
- למה אנו זקוקים?
- שטח אחסון (זיכרון)
- פקודה לשמירה אל הזיכרון
- הגדרה: על המחסנית ישמרו הכתובת והאיבר לשמירה
- פקודה לטעינה מהזיכרון
- הגדרה: על המחסנית יש כתובת, יוחזר הערך המבוקש

שמירה לזיכרון

```
public class MemoryStoreCommand implements Command {
    private double[] memory;

    MemoryStoreCommand(double[] memory) {
        this.memory = memory;
    }

    @Override
    public void interpret(Stack stack) {
        int address = (int)stack.pop();
        memory[address] = stack.pop();
    }

    @Override
    public String getName() {
        return "store";
    }
}
```

שטח הזיכרון והקצאתו אינן
באחריות המחלקה

מקבלים כתובת (מספר
שלם) וכן ערך מהמחסנית.

טעינה מהזיכרון

```
public class MemoryLoadCommand implements Command {
    private double[] memory;

    MemoryLoadCommand(double[] memory) {
        this.memory = memory;
    }

    @Override
    public void interpret(Stack stack) {
        int address = (int)stack.pop();
        stack.push(memory[address]);
    }

    @Override
    public String getName() {
        return "load";
    }
}
```

מקבלים כתובת (מספר שלם) ומחזירים ערך על גבי המחסנית.

אתחול המפרש

הוספת פקודות הגישה לזיכרון במפרש דורשת גם הקצאת שטח זיכרון

```
private static double[] memory = new double[MEMORY_SIZE];  
  
private static Command[] commands = {  
    ...  
    new MemoryStoreCommand(memory),  
    new MemoryLoadCommand(memory)  
};
```

שימושים

■ מה ניתן לעשות עם זיכרון?

■ מספר דוגמאות

■ חישוב שורשים של משוואה ריבועית

■ חישוב סדרת Fibonacci (עפ"י נוסחא סגורה)

■ חישוב טור גיאומטרי

■ (קוד CALC עבור הדוגמאות הנ"ל זמין באתר הקורס)

דוגמא: טור גיאומטרי

- ; input parameters: r n a
- ; **1. store parameters in memory**
- ; 2. place (1-r) on stack
- ; 3. place(1-r^n) on stack
- ; 4. divide
- ; 5. multiply by a

$$\sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r} :$$

דוגמא: טור גיאומטרי

```
; input parameters: r n a
; 1. store parameters in memory
3
store
2
store
1
Store
; 2. place (1-r) on stack
; 3. place(1-r^n) on stack
; 4. divide
; 5. multiply by a
```

$$\sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r} :$$

דוגמא: טור גיאומטרי

```
; input parameters: r n a
; 1. store parameters in memory
3
store
2
store
1
Store
; 2. place (1-r) on stack
; 3. place(1-r^n) on stack
; 4. divide
; 5. multiply by a
```

$$\sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r} :$$

דוגמא: טור גיאומטרי

```
; input parameters: r n a
; 1. store parameters in memory
3
store
2
store
1
Store
; 2. place (1-r) on stack
1
load
1
-
; 3. place(1-r^n) on stack
; 4. divide
; 5. multiply by a
```

$$\sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r} :$$

דוגמא: טור גיאומטרי

```
; input parameters: r n a
; 1. store parameters in memory
3
store
2
store
1
Store
; 2. place (1-r) on stack
1
load
1
-
; 3. place(1-r^n) on stack
; 4. divide
; 5. multiply by a
```

$$\sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r} :$$

דוגמא: טור גיאומטרי

```
; input parameters: r n a
; 1. store parameters in memory
3
store
2
store
1
Store
; 2. place (1-r) on stack
1
load
1
-
; 3. place(1-r^n) on stack
2
load
```

$$\sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r} :$$

```
1
load
**
1
-
; 4. divide
; 5. multiply by a
```

דוגמא: טור גיאומטרי

```
; input parameters: r n a
; 1. store parameters in memory
3
store
2
store
1
Store
; 2. place (1-r) on stack
1
load
1
-
; 3. place(1-r^n) on stack
2
load
```

$$\sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r} :$$

```
1
load
**
1
-
; 4. divide
; 5. multiply by a
```

דוגמא: טור גיאומטרי

```
; input parameters: r n a
; 1. store parameters in memory
3
store
2
store
1
Store
; 2. place (1-r) on stack
1
load
1
-
; 3. place(1-r^n) on stack
2
load
```

$$\sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r}$$

```
1
load
**
1
-
; 4. divide
/
; 5. multiply by a
```

דוגמא: טור גיאומטרי

```
; input parameters: r n a
; 1. store parameters in memory
3
store
2
store
1
Store
; 2. place (1-r) on stack
1
load
1
-
; 3. place(1-r^n) on stack
2
load
```

$$\sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r}$$

```
1
load
**
1
-
; 4. divide
/
; 5. multiply by a
```

דוגמא: טור גיאומטרי

```
; input parameters: r n a
; 1. store parameters in memory
3
store
2
store
1
Store
; 2. place (1-r) on stack
1
load
1
-
; 3. place(1-r^n) on stack
2
load
```

$$\sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r} :$$

```
1
load
**
1
-
; 4. divide
/
; 5. multiply by a
3
load
*
```

שיקולים נוספים

- דיון בשינויים שהוצגו:
- קוד המפרש תלוי במערך המכיל את רשימת הפקודות
 - הקוד לאתחול עלול להיות מורכב (כמו בדוגמת הזיכרון)
 - ניתן לכתוב מחלקה ייעודית לבניית המערך
 - צריך לשנות את קוד האתחול כאשר מוסיפים פקודה חדשה
 - האם עדיף על שינוי הפונקציה `runCommand?`
- יעילות מול מודולריות
 - סביר להניח שהקוד החדש איטי יותר, אך גם קל יותר לתחזוקה

חזרה: שדות מופע ומחלקה

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

דוגמא

```
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);
    }
}
```