

נראות (visibility)

- public – כל המחלקות
- protected – מחלקות באותה החבילה + מחלקות יורשות.
- package (default) – מחלקות באותה חבילה
- private – אותה המחלקה + מחלקות מקוננות / עוטפות



2

תוכנה 1 ב-JAVA
תרגול חזרה לפני בחינה

1

static vs. non-static

- משתנה סטטי (מחלקה) = משותף לכל המופעים – שינוי במופע אחד \Leftarrow שינוי בכל המופעים
- לא דורש קיום של מופע – ניתן לגשת דרך שם המחלקה: A.x (בתנאי שיש נראות)
- נקרא גם בשם משתנה גלובלי
- משתנה לא סטטי (מופע) = ייחודי לכל מופע – שינוי במופע אחד לא משפיע על השני



4

נראות - שאלות

```
package aaa;
public class A{
    public void f1();
    protected void f2();
    void f3();
    private void f4();
}
```

```
...
public class B ...{
    public void g(A a){
        // missing code
    }
}
```

אילו מהפקודות הבאות ניתן להוסיף ל-B.g() כך ש-B תתקמפל?
 a. f1();
 a. f2();
 a. f3();
 a. f4();



3

static vs. non-static - questions

```
public class A{
    public int i;
    public static Integer I;
    public static void soo(){
        ...
    }
    public void foo(){
        ...
    }
}
```

באילו משתי המתודות, foo ו-foo, יכולות להופיע הפקודות הבאות:

- foo - i=1;
- foo,soo - I = 5;
- foo,soo - soo();
- foo - foo();
- Foo - System.out.println(this)



6

static vs. non-static

- מתודה סטטית (מחלקה) - ניתנת להפעלה ללא קיום מופע של המחלקה. דוגמא: main
- בגוף המתודה ניתן להתייחס למשתנים ומתודות סטטיים בלבד של המחלקה
- מתודה לא סטטית (מופע) – ניתנת להפעלה רק באמצעות reference למופע
- בגוף המתודה ניתן להתייחס לכל המשתנים והמתודות של המחלקה (סטטים ולא סטטים)
- בגוף המתודה ניתן להתייחס ל- this = ה-reference למופע של המחלקה שבאמצעותו נקראה המתודה.



5

גושי אתחול סטטיים

- גוש אתחול סטטי מתבצע בעת טעינת המחלקה.
- דוגמה:

```
Public class A {  
    public static int x;  
    static {  
        x = 5;  
        System.out.println("Loading A...");  
    }  
}
```

8

static vs. non-static – questions (2)

- נניח שמתודה $f()$ אינה ניגשת לשדות / מתודות מופע של המחלקה.
- שאלה: האם במקרה זה רצוי להגדיר את $f()$ כ-
?static
- תשובה: תלוי במקרה. הגדרת $f()$ כ- static לא תאפשר פולימורפיזם בהפעלתה.
- פולימורפיזם = הפעלה בהתאם לטיפוס הדינמי של ה-
reference.
- פולימורפיזם מתאפשר ע"י override של המתודה במחלקות יורשות.

7

חתימה של מתודות - שאלות

```
public void f(Object o, String s)
```

- לאילו מהמתודות הבאות יש חתימה זהה לשל f ?
int f(Object x, String y)
- void f(Object o, String s) throws Exception
- void f(String s, Object o)

10

חתימה של מתודות

- האם החתימה כוללת:

- נראות (public/protected/...)?
- static / non-static?
- ערך החזר?
- שם המתודה
- סוגי הפרמטרים?
- שמות הפרמטרים?
- הצהרת החריגים שעלולים להיזרק (ביטוי throws)?

9

העמסה - שאלות

```
public interface I1{}  
public interface I2{}  
public class A implements I1, I2{  
    public static void f(I1 o){ System.out.println("I1");}  
    public static void f(I2 o){System.out.println("I2");}  
    public static void f(Object o){System.out.println("O");}  
}
```

```
A a = new A();  
Object o = a;  
I1 i1 = a;  
I2 i2 = a;
```

- האם השורות הבאות עוברות קומפילציה? אם כן מה יודפס?
f(o) - prints "O".
f(i1) - prints "I1".
f(i2) - prints "I2".
 f(a).

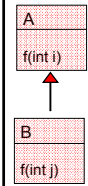
12

Overloading - העמסה

- העמסה של מתודות: שתי מתודות (או יותר) בעלות שם זהה אך חתימה שונה.
- רשימת הפרמטרים חייבת להיות שונה.
- כל שאר הרכיבים בהגדרת המתודה (מלבד השם) יכולים להיות שונים:
 - visibility (public /protected/...)
 - static / non-static
 - ערך החזר
 - הצהרת חריגים

11

חזרה על תנאים ל- override (תרגול 11, שקפים 7-10)



מתודת מופע m2 במחלקה B דורסת (overrides) מתודה m1 במחלקה A (A≠B) אם:

- B היא תת מחלקה של A
- ל- m1 ו- m2 יש חתימה זהה
- מתקיים לפחות אחד מהבאים:
 - m1 אינה פרטית ונגישה למחלקה של B.
 - m2 דורסת את m3 במחלקה C (C≠A,B) ו- m3 דורסת את m1 (הגדרה רקורסיבית)

14

Overriding - דריסה

- מתודות מופע (לא סטטיות) בלבד.
- overriding – מאפשר פולימורפיזם: דוגמא: a.f(...)
- אם לטיפוס הדינמי של a יש מתודה שמבצעת override ל- f(...) – היא תופעל.
- אחרת, תופעל המתודה f(...) שמוגדרת עבור הטיפוס הסטטי של a.
- חתימה זהה היא תנאי הכרחי אך לא מספיק לביצוע override.

13

אתחול ובנאים

- מה קורה בפקודה new B()?
 - טעינה של מחלקה B וכל המחלקות שהיא יורשת מהן (אם טרם נטענו)
 - הקצאת זיכרון לשדות העצם והצבת ערכי ברירת מחדל
 - הפעלה של הבנאי של B



16

overriding - questions

```
package a;
public class A {
    void foo() {
        System.out.println("A.foo()");
    }
    public void bar() {
        System.out.println("A.bar()");
        foo();
    }
}
```

Does the code compile? If no, why?
Does the code throw a runtime exception?
If yes, why? If no, what is the output?

```
package b;
public class B extends A {
    public void foo() {
        System.out.println("B.foo()");
    }
}
public static void main(String[] args) {
    A a = new B();
    a.bar();
}
```

The output is:
A.bar()
A.foo()

15

הרצת בנאים - שאלה

```
public class A {
    String bar = "A.bar";
    A() { foo(); }
    public void foo() {
        System.out.println("A.foo(): bar = " + bar);
    }
}

public class B extends A {
    String bar = "B.bar";
    B() { foo(); }
    public void foo() {
        System.out.println("B.foo(): bar = " + bar);
    }
}
public static void main(String[] args) {
    A a = new B();
}
```

The output is:
B.foo(): bar = null
B.foo(): bar = B.bar

18

אלגוריתם להרצת בנאים (תרגול 7, שקף 17)



1. אם השורה הראשונה היא this(...) – קרא לבנאי המתאים של אותה המחלקה המשך ל-4.
2. אם השורה הראשונה היא super(...) – קרא לבנאי המתאים של מחלקת האב.
3. הפעל את פקודות האתחול בהגדרת המשתנים ובלוקי אתחול (לא סטטיים) – לפי סדר הופעתם.
4. הרץ את גוף הבנאי

קריאות רקורסיביות

17

העברת ארגומנטים למתודה -

שאלה

```
public class Test {
    private static class Value { int v = 1; }
    public static void main(String[] args) {
        int v = 2;
        Value value = new Value();
        value.v = 3;
        foo(value, v);
        System.out.println(value.v + " " + v);
    }
    private static void foo(Value value, int v) {
        v = 4;
        value.v = 5;
        value = new Value();
        System.out.println(value.v + " " + v);
    }
}
```

20

העברת ארגומנטים למתודה

ארגומנטים תמיד מועברים By-Value:

- במתודה מוגדר משתנה (מקומי) מאותו הסוג של הארגומנט
- מתבצעת השמה של הארגומנט למשתנה של המתודה.
- מה קורה אם הארגומנט הוא מסוג reference?
- האם האובייקט מועתק?
- אילו פעולות במתודה יכולות להשפיע על ה-reference שהועבר כארגומנט?

```
String s="1234";
Value v= new Value();
f(s,v);
-----
f(String s, Value v){.....}
```

19

חוזים, מצב מופשט, פונקציית הפשטה ומשתמר הייצוג

חוזים: לכל שירות יש תנאי קדם ותנאי אחר.

- תנאי קדם ואחר יכולים להתייחס למצב הפנימי של האובייקט.
- דוגמא: תנאי קדם של המתודה `get(int i)` של מחלקה `MyList` יכול לדרוש שמספר האלמנטים ברשימה יהיה לפחות `i+1`.
- במקרה זה יש להוסיף שאילתות מתאימות לחוזה שמאפשרות לתשאל על מצבו הפנימי של האובייקט.
- בחוזה ניתן להגדיר "מצב מופשט" לצורך תיאור המצב הפנימי של האובייקט ולהשתמש בו לתיאור השירותים.
- לחוזה יכולים להיות מספר מימושים קונקרטיים שונים – המצב המופשט זהה לכולם.
- דוגמא: המנשק `IPoint` ושני מימושי `PolarPoint` ו-`CartesianPoint`.

22

העברת ארגומנטים למתודה - שאלה

```
public class Test {
    private static class Value { int v = 1; }
    public static void main(String[] args) {
        int v = 2;
        Value value = new Value();
        value.v = 3;
        foo(value, v);
        System.out.println(value.v + " " + v);
    }
    private static void foo(Value value2, int v2) {
        v2 = 4;
        value2.v = 5;
        value2 = new Value();
        System.out.println(value2.v + " " + v2);
    }
}
```

שינוי שמות משתנים

The output is:
1 4
5 2

21

דוגמא - DistSets

- המנשק `DistSets` מתאר אוסף של קבוצות זרות, לא ריקות של שלמים אי שליליים.
- תיאור פורמלי (מאוד) של המצב המופשט:

- $S = \{S_1, S_2, \dots, S_n\}, n > 0$
- $\forall i \in \{1, \dots, n\} S_i \subseteq \mathbb{Z}^*$
- $\forall i \in \{1, \dots, n\} S_i \neq \emptyset$
- $\forall i, j \in \{1, \dots, n\} i \neq j \implies S_i \cap S_j = \emptyset$

24

חוזים, מצב מופשט, פונקציית הפשטה ומשתמר הייצוג (2)

- פונקציית הפשטה – ממפה בין העצם למצב המופשט: מתארת את המצב המופשט כפונקציה של ערכי שדות העצם.

- שני עצמים עם ערכי שדות שונים יכולים לייצג את אותו מצב מופשט דוגמא: `SmartPoint`.
- משתמר – אוסף האילוצים הפומביים על המחלקה.
- משתמר הייצוג – אוסף כל האילוצים על המחלקה. אם משתמר הייצוג מתקיים – ניתן למפות את העצם למצב מופשט והאובייקט הוא במצב "חוקי". (לא כל עצם ניתן למפות למצב מופשט).

23

(3) DistSets - דוגמא

- המנשק DistSets מומש ע"י המחלקה SimpleArrayDistSets
 - כל קבוצה מיוצגת ע"י עץ.
 - כל צומת פנימי מצביע לאביו, ראש העץ מצביע לעצמו.
 - האיבר שבשורש העץ משמש כמזהה ייחודי של הקבוצה.

```
public class SimpleArrayDistSets
    implements DistSets{
    //fields:
    protected int[] parent;

    //methods:
    ....
}
```

26

(2) DistSets - דוגמא

- `public void makeSet(int x);`
 - // @pre: $\forall i \in \{1, \dots, n\} x \neq S_i$
 - // @abst: $S = \text{old}(S) \cup \{x\}$
- בכדי שהלקוח יוכל לבדוק את קיום תנאי הקדם - יש לדאוג לשאליות מתאימות. במקרה דנן:
 - // @pre `inASet(x)`

25

(5) DistSets - דוגמא

- משתמר הייצוג (מתייחס רק לערכי המערך `parent`):
 - לכל $x \in \{0, \dots, M-1\}$ מתקיים:
 - $\text{parent}[x] \in \{-1, 0, \dots, M-1\}$
 - $\text{parent}[\text{parent}[x]] \neq -1 \Leftrightarrow \text{parent}[x] \neq -1$
 - אם איבר נמצא בקבוצה - גם אביו בקבוצה.
 - לכל $x \in \{0, \dots, M-1\}$ מתקיים: $x = \text{parent}[x, m] \Leftrightarrow m = 0$.
 - תנאי זה מבטיח כי כל קבוצה היא אכן עץ. אם יהיו מעגלים המתודה שמחזירה את שורש העץ לא תסתיים (לולאה אינסופית)

```
private int root(int n) {
    int cur = n;
    while (parent[cur] != cur)
        cur = parent[cur];
    return cur;
}
```

28

(4) DistSets - דוגמא

- פונקציית ההפסטה:
 - `parent.length = M`
 - פונקציית עזר: $r(x, m) -$ מיהו האב הקדמון ה- m של x (-1 אם לא מוגדר) $r: \{0, \dots, M-1\} \times \mathbb{Z}^* \rightarrow \mathbb{Z}^*$.
 - אם $\text{parent}[x] = -1$ אזי $r(x, m) = -1$ לכל $m \in \mathbb{Z}^*$.
 - אחרת, אם $\text{parent}[x] = x$ (שרש עץ) $r(x, m) = -1$ לכל $m \in \mathbb{Z}^*$.
 - אחרת (x נמצא בקבוצה אך אינו השרש) $r(x, m) = r(\text{parent}[x], m)$, $r(x, 0) = x$ לכל $m \in \mathbb{Z}^*$.
 - $S(\text{this}) = \{S_1, \dots, S_n\}$ כך ש:
 - x שייך לקבוצה ב- S וגם $x \in \{0, \dots, M-1\}$ $\Leftrightarrow \text{parent}[x] \neq -1$
 - x ו- y שייכים לאותה קבוצה ב- S $\Leftrightarrow m_1, m_2 \in \mathbb{Z}^*$ קיימים וגם $x, y \in \{0, \dots, M-1\}$ כך שמתקיים $r(x, m_1) = r(y, m_2) \neq -1$

27

ג'וקרים

- המתודה `Collections.reverse(...)` מקבלת `List` (מכל סוג שהוא) והופכת את סדר האיברים בה.
- שאלה: מהו טיפוס הפרמטר של `reverse`?
 - תשובה: `List<?>`.
- בניח הטיפוס של `list` הוא `List<?>`. עם אילו ערכים ניתן לקרוא ל- `list.add(...)`
 - אף ערך - מאחר שלא ידוע שום דבר על הסוג של הרשימה!

```
אם היה כך לא היה בסדר
public void add(List<? super String> l) {
    l.add("moshe");
}
```

30

הכללה Generics

- מטרת ההכללה - לחסוך בהמרות (casting) שנבדקות בזמן ריצה ע"י ביצוע בדיקות בזמן קומפילציה.
- לאחר ביצוע הבדיקות בזמן קומפילציה - טיפוס ההכללה נמחק: $A < T >$ הופך ל- A (מחלקה לא מוכללת)
- `List<Object> lo = new ArrayList<Object>();`
- `List<String> ls = new ArrayList<String>();`
- האם הפקודות הבאות חוקיות:
 - `ls = lo` - ❌
 - `lo = ls` - ❌
 - `lo.add("123")` - ✅
 - `ls.add(new Object())` - ❌

29

טיפוסים נאים Raw Types

- טיפוס נא – ללא הכללה. למשל List, ArrayList, וכו'.
- טיפוסים נאים נחוצים לשם תמיכה בקוד ישן (generics הוסף בגרסא 1.5)
- בשימוש בטיפוסים נאים – אין בדיקת טיפוסים של הקומפיילר. הקומפיילר מזהיר (warning) במצבים שבהם יש צורך לבדוק את הטיפוס ולא ניתן. דוגמא:

```
List lraw = ls;
lraw.add(1);
```

Compiler
"checked"
warning

32

ג'וקרים

- ניתן לספק חסם עליון לג'וקר:

```
public static double sumList(List<? extends
Number> list) {
    double total = 0.0;
    for(Number n : list)
        total += n.doubleValue();
    return total;
}
```

- וגם חסם תחתון:

```
public static <T> boolean addAll(List<? super T>
c, T... a)
```

31

מחלקות אנונימיות

- האם למחלקה אנונימית יש גישה ל-
 – שדות מופע של המחלקה העוטפת?
 – שדות מחלקה של המחלקה העוטפת?
- האם למחלקה העוטפת יש גישה ל-
 – שדות מופע של המחלקה האנונימית?
 – שדות מחלקה של המחלקה האנונימית?
- מה קורה כשיש פנימית לפנימית?
 היחס טרנזיטיבי

ק
 כן ולא בהכרח אין
 "טיפוס" למחלקה
 האנונימית היא
 מרחיבה מחלקה ידועה
 אחרת, או מממשת
 ממשק חסום. אם
 למחלקה העוטפת יש
 גישה לממשק/מחלקה
 זו אז יש לה גישה
 לשדות ושדות שלה
 באנונימית

תזכרו שמחלקה מקומית יכולה לגשת רק למשתנים
 לוקאליים סופיים (final) בשירות בו היא מוגדרת.

34

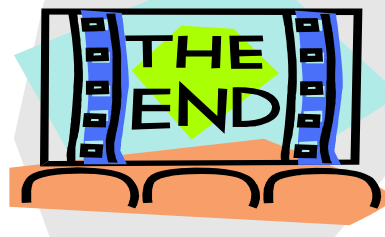
משתנים בממשקים?

- האם יכולים להיות משתנים בממשק?

```
public interface TestMe {
    void x();
    void y();
    final int t = 10;
}
```

- כן, והם מוגדרים

33



התאמה במחיימה!

35