

# חלק 8

מחלקות פנימיות, טיפוס  
מנייה, אתחול והשתקפות

# מרחב השמות בתוכנית ג'אווה

- מרחב השמות בתוכנית ג'אווה, כפי שהצגנו אותו עד כה, הוא מרחב דו־שכבתי, כמעט שלם, עם חוקי נראות (visibility) דו־מימדיים

- שלמות: לכל דבר יש שם; ראינו שניתן להעביר לשירות עצם או מערך אנונימי, אבל לכל טיפוס היה עד כה שם

```
vi.add( new Integer (3) );
```

```
printPrimes( new int[] { 1, 2, 3, 5, 7 } );
```

- דו־שכבתיות: אוסף של חבילות, שבכל אחת יש מחלקות; שם טיפוס מורכב משם החבילה ושם המחלקה; אוסף החבילות שטוח (אם כי נראה היררכי) ואוסף המחלקות בחבילה שטוח
- דו־מימדיות: נראות מוחלטת, בחבילה, או ליורשים

## **בעצם מרחב השמות יותר מורכב**

- יש לו יותר משתי שכבות: אפשר להגדיר מחלקות בתוך מחלקות, ואפילו מחלקות בתוך שירותים
- ואפשר גם ליצור מחלקות אנונימיות
- בחלק הזה של הקורס נראה את המנגנונים הללו

# מחלקות פנימיות סטאטיות

- הסוג הפשוט ביותר של מחלקה פנימית

```
public class PersistentVersionedString
    implements VersionedString {
    public static class PVSFilter
        implements java.io.FileFilter {
        public boolean accept(java.io.File f) {
            return f.getName().endsWith(".pvs");
        }
    }
    ...
}
```

# שימוש במחלקה פנימית סטאטית

- מחלקה כזו היא מחלקה רגילה לכל דבר, פרט לזה ששמה המלא כולל את שם המחלקה החיצונית

```
FileFilter filter = new  
    PersistentVersionedString.PVSFilter();  
File dir = new File("/Projects/oopj");  
File[] files = dir.listFiles( filter );
```

- עצמים מהמחלקה הפנימית הם עצמאיים לחלוטין; אין שום קשר בינם ובין עצמים מהמחלקה החיצונית
- אבל בגלל שהפנימית היא מעין שדה של החיצונית, שירותים של שתיהן יכולים לגשת לכל שדות המחלקה של שתיהן, גם לשדות מוגנים (private ו-protected)

# הגנה על מחלקות פנימיות סטאטיות

- אם המחלקה הפנימית אינה ציבורית (אינה מוגדרת `public`), הטיפוס שלה מוסתר, אבל עצמים מהמחלקה אינם מוסתרים אם יש התייחסות אליהם

```
public class PersVS ... {  
    private static class PVSFilter ... {...}  
    public static FileFilter getFilter()  
        { return new PVSFilter(); }  
    ...  
}
```

```
FileFilter f = new PersVS.PVSFilter(); error  
FileFilter f = PersVS.getFilter(); ok
```

# עוד שימוש למחלקה פנימית

- Version הוא מחלקת עזר במימוש של `LinkedVersionedString`, עדיף להסתיר אותה

```
public class LinkedVersionedString
    extends VersionedString {
    private class Version {
        ...
    }
    ...
}
```

# מחלקות פנימיות לא סטאטיות

- מבנה מיותר בג'אווה; מעט מאוד שימושים אמיתיים. ובכל זאת, מה זה?
- מחלקה של עצמים שכל אחד מהם "שייך" לעצם של המחלקה המכילה ומכיר את שדות המופע שלו



# קשירה של מחלקה פנימית

```
public class Outer {  
    private int o;  
    public class Inner {  
        private int i;  
        public void set() { i = o; }  
        public int get() { return i; }  
    }  
  
    public Inner getInner() {  
        return new Inner(); }  
    public void increment() { o++; }  
}
```

# קשירה של מחלקה פנימית (המשך)

```
Outer x          = new Outer();  
Outer.Inner y1  = x.getInner();  
Outer.Inner y2  = x.getInner();
```

```
x.increment();  now x.o == 1  
y1.set();      y1.i = x.o == 1  
x.increment();  now x.o == 2  
y2.set();      y2.i = x.o == 2  
y1.get();      returns 1  
y2.get();      returns 2
```

# אותה תוצאה עם מחלקה פנימית סטאטית

```
public class Outer {  
    public static class SInner {  
        private int i;  
        private Outer outer;           an explicit reference  
        public SInner(Outer outer) {  
            this.outer = outer; }  
        public void set() { i = outer.o; }  
    }  
    public SInner getInner() {  
        return new SInner(this); }  
    ...  
}
```

# תחביר (מסובך) לשימוש במח' פנימיות

- בנייה ישירה של עצם פנימי

```
Outer x = new Outer();
```

```
Outer.Inner y3 = x.new Outer.Inner();
```

- הרחבה של מחלקה פנימית על ידי מחלקה רגילה

```
class SubInner extends Outer.Inner {  
    public SubInner (Outer outer) {  
        outer.super(); } invoke the super's constructor
```

- שימוש בשדה מוסתר של המחלקה החיצונית על ידי הפנימית,  
ובשדה מוסתר של המחלקה שהחיצונית מרחיבה

```
Outer.this.field
```

```
Outer.super.field
```

## מחלקה מקומית (לא שימושי)

```
public VersionedString someMethod() {  
    final int fi = 3;  
        int mi = 4;  
    class LocalVS implements VersionedString {  
        public void add(String s) {  
            int x = fi; ok  
            int y = mi; compilation error; mi is not final  
            ... }  
        ... }  
    return new LocalVS();  
}
```

# תכונות מחלקה מקומית

- מוגדרת בתוך בלוק (בדרך כלל שרות).
- לא סטאטית; אסור להשתמש במילת המפתח `static`
- הטיפוס לא מוכר מחוץ לגוש (שירות) שבו היא מוגדרת, ולכן אין צורך בהגדרת נראות (`private` או `protected`)
- מותר לה להשתמש במשתנים של הגוש (שירות) שבו היא מוגדרת; זו הסיבה להגדירה מקומית ולא סתם פנימית
- העצמים עצמם יכולים לחמוק מהגוש שבו המחלקה מוגדרת, ולכן מותר לה לגשת רק למשתנים שערכם לא ישתנה אחרי סיום פעולת הגוש (משתנים שמוגדרים `final`), אחרת היא הייתה עלולה לגשת למשתנים שכבר לא קיימים
- אין למחלקות כאלה הרבה שימושים

# מחלקות אנונימיות (מאוד שימושיות)

- מחלקות אנונימיות הן בעצם הסיבה להגדרה של מחלקות פנימיות ומקומיות

- משמשות בדרך כלל לאריזה של פרוצדורה שמיועדת להישמר במבנה נתונים להפעלה בעתיד

```
Button b = new Button(...);
```

```
b.addMouseListener( new MouseListener() {  
    public void mouseClicked(Event e) {...}});
```

- העברנו לכפתור פרוצדורה שהוא אמור להפעיל כאשר לוחצים על הכפתור; הפרוצדורה הזו תגרום לתוצא הלוואי הרצוי; כדי לגרום לתוצא הלוואי, היא צריכה לשמור התייחסות לעצמים שהיא תשנה את מצבם; פרוצדורה כזו נקראת closure

# שימוש טיפוס במחלקה אנונימית

```
interface MouseListener
{ public void mouseClicked(); }

class Button {
    Set mouse_listeners = new TreeSet();
    public void addMouseListener(MouseListener ml)
        { mouse_listeners.add(ml); }
    ...
}

b.addMouseListener( new MouseListener() {
    public void mouseClicked(Event e) {...});
```



# משמעות הקריאה

- הקריאה

```
b.addMouseListener( new MouseListener() {  
    public void mouseClicked(Event e) {...}});
```

- היא קיצור של קטע הקוד הבא

```
class ML implements MouseListener {  
    public void mouseClicked(Event e) {...}  
}
```

```
ML ml = new ML();
```

```
b.addMouseListener( ml);
```

- חסכנו את הצורך בשם של המחלקה ML והמשתנה ml

# סיכום מחלקות פנימיות

- הסוג השימושי ביותר הוא מחלקות אנונימיות, משום שהוא מפצה על היעדר התייחסויות לפרוצדורות בג'אווה
- מחלקה אנונימית מאפשרת להעביר פרוצדורה שהתנהגותה תלויה בהקשר בו הוגדרה (יכולה להשתמש בשדות של העצם ובמשתנים מקובעים של השירות בו היא מוגדרת).
- מחלקות אנונימיות שימושיות בתבניות התיכון `observer`, `command`, `strategy` ו-
- מחלקות פנימיות סטאטיות מאפשרות לעצב את מרחב השמות באופן גמיש ולהסתיר מחלקות עזר
- השאר (פנימיות לא סטאטיות ומקומיות) פחות שימושיות
- עדיף להגביל את התלות בין המחלקה החיצונית והפנימית

# טיפוסי מנייה Enumerated Types

- מג'אווה 5: סוג מיוחד של טיפוס ייחוס (נקרא בקיצור enum)
- מספר סופי (בדרך כלל קטן) של ערכי מנייה (קבועי enum)
- הערכים מופיעים במפורש בהגדרת הטיפוס.

```
public enum DownloadStatus
```

```
{CONNECTING, READING, DONE, ERROR }
```

- הערכים הם כמו שדות `static final` ולכן נהוג להשתמש בשמות שכל האותיות שלהם גדולות
- ניתן להתייחס אליהם בשם כמו `DownloadStatus.DONE`
- ערכו של משתנה מהטיפוס `DownloadStatus` יכול להיות אחד מארבעת הערכים האלה או `null` בלבד.

# מה עשינו כשאין טיפוסים מנייה

- בגירסאות קודמות של ג'אווה היה צורך להגדיר מחלקה או ממשק שבתוכו

```
public static final int CONNECTING = 1;  
public static final int READING = 2;  
public static final int DONE = 3;  
public static final int ERROR = 4;
```

- לקבועים שלמים אין בטיחות טיפוסים (type safety)
- אם יש שני טיפוסים כאלה, ניתן לבצע השמה מאחד לשני: הם אינם נבדלים כי הם שלמים מבחינת הקומפילר

# טיפוסי מנייה אינם קבועים שלמים

- טיפוסי מנייה אינם קבועים שלמים
- הטיפוס הוא מחלקה, והערכים הם עצמים מהמחלקה.
- יש בטיחות טיפוסים (לא ניתן לערבב בין שני טיפוסים כאלה)
- אין בנאי מיוצא (`public`). לא ניתן ליצור משתנה מהטיפוס שאין לו ערך. העצמים היחידים הם אלה שהוגדרו ב `enum`
- לא ניתן ליצור עותקים של העצמים
- העצמים האלה מקובעים
- ניתן להשוות ערכים אלה ב `==` . קיים גם `equals` שממומש באמצעות `==` ולא ניתן לדריסה. ערכי `enum` יכולים להיות אברים באוסף (`collection`) כמו `Set`, `List`, `Map`

## עוד על טיפוס מנייה

- מממשים את הממשק Comparable. השרות compareTo() מוגדר עפ"י הסדר בו הופיעו קבועי המנייה
- שרות toString() מחזיר את המזהה (כמו DONE), אך ניתן להגדירו מחדש.
- שרות מחלקה (סטטי) valueOf שהוא ההיפוך של toString לדוגמא: הקריאה הבאה מחזירה DONE  
DownloadStatus.valueOf("DONE")
- שרות ordinal() מחזיר שלם לפי סדר הקבועים החל מ 0. לדוגמא: הקריאה DONE.ordinal() מחזירה 2
- שרות מחלקה values מחזיר מערך של כל ערכי ה enum

# טיפוסי מנייה כמחלקות

- טיפוס מנייה הם תת מחלקה של `java.lang.Enum` שבעצמה אינה `enum`. לא ניתן ליצור טיפוס מנייה ע"י ירושה מפורשת מ `java.lang.Enum`
- לא ניתן לרשת מטיפוסי מנייה, כאילו הם הוגדרו `final`
- טיפוס מנייה יכולים לממש מנשקים, וניתן להגדיר להם שרותים (לא נפרט)
- דוגמא: משפט `switch` עם טיפוס מנייה. ערך `null` יגרום לחריג. אם לא כל הערכים מופיעים כתווית (ואם לא מופיעה תווית `default`) נקבל אזהרה מהקומפילר

```
DownloadStatus status = imagLoader.getStatus();
switch(status) {
case CONNECTING:
    imagLoader.waitForConnection();
    imagLoader.startReading();
    break;
case READING:
    break;
case DONE:
    return imagLoader.waitForConnection();
case ERROR:
    throw new IOException(imagLoader.getError())
}
}
```



# שימושים של טיפוס מנייה

- מיפוי מטיפוס מנייה (כטיפוס המפתח) לערכים אחרים. בג'אווה 1.5 הוסיפו מחלקה מיוחדת EnumMap לצורך זה
- קבוצות של ערכי מנייה - במקום להשתמש בקבועים שהם חזקות של 2 , ופעולות לוגיות על ביטים.

```
public enum DrinkFlags {
    SHORT, TALL, GRANDE, DOUBLE, SKINNY,
    WITH_ROOM, SPLIT_SHOT, DECAF }
EnumSet<DrinkFlags> drinkflags =
    EnumSet.of(DrinkFlags.DOUBLE,
               DrinkFlags.SHORT,
               DrinkFlags.WITH_ROOM);
boolean isbig =
    drinkflags.contains(DrinkFlags.TALL ||
                       DrinkFlags.GRANDE);
```

# אתחול שדות

- ניתן לאתחל שדות בכמה דרכים:
- אתחול אוטומטי (ערך המחדל של הטיפוס)
- אתחול מפורש בהגדרת השדה:

```
class SomeClass {  
    private int x = 13;  
    private static String msg = "hello";  
    ....  
}
```

- מה נעשה אם האתחול מורכב יותר, ודורש מספר פעולות?

# אתחול שדות - המשך

- שדות מופע ניתן לאתחל בבנאי.
- הערכים שהבנאי נותן לשדות המופע נקבעים כך שלאחר ביצוע הבנאי העצם מקיים את המשתמר של המחלקה.
- מה לגבי שדות מחלקה (סטטיים)?
- שדות המחלקה מהווים מעין עצם עצמאי; השירותים שלו הם שירותי המחלקה; אבל לכל עצם מהמחלקה יש התייחסות אליו ומותר לשירותי המופע לגשת אליו
- לכן, לשדות המחלקה יהיה משתמר משלהם
- גם שירותי המחלקה וגם שירותי המופע חייבים לכבד אותו, כי לכולם יש גישה לשדות המחלקה

# אתחול שדות המחלקה

- לפעמים המשתמר של שדות המחלקה פשוט ואתחול האוטומטי או הידני שלהם מבטיח את קיומו,

```
class Version {  
    private static Version free_list; null is ok  
    ...  
class Paragraph {  
    public final static int DIR_LTR      =0x01;  
    ...
```

- אבל לפעמים זה לא מספיק; לפעמים המשתמר מסובך מדי, ולפעמים האתחול עלול להודיע על חריגים שצריך לטפל בהם

# אתחול סטאטי

```
class Sentence {
    private static Set prepositions;
    static {
        prepositions = new HashSet();
        try {
            FileReader r
                = new FileReader("preps.txt");
            ... read the file and fill the set
        } catch (IOException e) {...}
    }
    ...
}
```

# חוקי האתחול

- האתחול הסטאטי יתבצע לפני הפעלת שירות מהמחלקה, כולל שירותי מחלקה וכולל בנאים
- אם יש כמה גושי אתחול סטאטיים (ואולי גם פסוקי אתחול פשוטים) הם יתבצעו לפי סדר הופעתם; סגנונית, עדיף לרכז את כל האתחולים הסטאטיים לגוש אחד
- במחלקה מרחיבה אם גם למחלקת הבסיס יש אתחול סטאטי, האתחול של הבסיס יתבצע לפני האתחול של המרחיבה
- אם האתחול הסטאטי מפעיל שירותים של מחלקות אחרות, הן יאותחלו קודם
- סביבת זמן הריצה בוחרת בעצמה את זמן האתחול הסטאטי בהתחשב באילוצים הללו

# singleton במקום אתחול סטאטי מסובך

- עדיף אולי להשתמש בתבנית היחיד (singleton) במקום בשדות מחלקה מרובים; מצב המחלקה נהפך למצב של עצם רגיל, שמשותף לכל העצמים במחלקה

```
class MyClassStatic {  
    ... instance fields  
    public MyClassStatic () {...}  
    ... instance methods  
}  
  
class MyClass {  
    private final static MyClassStatic mcs  
        = new MyClassStatic();  
}
```



# כמה מילים על השתקפות (reflection)

- בג'אווה, המבנה של הקוד (מחלקות, שירותים, ושדות) זמין בזמן ריצה ומאפשר לחקור את מבנה הקוד
- יש מחלקה שהעצמים שלה מייצגים מחלקות, מחלקה שמייצגת חבילות, מחלקה לשירותים, מחלקה לשדות, ומחלקה לבנאים
- ניתן להפעיל בנאים ושירותים בעזרת העצמים המייצגים

`java.lang.Class`

`java.lang.Package`

`java.lang.reflect.Constructor`

`java.lang.reflect.Method`

`java.lang.reflect.Field`

# המחלקה

```
VersionedString vs = ...  
Class x = vs.getClass();           an Object method  
Class y = VersionedString.class;   literal  
Class z = Class.forName("VersionedString");  
                                     static lookup
```

- העצם שמייצג את המחלקה יכול להחזיר את הפרטים לגביה: שם, את מי מרחיבה ומממשת, שדות, שירותים ובנאים
- למשל, בניית עצם תוך שימוש בבנאי ברירת המחדל:

```
VersionedString a  
    = (VersionedString) z.newInstance();
```

- אפשר אפילו ליצור עצמים באופן דינאמי