

# תוכנה 1

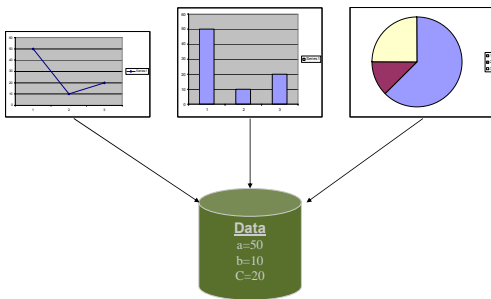
Design Patterns :10 תרגול  
ומחלקות פנימיות  
נעמה מאיר, מתי שמרת

## Design Patterns

- A general reusable solution to recurring design problems.
  - Not a recipe
- A higher level language for design
  - Factory, Singleton, observer and not "this class inherits from that other class"
- *Design Patterns: Elements of Reusable Object-Oriented Software*
- Lots of information online



## Different Views

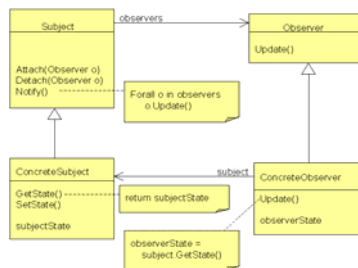


## Different Views (cont.)

- When the data change all views should change
  - Views dependent on data
- Views may vary, more added in the future
- Data store implementation may changes
- We want:
  - Separate the data aspect from the view one
  - Notify views upon change in data

## The Observer Design Pattern

- A.k.a publish/subscribe



## Observer and Java

- Java provides an observer interface and an Observable class.
- Subclass Observable to implement your own subject
  - registration and removal of observers
  - notification
- Implement Observer
- Other uses of this pattern throughout the JDK

## Example Code - Subject

```
public class IntegerDataBag extends Observable
implements Iterable<Integer> {
    private ArrayList<Integer> list = new ArrayList<Integer>();

    public void add( Integer i ) {
        list.add(i);
        setChanged();
        notifyObservers();
    }

    public Iterator<Integer> iterator() {
        return list.iterator();
    }

    public Integer remove( int index ) {
        if( index < list.size() ) {
            Integer i = list.remove( index );
            setChanged();
            notifyObservers();
            return i;
        }
        return null;
    }
}
```

## Example Code - Observer

```
public class IntegerAdder implements Observer {
    private IntegerDataBag bag;

    public IntegerAdder( IntegerDataBag bag ) {
        this.bag = bag;
        bag.addObserver( this );
    }

    public void update(Observable o, Object arg) {
        if (o == bag) {
            println("The contents of the IntegerDataBag have changed.");
            int sum = 0;
            for (Integer i : bag) {
                sum += i;
            }
            println("The new sum of the integers is: " + sum);
        }
    }
    ...
}
```

8

## מחלקות פנימיות (מקוננות) Inner (Nested) Classes

9

## Inner Classes

- מחלקה פנימית היא מחלקה שהוגדרה בתחום (Scope – בין המסולסליים) של מחלקה אחרת

■ דוגמא:

```
public class House {
    private String address;
    public class Room {
        private double width;
        private double height;
    }
}
```

שימוש לב  
Room אינה שדה של  
House המחלקה

10

## מחלקות פנימיות

- הגדרת מחלקה כפנימית מרמזת על היחס בין המחלקה הפנימית והמחלקה העוטפת:
- למחלקה הפנימית יש משמעות רק בהקשר של המחלקה החיצונית
- למחלקה הפנימית יש הכרות אינטימית עם המחלקה החיצונית
- המחלקה הפנימית היא מחלקת עזר של המחלקה החיצונית
- דוגמאות:
  - Iterator - Collection
  - Brain - Body
  - מבני נתונים המוגדרים ברקורסיה: List - Cell

11

## Inner Classes

- ב Java כל מופע של עצם מטיפוס המחלקה הפנימית צריך להיות משויך לעצם מטיפוס המחלקה העוטפת

■ השלכות

- תחביר מיוחד לבנאי
- לעצם מטיפוס המחלקה הפנימית יש שדה הפנייה שמיצור אוטומטית לעצם מהמחלקה העוטפת
- כתוצאה לכך יש למחלה הפנימית גישה לשדות ולשרותים (אפילו פרטיים!) של המחלקה העוטפת ולהיפך

12

## Inner Classes

```
public class House {
    private String address;
    public class Room {
        // implicit reference to a House
        private double width;
        private double height;
        public String toString(){
            return "Room inside: " + address;
        }
    }
}
```

13

## Inner Classes

```
public class House {
    private String address;
    private double height;
    public class Room {
        // implicit reference to a House
        private double height;
        public String toString(){
            return "Room height: " + height
                + " House height: " + House.this.height;
        }
    }
}
```

Height of House

Height of Room

Height of Room Same as this.height

14

## יצירת מופעים

- כאשר המחלקה החיצונית יוצרת מופע של עצם מטיפוס המחלקה הפנימית אזי העצם יוצר בהקשר של העצם היוצר
- כאשר עצם מטיפוס המחלקה הפנימית נוצר מחוץ למחלקה העוטפת, יש צורך בתחביר מיוחד

15

## יצירת מופע ע"י המחלקה החיצונית

```
public class House {
    private String address;
    public void test() {
        Room r = new Room();
        System.out.println( r );
    }

    public class Room {
        ...
    }
}
```

16

## יצירת מופע שלא ע"י המחלקה החיצונית

```
public class Test {
    public static void main(String[] args){
        House h = new House();
        House.Room r = h.new Room();
    }
}
```

`outerObject.new InnerClassName`

17

## Static Nested Classes

- ניתן להגדיר מחלקה פנימית כ `static` ובכך לציין שהיא אינה קשורה למופע מסוים של המחלקה העוטפת
- הדבר אנלוגי למחלקה שכל שרותיה הגודרו כ `static` והיא משמשת כספרייה עבור מחלקה מסוימת
- בשפת C++ יחס זה מושג ע"י הגדרת יחס `friend`

18

```

public class House {
    private String address;
    public static class Room {
        public String toString(){
            return "Room " + address;
        }
    }
}

public class Test {
    public static void main(String[] args){
        House.Room r = new House.Room();
        ...
    }
}

```

Error: this room is not related to any house

Not related to any specific house

`new OuterClassName.InnerClassName`

## הגנה על מחלקות פנימיות

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

```

public class Outer {
    private static class Inner implement Interface {
        ...
    }

    public static Interface getInner() {
        return new Inner ();
    }
}

Interface i = new Outer.Inner(); //error
Interface i = Outer.getInner(); // ok

```

20

## מחלקות מקומיות - מחלקות פנימיות בתוך מתודות

- ניתן להגדיר מחלקה פנימית בתוך מתודה של המחלקה החיצונית
- הדבר מגביל את תחום ההכרה של אותה מחלקה לתחום אותה המתודה בלבד
- המחלקה הפנימית תוכל להשתמש במשתנים מקומיים של המתודה רק אם הם הוגדרו כ `final` (מדוע?)

21

## מחלקות מקומיות

```

public class Test {
    ...
    public void test () {
        class Info {
            private int x;
            public Info(int x) {this.x=x;}
            public String toString() {
                return "*" + x + "*";
            }
        };
        Info info = new Info(0);
        System.out.println(info);
    }
}

```

22

## שימוש במשתנים מקומיים

```

public class Test {
    public void test (int x) {
        final int y = x+3;
        class Info {
            public String toString(){
                return "*" + y + "*";
            }
        };
        System.out.println( new Info());
    }
}

```

23

## מחלקות אנונימיות

- בעזרת מחלקות פנימיות ניתן להגדיר מחלקות אנונימיות – מחלקות ללא שם
- מחלקות אנונימיות שימושיות מאוד במערכות מונחות ארועים (כגון GUI) וילמדו בהמשך הקורס

24

## הידור של מחלקות פנימיות

- המהדר (קומפיילר) יוצר קובץ class. עבור כל מחלקה. מחלקה פנימית אינה שונה במובן זה ממחלקה רגילה
- שם המחלקה הפנימית יהיה `Outer$Inner.class`
- אם המחלקה הפנימית אנונימית, שם המחלקה שיוצר הקומפיילר יהיה `Outer$1.class`