

תוכנה 1

תרגול מספר 11:

מחלקות מקוננות Nested Classes

תבנית העיצוב Observer

NESTED CLASSES

מחלקות מקוננות

מחלקה מקוננת

• מחלקה שמוגדרת בתוך מחלקה אחרת

.1 סטטית (static member)

.2 לא סטטית (nonstatic member)

.3 אנונימית (anonymous)

.4 מקומית (local)

מחלקות פנימיות
(inner)

```
class Outer {
    static class NestedButNotInner {
        ...
    }
    class Inner {
        ...
    }
}
```

בשביל מה זה טוב

• קיבוץ לוגי

אם עושים שימוש בטיפוס רק בהקשר של טיפוס מסוים נטמיע את הטיפוס כדי לשמר את הקשר הלוגי

• הכמסה מוגברת

על ידי הטמעת טיפוס אחד באחר אנו חושפים את המידע הפרטי רק לטיפוס המוטמע ולא לכולם

• קריאות

מיקום הגדרת טיפוס בסמוך למקום השימוש בו

תכונות משותפות

- למחלקה מקוננת יש גישה לשדות הפרטיים של המחלקה העוטפת ולהפך
- הנראות של המחלקה היא עבור "צד שלישי"
- אלו מחלקות (כמעט) רגילות לכל דבר ועניין
- יכולות להיות אבסטרקטיות, לממש מנשקים, לרשת ממחלקות אחרות וכדומה

Static Member Class

- מחלקה רגילה ש"במקרה" מוגדרת בתוך מחלקה אחרת
- החוקים החלים על איברים סטטיים אחרים חלים גם על מחלקות סטטיות

- גישה לשדות / פונקציות סטטיים בלבד
- גישה לאיברים לא סטטיים רק בעזרת הפניה לאובייקט
- גישה לטיפוס בעזרת שם המחלקה העוטפת

```
OuterClass.StaticNestedClass
```

- יצירת אובייקט

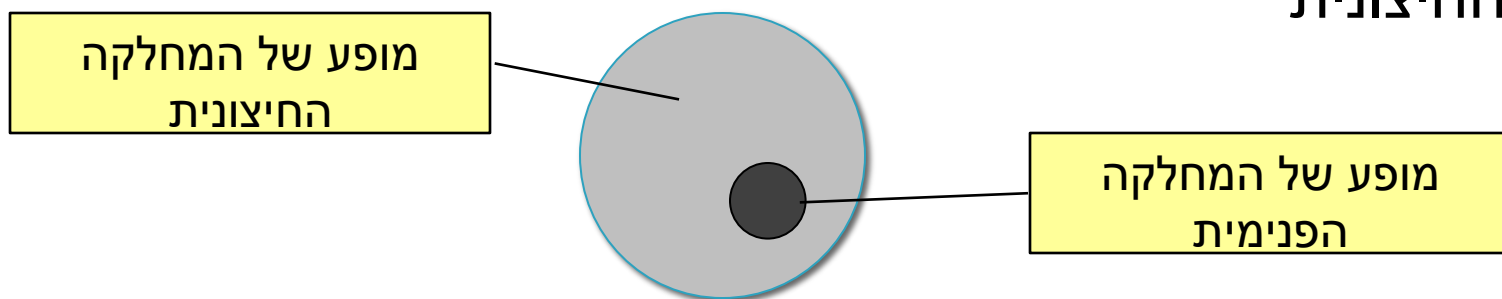
```
OuterClass.StaticNestedClass nested =  
    new OuterClass.StaticNestedClass();
```

AbstractMap

```
public abstract class AbstractMap<K,V> implements Map<K,V> {  
  
    public static class SimpleEntry<K,V>  
        implements Entry<K,V>, java.io.Serializable {  
        private final K key;  
        private V value;  
  
        ...  
    }  
  
    ...  
}
```

Non-static Member Class

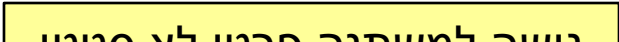
- כל מופע של המחלקה הפנימית משויך למופע של המחלקה החיצונית



- השיוך מבוצע בזמן יצירת האובייקט ואינו ניתן לשינוי
- באובייקט הפנימי קיימת הפניה לאובייקט החיצוני (qualified this)

House Example

```
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;  
        }  
    }  
}
```



גישה למשתנה פרטי לא סטטי

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
Same as `this.height`

AbstractList

```
public abstract class AbstractList<E> extends
    AbstractCollection<E> implements List<E> {
    public Iterator<E> iterator() {
        return new Itr();
    }

    private class Itr implements Iterator<E> {
        ...
    }

    private class ListItr extends Itr implements
        ListIterator<E> {
        ...
    }
}
```

יצירת מופעים

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

`outerObject.new InnerClassConstructor(...)`

מחלקות אנונימיות

- מחלקה ללא שם
- הגדרה ויצירת מופע בנקודת השימוש
- מגבלות:
- חייבת לרשת מטיפוס קיים (מנשק או מחלקה)
- לא ניתן להגדיר איברים סטטיים, לא ניתן להשתמש בהקשר שדורש שם (instanceof), לא ניתן לממש מספר מנשקים, לקוחות מוגבלים לממשק של טיפוס האב, גישה למשתנים מקומיים שהם final בלבד.
- מחלקה אנונימית צריכה להיות קצרה כדי לא לפגוע בקריאות של הקוד

דוגמאות שימוש

Function object (functor) •

• מיון מחוזות לפי אורך

```
Arrays.sort(stringArray, new Comparator<String>() {
    public int compare(String s1, String s2) {
        return s1.length() - s2.length();
    }
})
```

• מימוש איטרטור

```
public Iterator<E> iterator() {
    return new Iterator<E>() {
        boolean hasNext() {...}
        E next() {...}
        void remove() {...}
    }
}
```

המרה ממערך לרשימה

```
static List<Integer> intArrayAsList(final int[] a) {  
    if (a == null)  
        throw new IllegalArgumentException();  
    return new AbstractList<Integer>() {  
        public Integer get(int i) {  
            return a[i];  
        }  
        public Integer set(int i, Integer val) {  
            int oldVal = a[i];  
            a[i] = val;  
            return oldVal;  
        }  
        public int size() {  
            return a.length;  
        }  
    };  
}
```

גישה למשתנים מקומיים
שהוגדרו final

מחלקות מקומיות

- מוגדרות בתוך מתודות
- יש להם שם וניתן להשתמש בהם מספר פעמים, בתוך אותה מתודה
- אובייקט עוטף רק אם הוגדרו בהקשר לא סטטי; לא ניתן להגדיר משתנים סטטיים
- המחלקה הפנימית תוכל להשתמש גם במשתנים מקומיים של המתודה אבל רק אם הם הוגדרו כ-`final`

דוגמא - המרה ממערך לרשימה

```
static List<Integer> intArrayAsList(final int[] a) {
    if (a == null)
        throw new IllegalArgumentException();
    class IntegerList extends AbstractList<Integer> {
        public Integer get(int i) {
            return a[i];
        }
        public Integer set(int i, Integer val) {
            int oldVal = a[i];
            a[i] = val;
            return oldVal;
        }
        public int size() {
            return a.length;
        }
    }
    return new IntegerList();
}
```

OBSERVER תבנית העיצוב

תבניות עיצוב

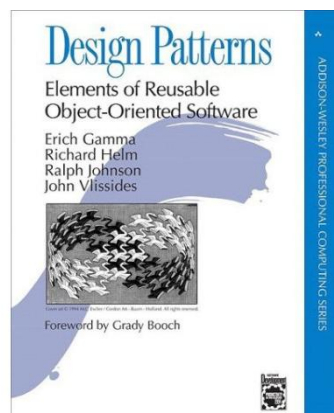
Observer תבנית העיצוב

תבניות עיצוב (Design Patterns)

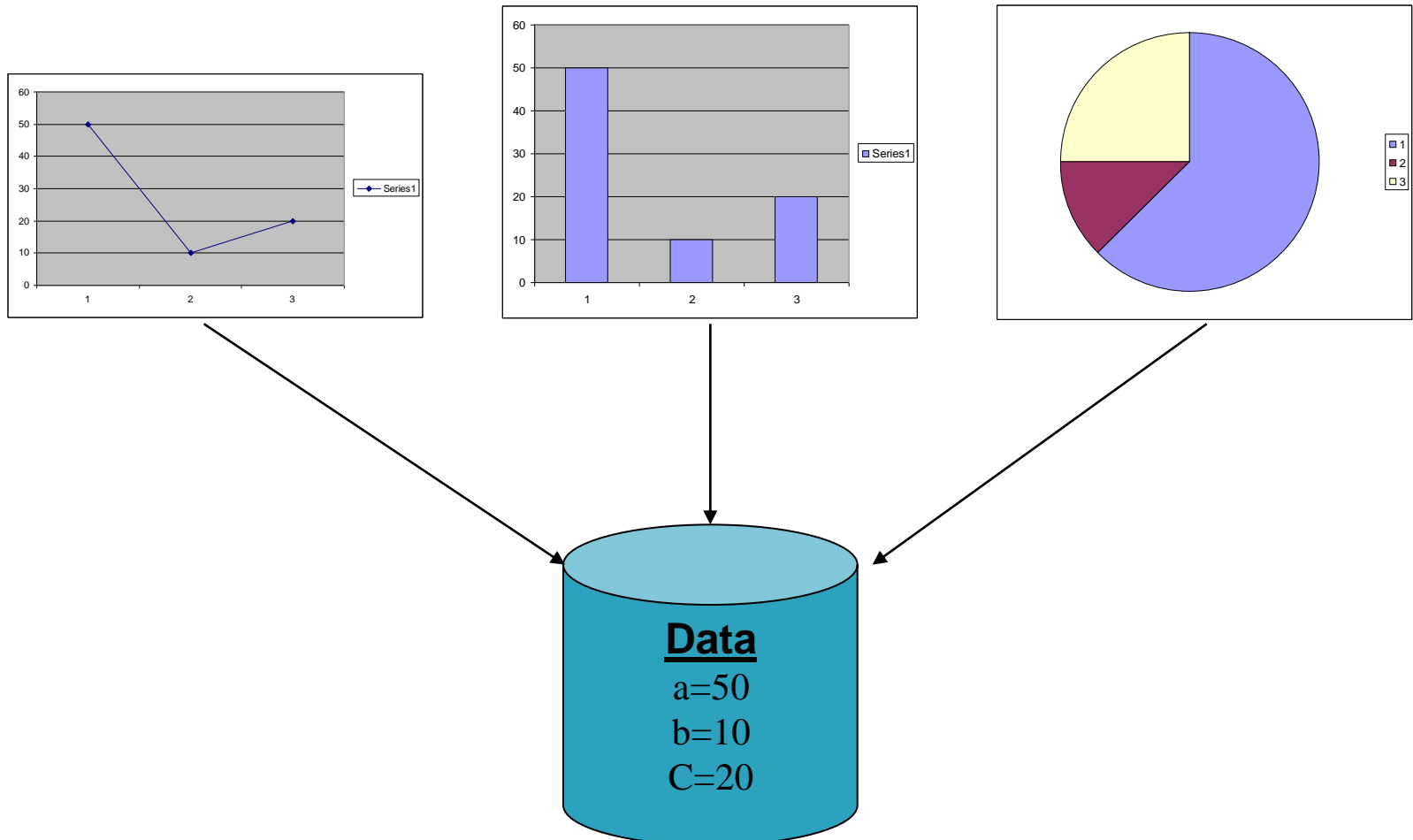
- פתרונות כלליים לבעיות עיצוב שחוזרות על עצמן
- מגדירים שפה כללית יותר לדין על עיצוב התכנית
- Factory, Singleton, Observer במקום
"המחלקה A יורשת מהמחלקה B"

• **ספר:** *Design Patterns: Elements of Reusable Object-Oriented Software*

• מידע רב בנושא קיים ברשת



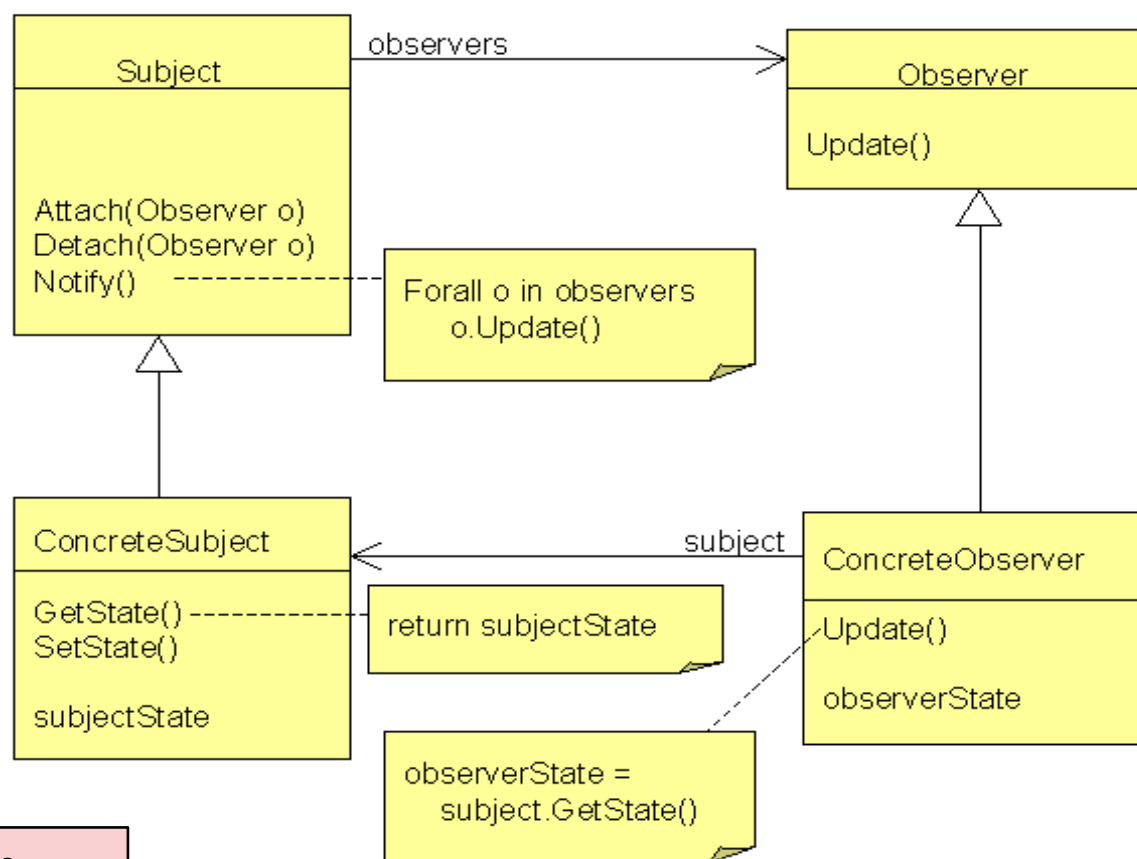
Different Views



Different Views (cont.)

- When the data changes, all views should change
 - Views dependant 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

תבנית העיצוב Observer



ניתן לתכנן את
המחלקות גם אחרת!

Observer בג'אווה

- ג'אווה מספקת לנו מנשק Observer, ומחלקה Observable
- נממש את Observer
- כדי ליצור subject, נכתוב מחלקה שיורשת מ-Observable. כבר נתונים לנו
 - הוספה והסרה של Observers
 - מסירת ההודעה ל-Observers הרשומים

Observer

java.util

Interface Observer

```
public interface Observer
```

A class can implement the `Observer` interface when it wants to be informed of changes in observable objects.

Since:

JDK1.0

See Also:

[Observable](#)

Method Summary

Methods

Modifier and Type	Method and Description
void	<code>update(Observable o, Object arg)</code> This method is called whenever the observed object is changed.

Observable

Method Summary

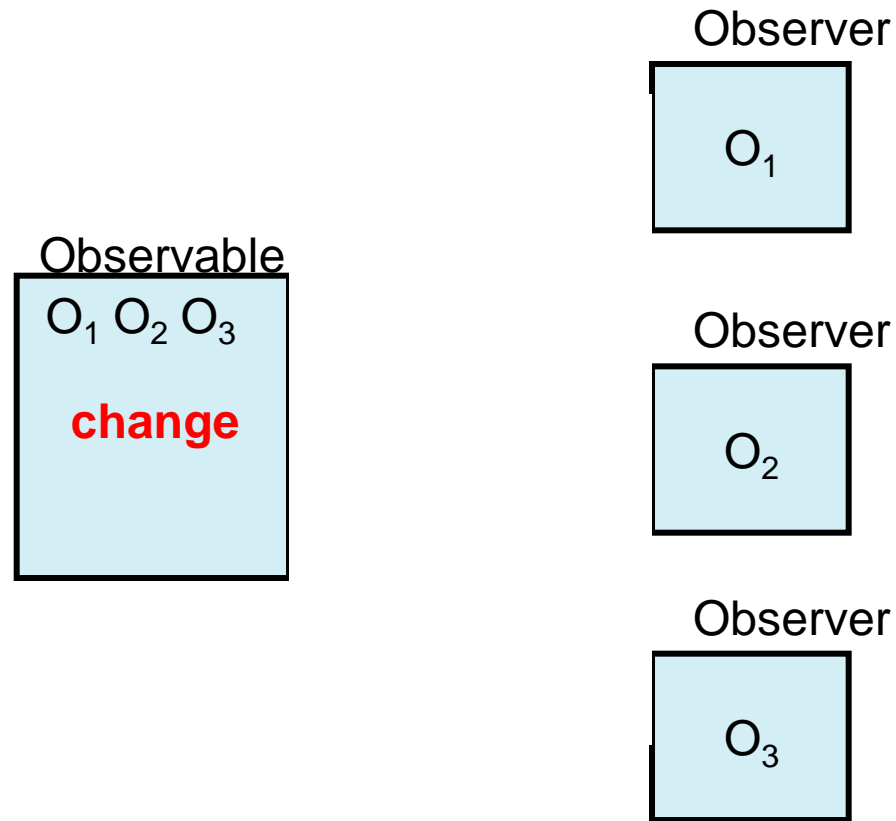
Methods

Modifier and Type	Method and Description
void	<code>addObserver (Observer o)</code> Adds an observer to the set of observers for this object, provided that it is not the same as some observer already in the set.
protected void	<code>clearChanged ()</code> Indicates that this object has no longer changed, or that it has already notified all of its observers of its most recent change, so that the <code>hasChanged</code> method will now return <code>false</code> .
int	<code>countObservers ()</code> Returns the number of observers of this <code>Observable</code> object.
void	<code>deleteObserver (Observer o)</code> Deletes an observer from the set of observers of this object.
void	<code>deleteObservers ()</code> Clears the observer list so that this object no longer has any observers.
boolean	<code>hasChanged ()</code> Tests if this object has changed.
void	<code>notifyObservers ()</code> If this object has changed, as indicated by the <code>hasChanged</code> method, then notify all of its observers and then call the <code>clearChanged</code> method to indicate that this object has no longer changed.
void	<code>notifyObservers (Object arg)</code> If this object has changed, as indicated by the <code>hasChanged</code> method, then notify all of its observers and then call the <code>clearChanged</code> method to indicate that this object has no longer changed.
protected void	<code>setChanged ()</code> Marks this <code>Observable</code> object as having been changed; the <code>hasChanged</code> method will now return <code>true</code> .

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Observable and Observer



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);  
        }  
    }  
  
    ...  
}
```