תבניות עיצוב (Design Patterns)

- פתרונות כלליים לבעיות עיצוב שחוזרות על עצמן
- מגדירים שפה כללית יותר לדיון על עיצוב התכנית
  - במקום Factory, Singleton, Observer יורשת מהמחלקה A "המחלקה B"
- **ספר:** *Design Patterns: Elements of Reusable Object-Oriented Software*
- מידע רב בנושא קיים ברשת

תוכנה 1

תרגול מספר 11:

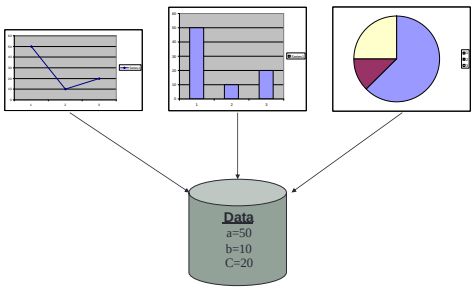
תבניות עיצוב (Observer)
ותרגיל חזרה – חברת הייטק

בית הספר למדעי המחשב
אוניברסיטת תל אביב

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

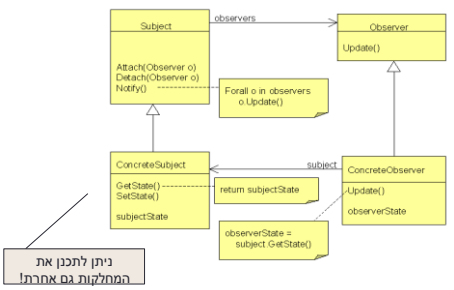
Different Views



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

תבנית העיצוב Observer



Observable

Method Summary

Methods

Modifier and Type	Method and Description
void	addObserver(Observer o) 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	clearChanged() 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 hasChanged method will now return false.
int	countObservers() Returns the number of observers of this Observable object.
void	deleteObserver(Observer o) Deletes an observer from the set of observers of this object.
void	deleteObservers() Clears the observer list so that this object no longer has any observers.
boolean	hasChanged() Tests if this object has changed.
void	notifyObservers() If this object has changed, as indicated by the hasChanged method, then notify all of its observers and then call the clearChanged method to indicate that this object has no longer changed.
void	notifyObservers(Object arg) If this object has changed, as indicated by the hasChanged method, then notify all of its observers and then call the clearChanged method to indicate that this object has no longer changed.
protected void	setChanged() Marks this Observable object as having been changed; the hasChanged method will now return true.

Methods inherited from class java.lang.Object

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

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

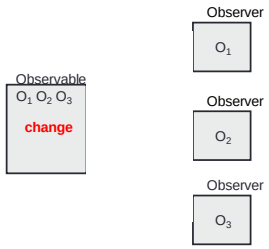
Methods

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

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;
    }
}
```

Observable and Observer



HI-TECH COMPANY

תרגיל חזרה

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);
        }
    }

    ...
}
```

Hi-tech Company

- בתרגיל זה נתרגל מספר נושאים אותם למדנו בשיעורים האחרונים:
  - עיצוב ובניית מודל המורכב ממחלקות לתיאור סביבה מסוימת
  - מנשקים, מחלקות מופשטות וירושה
  - אוספים
- במסגרת התרגיל נכתוב תוכנית לחישוב שכר בחברת הייטק המורכבת ממספר סוגים של עובדים.

עצבו מחלקות לייצוג עובדים בחברה על פי המפרט הבא:

- בחברת הייטק מצליחה ישנם 3 סוגי עובדים: תוכניתנים, בודקי תוכנה ומנהלים.
- לכל עובד יש מזהה מספרי, שם ובוס (מסוג מנהל).
- כל מנהל מחזיק רשימה של עובדים אותם הוא מנהל.
- שכרו של כל מנהל נקבע כמספר העובדים שהוא מנהל ישירות * פקטור אישי.
- תוכניתנים ובודקי תוכנה מקבלים שכר שבועי אישי, כאשר הבודקים מקבלים גם בונוס על כל באג שמצאו השבוע (בונוס אחיד לכל הבודקים פר באג).
- לכל תוכניתן יש שפת תכנות מועדפת עליו שנשמרת בין פרטיו.

הוראות:

- כתבו תוכנית המייצרת אובייקטים של עובדים עם נתונים אקראיים ושומרת אותם בשלוש רמות היררכיות לפי הפירוט הבא:
  - בראש ההיררכיה נמצא המנכ"ל שהינו מנהל
  - מתחתיו בהיררכיה יש 5 מנהלים
  - מתחת לכל מנהל מצויים בהיררכיה 10 תוכניתנים או בודקי תוכנה (בהסתברות שווה).
- לאחר מכן, התוכנית תדפיס את פרטי 3 העובדים עם המשכורת הגבוהה ביותר בכל רמה היררכית.

דוגמא לפלט:

CEO:

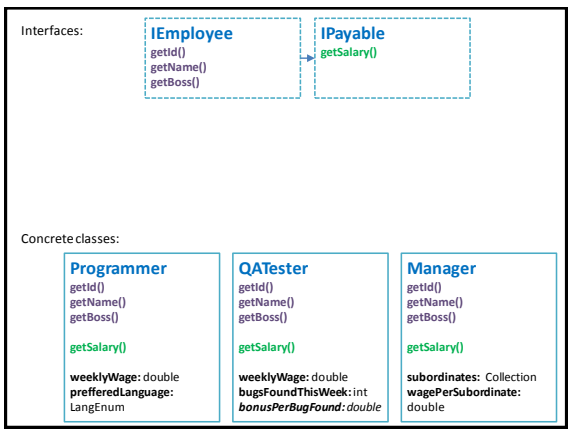
ID: 300001	Name: Moran Cohen	Boss Name: None	Salary: 1245.00	Number of subbordinates: 5	

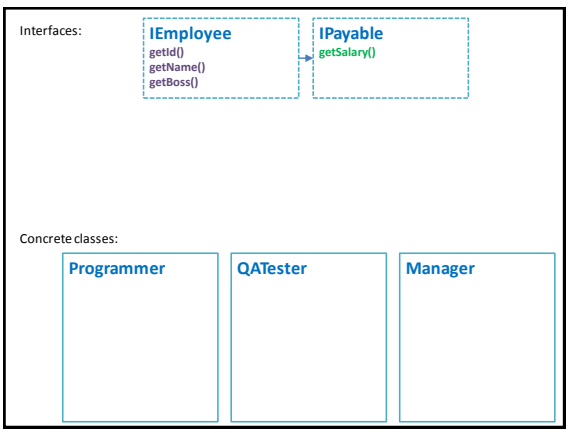
Managers:

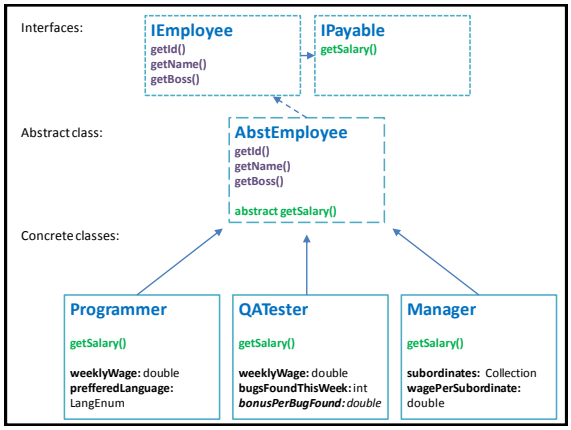
ID: 300024	Name: Kate Cohen	Boss Name: Moran Cohen	Salary: 2260.00	Number of subbordinates: 10	
ID: 300013	Name: Jake Levi	Boss Name: Moran Cohen	Salary: 2250.00	Number of subbordinates: 10	
ID: 300046	Name: Shlomo Jackson	Boss Name: Moran Cohen	Salary: 2250.00	Number of subbordinates: 10	

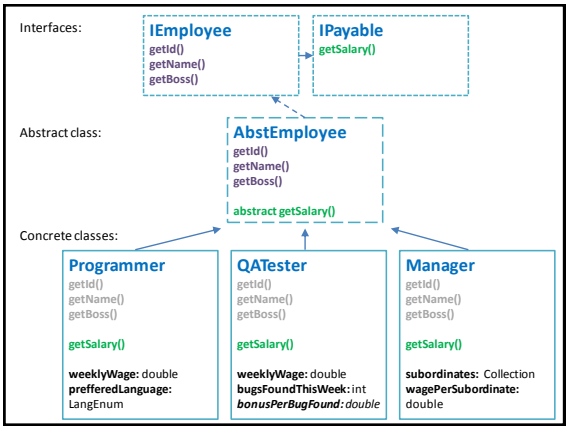
TeamMembers:

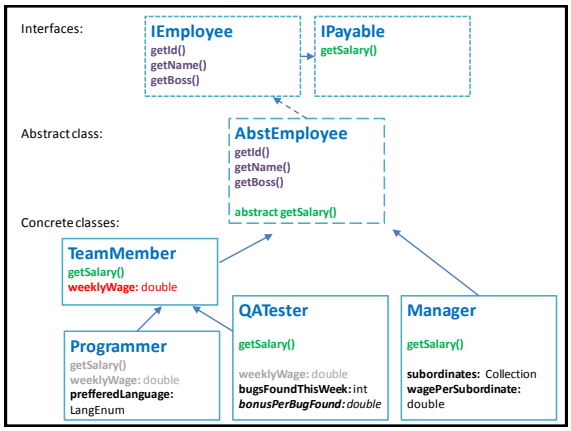
ID: 300006	Name: Abrasha Taylor	Boss Name: Nur Lee	Salary: 713.00	Weekly wage: 673.00	Bugs Found: 8
ID: 300028	Name: Moran Anderson	Boss Name: Kate Cohen	Salary: 711.00	Weekly wage: 696.00	Bugs Found: 3
ID: 300044	Name: David Cohen	Boss Name: Jean-Luc Lee	Salary: 710.00	Weekly wage: 685.00	Bugs Found: 5

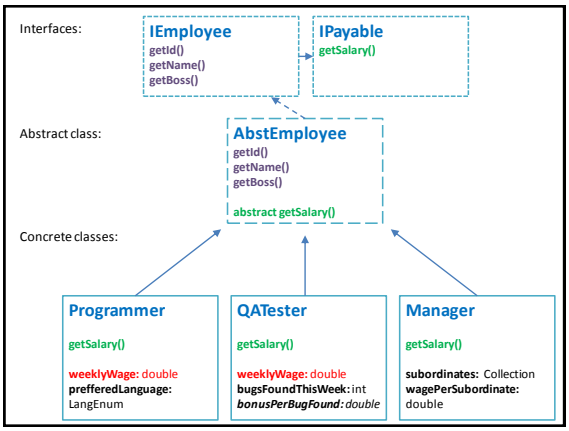












IEmployee

```
/**
 * Interface for accessing basic fields of a company employee
 */
public interface IEmployee extends IPayable {

	public long getId();

	public String getName();

	public IEmployee getBoss();
}
```

IPayable

```
/**
 * Interface supporting getting salary from the company
 */
public interface IPayable {

	public double getSalary();

}
```

AbstEmployee – setBoss

```
/**
 * Sets a new boss for the employee. Removes the employee from the
 * subordinates of the old boss, and adds the employee as a subordinate
 * of the new boss.
 *
 * @param boss
 *            the new boss
 */
public void setBoss(Manager boss) {
   if (getBoss() != null) {
      getBoss().removeSubordinate(this);
   }
   this.boss = boss;
   if (getBoss() != null) {
      getBoss().addSubordinate(this);
   }
}
```

AbstEmployee – Private fields, Constructor and abstract method declaration

```
public abstract class AbstEmployee implements IEmployee {

   private long id;

   private String name;

   private Manager boss;

   public AbstEmployee(long id, String name, Manager boss) {
      super();
      this.id = id;
      this.name = name;
      this.boss = boss;
   }

   public abstract double getSalary();
```

AbstEmployee – toString() method

```
public String toString() {
   return String.format(
         "ID: %10d\t\tName: %s\t\tBoss Name: %s\t\tSalary: %.2f",
         getId(), getName(), getBoss() != null ?
               getBoss().getName() : "None", getSalary());
   }
```

AbstEmployee – hashCode() & equals() methods

```
@Override
public int hashCode() {
   final int prime = 31;
   int result = 1;
   result = prime * result + (int) (id ^ (id >>> 32));
   return result;
}

@Override
public boolean equals(Object obj) {
   if (this == obj)
      return true;
   if (obj == null)
      return false;
   if (getClass() != obj.getClass())
      return false;
   IEmployee other = (IEmployee) obj;
   if (getId() != other.getId())
      return false;
   return true;
}
```

Manager

```
   @Override
   public double getSalary() {
      return this.subordinates.size() * this.wagePerSubordinate;
   }

   @Override
   public String toString() {
      return super.toString()
            + String.format("\tNumber of subbordinates: %s",
                  this.subordinates.size());
   }
```

Manager

```
public class Manager extends AbstEmployee {

   private final static double DEFAULT_WAGE_PER_SUBORDINATE = 50;

   private Set<IEmployee> subordinates;

   private double wagePerSubordinate;

   public Manager(long id, String name, Manager boss, double
                     wagePerSubordinate) {
      super(id, name, boss);
      this.wagePerSubordinate = wagePerSubordinate;
      this.subordinates = new HashSet<IEmployee>();
   }

   public Manager(long id, String name, Manager boss) {
      this(id, name, boss, DEFAULT_WAGE_PER_SUBORDINATE);
   }
```

Programmer

```
public class Programmer extends TeamMember {
    public enum ProgrammingLanguage {
        None, C, Java, Pascal, Logo, VisualBasic
    }

    private ProgrammingLanguage prefferedLanguage;

    public Programmer(long id, String name, Manager boss, double weeklyWage,
            ProgrammingLanguage prefferedLanguage) {
        super(id, name, boss, weeklyWage);
        this.prefferedLanguage = prefferedLanguage;
    }

    public Programmer(long id, String name, Manager boss, double weeklyWage) {
        this(id, name, boss, weeklyWage, ProgrammingLanguage.None);
    }

    …
    @Override
    public String toString() {
        return super.toString()
                + String.format("\tPreffered language: %s",
                                this.prefferedLanguage);
    }

}
```

TeamMember Class

```
public class TeamMember extends AbstEmployee {

    private double weeklyWage;

    public TeamMember(long id, String name, Manager boss, double weeklyWage) {
        super(id, name, boss);
        this.weeklyWage = weeklyWage;
    }
…
    @Override
    public double getSalary() {
        return weeklyWage;
    }

    @Override
    public String toString() {
        return super.toString()
                    + String.format("\tWeekly wage: %.2f", this.weeklyWage);
    }

}
```

Company – main

```
public class Company {

    private static final int NUM_MANAGERS = 5;
    private static final int NUM_MEMBERS_IN_TEAM = 10;
    private static final double PROPORTION_OF_PROGRAMMERS = 0.5;
    private static final int NUM_TO_PRINT = 3;

    public enum ManagementLevel {
        CEO, Managers, TeamMembers
    }

    public static void main(String[] args) {
        Map<ManagementLevel, List<IEmployee>> employees =
            generateCompanyEmployees();
        printMostPaidEmployeesPerLevel(employees);
    }
…
}
```

QATester

```
public class QATester extends TeamMember {

    public static double bonusPerBugFound = 5; // Same for all QA Testers

    private int bugsFoundThisWeek;

    public QATester(long id, String name, Manager boss, double weeklyWage) {
        super(id, name, boss, weeklyWage);
        resetBugsFound();
    }

    …

    /**
     * The salary of a QA tester is composed of the weekly wage plus the bonus
     * on each bug found.
     */
    @Override
    public double getSalary() {
        return super.getSalary() + bugsFoundThisWeek * bonusPerBugFound;
    }

    @Override
    public String toString() {
        return super.toString()
                    + String.format("\tBugs Found: %d", this.bugsFoundThisWeek);

    }
}
```

Company – `generateTeamMembers()`

```
private static void generateTeamMembers(Random rand,
        Map<ManagementLevel, List<IEmployee>> employees, Manager manager) {
    for (int j = 0; j < NUM_MEMBERS_IN_TEAM; j++) {
        float r = rand.nextFloat();
        TeamMember newTeamMember;
        if (r < PROPORTION_OF_PROGRAMMERS)
            newTeamMember = EmployeeUtils.generateRandomProgrammer();
        else
            newTeamMember = EmployeeUtils.generateRandomQATester();

        newTeamMember.setBoss(manager);
        employees.get(ManagementLevel.TeamMembers).add(newTeamMember);
    }
}
```

Company – `generateCompanyEmployees()`

```
private static Map<ManagementLevel, List<IEmployee>>
        generateCompanyEmployees() {

    Random rand = new Random();
    Map<ManagementLevel, List<IEmployee>> employees = new
        TreeMap<ManagementLevel, List<IEmployee>>();

    for (ManagementLevel level : ManagementLevel.values())
        employees.put(level, new LinkedList<IEmployee>());

    Manager ceo = EmployeeUtils.generateRandomManager();
    employees.get(ManagementLevel.CEO).add(ceo);

    generateManagersAndTeams(rand, employees, ceo);
    return employees;
}

private static void generateManagersAndTeams(Random rand,
        Map<ManagementLevel, List<IEmployee>> employees, Manager ceo) {
    for (int i = 0; i < NUM_MANAGERS; i++) {
        Manager newManager = EmployeeUtils.generateRandomManager();
        newManager.setBoss(ceo);
        employees.get(ManagementLevel.Managers).add(newManager);

        generateTeamMembers(rand, employees, newManager);
    }
}
```

Company – printTopAtEachLevel()

```
private static void printTopAtEachLevel(
      Map<ManagementLevel, List<IEmployee>> employees) {
   for (Entry<ManagementLevel, List<IEmployee>> entry : employees
         .entrySet()) {
      ManagementLevel level = entry.getKey();
      List<IEmployee> employeesAtCurrentLevel = entry.getValue();
      int numEmployeesToPrint = Math.min(NUM_TO_PRINT,
            employeesAtCurrentLevel.size());

      System.out.println(level.toString() + ":");

      for (int i = 0; i < numEmployeesToPrint; i++) {
         System.out.println(employeesAtCurrentLevel.get(i));
      }
   }
}
```

Company – printMostPaidEmployeesPerLevel()

```
private static void printMostPaidEmployeesPerLevel(
      Map<ManagementLevel, List<IEmployee>> employees) {

   sortBySalary(employees);

   printTopAtEachLevel(employees);
}

private static void sortBySalary(
      Map<ManagementLevel, List<IEmployee>> employees) {
   for (List<IEmployee> employeesAtCurrentLevel : employees.values()) {
      Collections.sort(employeesAtCurrentLevel,
            new Comparator<IPayable>() {
               public int compare(IPayable e1, IPayable e2) {
                     return Double.compare(e2.getSalary(),
                                           e1.getSalary());
               }
            });
   }
}
```

THE END

הקוד נמצא במלואו באתר הקורס