# Java and C# in Depth

Exercise Session – Week 1

# Today's Exercise Session

Organizational information

IDEs

Unit Testing Frameworks

- JUnit
- NUnit

Visitor design pattern

Assignment 1: Calculator Component

# Organizational information

- 2 Assignments (voluntary submission for feedback)
  - RPN Calculator in Java and C#
- Project: 50% of grade
  - A Virtual File System a.k.a DropBox
- Written exam: 50% of grade
  - Last lecture of the semester
- Organization of exercise sessions
  - First weeks discussion of assignments
  - Later discussion of project progress

# IDE

Java

- Eclipse (e.g. IDE for Java Developers)
- IntelliJ-IDEA

C#

- VisualStudio 2010 or 2012 (Win)
- MonoDevelop 3.0.x (Win/Linux/Mac)

or whatever you feel like …

# Unit Testing

JUnit 4

- junit.org
- Integrated into Eclipse
- Use the given test class as example

NUnit

- www.nunit.org
- http://www.go-mono.com/docs/
- Integrated into MonoDevelop
- Decent Integration into Visual Studio by Resharper
- Use the given test class as example

# The Visitor Pattern

A hierarchy of element classes

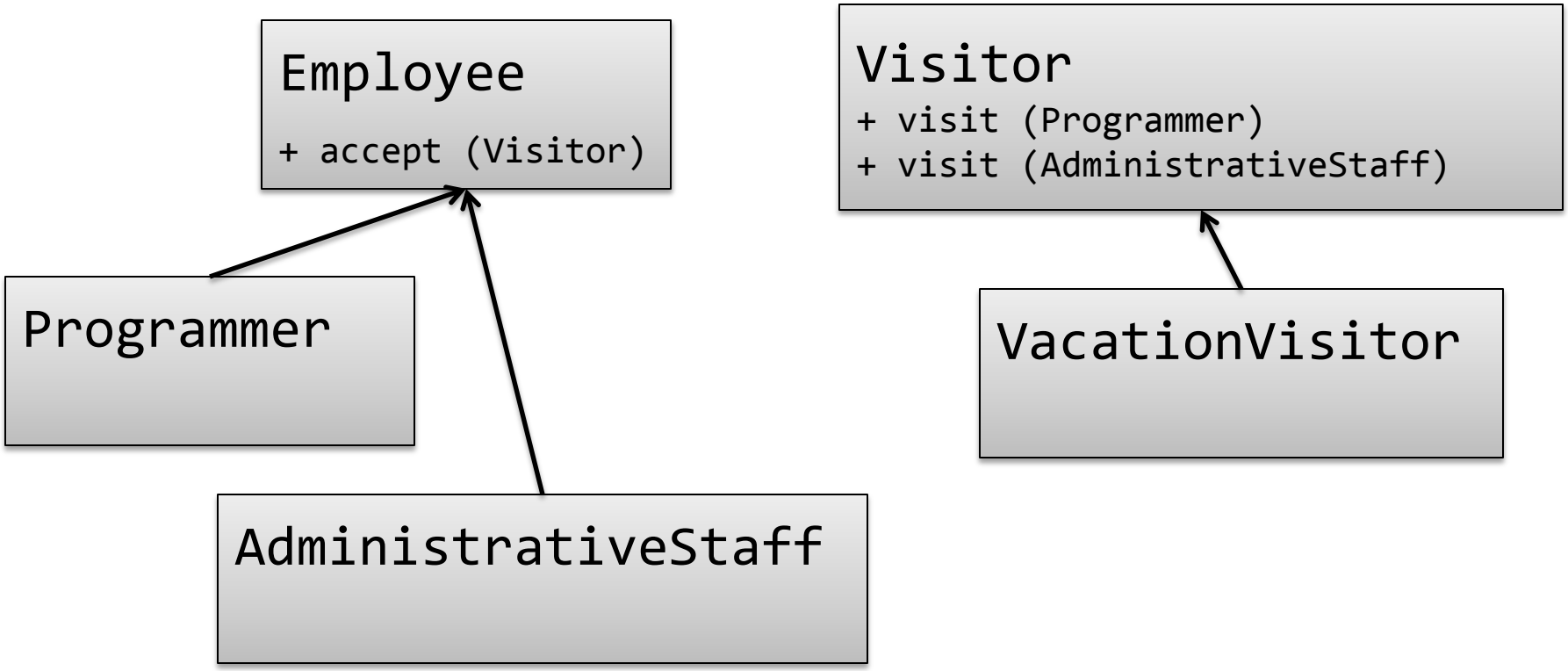
A hierarchy of visitor classes, each performing operations on elements

Uses double dispatching to execute code depending on both element type and visitor type

Commonly used to perform operations on abstract syntax trees

http://en.wikipedia.org/wiki/Visitor_pattern

# Example hierarchy



# The Employee Class

```
public abstract class Employee {
  private int vacDays;
  private BigDecimal salary;
  private String name;

  public Employee(String n, BigDecimal s,
    int vd){
    name = n;
    salary = s;
    vacDays = vd;
  }

  public abstract void accept (Visitor v);
}
```

# The Programmer Class

```
public class Programmer extends Employee {
  private int totalLoc;
  private int projectsCompleted;

  public Programmer(String n, BigDecimal s,
    int vd, int loc, int prj) {
    super(n, s, vd);
    totalLoc = loc;
    projectsCompleted = prj;
  }

  public void accept (Visitor v) {
    v.visit(this);
  }
}
```

# The AdministrativeStaff Class

```
public class AdministrativeStaff extends Employee {
  private int payrollsProcessed;
  private int candidatesInterviewed;

  public AdministrativeStaff(String n,
  BigDecimal s, int vd, int prp, int cand) {
    super(n, s, vd);
    payrollsProcessed = prp;
    candidatesInterviewed = cand;
  }

  public void accept (Visitor v) {
    v.visit(this);
  }
}
```

# The Visitor Interface

```
public interface Visitor {
   void visit(Programmer p);
   void visit(AdministrativeStaff a);
}
```

# The Vacation Visitor

```
public class VacationVisitor implements Visitor {
  private int totalVacDays = 0;

  public void visit(Programmer p){
    totalVacDays +=
      p.getVacDays() +
      p.getProjectsCompleted() * 5;
  }

  public void visit(AdministrativeStaff a) {
    totalVacDays += a.getVacDays();
  }
}
```

# Computing vacation days

```
...
List<Employee> empList = ...
VacationVisitor vv = new VacationVisitor();
for (Employee employee : empList) {
  employee.accept(vv);
}
System.out.println("Total vacation days: " +
  vv.getTotalVacDays());
```

# Another example

Add new visitor to compute productivity of all employees

We measure productivity like this:

- Programmers: loc/tot working days
- Administratives: (payrolls processed + candidates interviewed)/tot working days

# The Productivity Visitor

```
public class ProductivityVisitor implements Visitor{
  private int totalWorkingDays = 365;
  private int totalProductivity = 0;

  public void visit(Programmer p){
    totalProductivity +=
      p.getTotalLoc() / totalWorkingDays;
  }

  public void visit(AdministrativeStaff a) {
    totalProductivity +=
      (a.getPayrollsProcessed() +
       a.getCandidatesInterviewed()) /
                          totalWorkingDays;  }
}
```

# Assessing the Visitor Pattern

Pros

- Adding new operations (visitors) is easy
- Related operations are grouped together

Cons

- Adding new element types is expensive
- As visitor must know about all element classes, it may violate encapsulation

# Assignment 1

- If you want feedback about your solution, you can send it to your assistant - you are not required to hand in your solution

- Java implementation due before 24 February
- C# implementation due before 3 March

- Please use the email subject
  - [JCD]-1-Java-YOUR NAME
  - [JCD]-1-C#-YOUR NAME

# Assignment 1: calculator component

See published PDF.

# Questions?