Introduction to Software Engineering
(2+1 SWS)
Winter Term 2009 / 2010

Dr. Michael Eichberg
Vertretungsprofessur Software Engineering
Department of Computer Science
Technische Universität Darmstadt
Design Heuristics

• Object-Oriented Design Heuristics; J. Riel, Addison-Wesley, 1996
Design Heuristics

• Design Heuristics helps to answer the question: “Is it good, bad, or somewhere in between?”

• Object-Oriented Design Heuristics offers insight into object-oriented design improvement

• The following guidelines are language-independent and allow to rate the integrity of a software design

• Heuristics are not hard and fast rules; they are meant to serve as warning mechanisms which allows the flexibility of ignoring the heuristic as necessary

• Many heuristics are small tweakings on a design and are local in nature

A single violation rarely causes major ramifications on the entire application.
Two areas where the object-oriented paradigm can drive design in dangerous directions...

- ...poorly distributed systems intelligence
  **The God Class Problem**

- ...creation of too many classes for the size of the design problem
  **Proliferation of Classes**
  (Proliferation = dt. starke Vermehrung)
A Very Basic Heuristic

Heuristic

All data in a base class should be private; do not use non-private data.
Define protected accessor methods instead.

If you violate this heuristic your design tends to be more fragile
A Very Basic Heuristic

All data in a base class should be private; do not use non-private data. Define protected accessor methods instead.

```java
public class Line {
    // a "very smart developer" decided:
    // p and v are package visible to enable efficient access
    /*package visible*/ Point p;
    /*package visible*/ Vector v;
    public boolean intersects(Line l) {...}
    public boolean contains(Point p) {...}
}
```

Line l1 = ...;
Line l2 = ...;
// check if both lines are parallel
if (l1.v.equals(l2.v)) {...}

Implementation of a `Line` class as part of a math library.

Some code in the same package that uses `Line` objects.
All data in a base class should be private; do not use non-private data. Define protected accessor methods instead.

```java
public class Line {
    /*package visible*/ Point p1;
    /*package visible*/ Point p2;
    public boolean intersects(Line l) {…}
    public boolean contains(Point p) {…}
}
```

Line l1 = …;
Line l2 = …;
// check if both lines are parallel
if (l1.v.equals(l2.v)) {…}

Now, assume the following change to the implementation of Line.

```
The public interface remains stable - just implementation details are changed.
```

The change breaks our code.
A Very Basic Heuristic

All data in a base class should be private; do not use non-private data. Define protected accessor methods instead.

```java
public class Line {
    private Point p;
    private Vector v;
    public boolean intersects(Line l) {…}
    public boolean contains(Point p) {…}
    protected Vector getVector() { return v; }
}
```

Line l1 = …;
Line l2 = …;
// check if both lines are parallel
if (l1.getVector().equals(l2.getVector())) {…}

Better design.

Some code in the same package that uses Line objects.
Distribute system intelligence as uniformly as possible, that is, the top-level classes in a design should share the work uniformly.

Beware of classes that have too much noncommunicating behavior, that is, methods that operate on a proper subset of the data members of a class. God classes often exhibit much noncommunicating behavior.

Beware of classes that have many accessor methods defined in their public interface. Having many implies that related data and behavior are not kept in one place.
The Problem of Accessor Methods

The class `Point` has accessor operations in the public interface. Are there any problems with this design of `Point`, you can think of?

Is `Point` eventually giving too much implementation details away to clients?
The class `Point` has accessor operations in the public interface. Are there any problems with this design of `Point`, you can think of?

Is `Point` eventually giving too much implementation details away to clients?

The answer to this question is: “No, accessor methods do not necessarily expose implementation details.”
The Problem of Accessor Methods

Accessor methods indicate poor encapsulation of related data and behavior; someone is getting the x- and y-values of Point objects to do something with them – executing behavior that is related to points - that the class Point is not providing.

But, still there is an issue. What is it?

- Often the client that is using accessor methods is a god class capturing centralized control that requires data from the mindless Point object.
The Problem of Accessor Methods

public class Line {
    private Point p;
    private Vector v;
    public boolean intersects(Line l) {…}
    public boolean contains(Point p) {…}
    protected Vector getVector() {return v;};
    public boolean isParallel(Line l) {…};
}

Line l1 = …;
Line l2 = …;
// check if both lines are parallel
if (l1.isParallel(l2)) {…}
Two Reasonable Explanations For the Need of Accessor Methods...

- a class performing the gets and sets is implementing a policy
  \( \text{policy} = \text{dt. Verfahren(-sweise)} \)

- or it is in the interface portion of a system consisting of an object-oriented model and a user interface
Implementing Policies Between Two or More Classes
Example from the Course-scheduling Domain

Captures **static information about students**, e.g., name, identification number, list of courses (s)he has taken, etc.

Captures **static information about the course** objects, e.g., the course number, description, duration, minimum and maximum number of students, list of prerequisites, etc.
Implementing Policies Between Two or More Classes
Example from the Course-scheduling Domain

Captures **static and dynamic information** related to a particular section of a given **course**, e.g., the course being offered, the room and schedule, instructor, list of attendees, etc.
Implementing Policies Between Two or More Classes
Example from the Course-scheduling Domain

First design for checking the prerequisites of students
Implementing Policies Between Two or More Classes
Example from the Course-scheduling Domain

Second design for checking the prerequisites of students
Implementing Policies Between Two or More Classes
Example from the Course-scheduling Domain

Third design for checking the prerequisites of students

The policy is implemented by course offering.
Implementing Policies Between Two or More Classes.
Example from the Course-scheduling Domain

What do you think of these three designs?
(Discuss the pros and cons - regarding the implementation of the policy - with your fellow students.)
In general, always try to model the real world
(Low representational gap facilitates maintenance and evolution.)
But modeling the real world is not as important as the other heuristics.
(E.g., in the real world a room does not exhibit any behavior, but for a heating system it is imaginable to assign the responsibility for heating up or cooling down a room to a corresponding class.)

Basically, a god class is a class that does too much
(Behavioral Form)

By systematically applying the principles that we have studied previously, the creation of god classes becomes less likely
The God Class Problem - Data Form

Summary

Occurs very often when a designer is migrating a legacy system to a new object-oriented design.
Poor Migration to an Object-oriented Systems.

DataStructure is a god class that holds all data.
The God Class Problem - Data Form

Summary

Better Migration to an Object-oriented System.
In applications that consist of an object-oriented model interacting with a UI, the model should never be dependent on the interface. The interface should be dependent on the model.

To which “principle” is this heuristic related?
„I want to add this feature to my system, which 54 classes of the 16,789 classes in my system need to be modified?“

We need to address this question...
Heuristic

Be sure that the abstractions that you model are classes and not simply the roles objects play.
Classes That Model the Roles an Object Plays

<table>
<thead>
<tr>
<th>Variant A</th>
<th>Variant B</th>
</tr>
</thead>
<tbody>
<tr>
<td>class Person {…}</td>
<td>class Person {…}</td>
</tr>
<tr>
<td>class Father extends Person {…}</td>
<td></td>
</tr>
<tr>
<td>class Mother extends Person {…}</td>
<td></td>
</tr>
<tr>
<td>main () {</td>
<td>main () {</td>
</tr>
<tr>
<td>Father f = new Father(…);</td>
<td>Person father</td>
</tr>
<tr>
<td>Mother m = new Mother(…);</td>
<td>= new Person(…);</td>
</tr>
<tr>
<td>}</td>
<td>Person mother</td>
</tr>
<tr>
<td></td>
<td>= new Person(…);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

- Whether to choose Variant A or B depends on the domain you are modeling; i.e. if **Mother** and **Father** exhibit different behavior
- Before creating new classes, be sure the behavior is truly different and that you do not have a situation where each role is using a subset of Person functionality
The Proliferation of Classes

Classes That Model the Roles an Object Plays

What do you think of the following design?

Which question do you have to ask yourself to decide if such a design makes sense?
Irrelevant Classes

Heuristic

Eliminate irrelevant classes from your design.

- An irrelevant class is one that has no meaningful behavior in the domain of your system
- Usually detected by looking for classes that have no operations besides accessors and print functions
- To fix this issue an irrelevant class is typically demoted to an attribute
For many more heuristics….
Object-Oriented Design Heuristics; J. Riel, Addison-Wesley, 1996