DM820
Advanced Topics in Programming Languages

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM820/

ASPECT-ORIENTED PROGRAMMING

The Problem™

- Different concerns should be encapsulated in different parts of the program
- Traditional approach uses classes and modules / packages
- Works well for many concerns:
  - Players, AIs, graphics engine, sound engine, game play
  - Database, business logic, presentation layer
- Does not work well for others:
  - Logging, profiling, …
  - Different concerns using the same data representation
- These concerns are called "cross-cutting concerns"
- Need to find new way to modularize these

The Idea®

- Aspect-Orientation is modularization of cross-cutting concerns
- Lots of funny names for concepts:
  - "Aspects" are cross-cutting concerns
  - Aspects offer "advice" (additional behaviour)
  - "Join points" are specified in "pointcuts"
  - Code of aspects "scattered" (spread out)
  - Aspects can become "tangled" (interacting with each other)
- Main Idea:
  - Keep base source code and aspect source code separate
  - "Weave" bytecode/machine code into compiled base code
  - Alternatively, "weave" source code into base source code

Example: Aspect/J

- Adds aspect-orientation to the Java programming language
- Join points defined by matching Java constructs:
  - execution(* set*(*)))
  - this(Point)
  - within(com.company.*)
  - Can be named:
    pointcut set() : execution(* set*(*)) && this(Point) && within(com.company.*);
- Advice defined using join points:
  - before() : execution(* set*(*))) { Graphics.repaint(); }
  - after() : set() { Model.hasChanged(); }

Hands-On

- Aspect/J tutorial ☺