Runtime Specialization

Java has never been so dynamic before

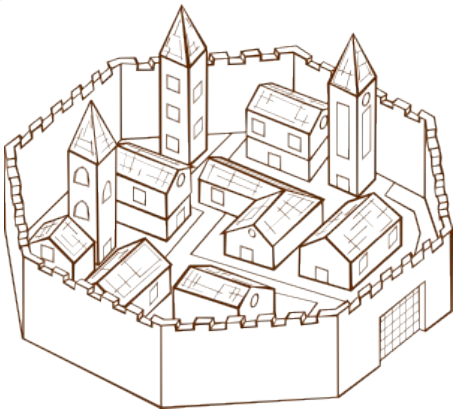
Stephan Herrmann

GK SOFTWARE

Simply Retail.

Two Camps

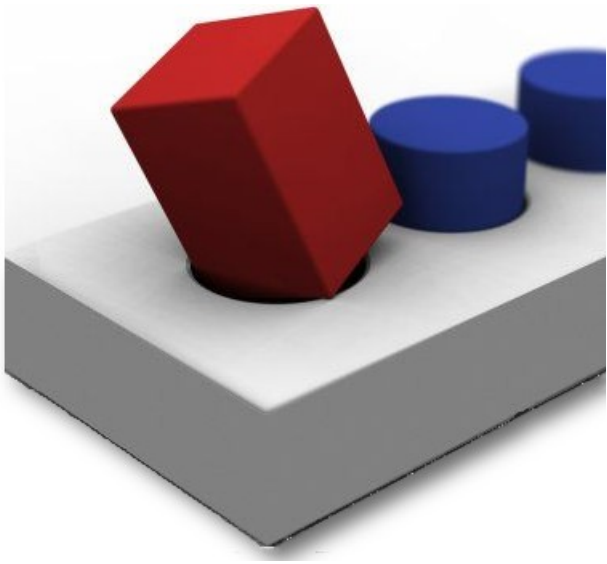
- **No Ceremony**
- **Freedom**
- **Flexibility**
  - self modifying code
  - code modifies language



- **Strict Rules**
  - compiler detects errors
- **Modularity**
  - separate maintenance
- **Enforced Boundaries**
  - blame assignment
  - prevention

Building Blocks

- **Boundary inside vs. outside**
- **What I get**
  - lots of existing building blocks to **choose** from
  - can **compose** them into my application
- **What I don't get**
  - the right to **open** the box to make **changes**
- **Near miss**
  - looks like a good match
  - but doesn't totally fit

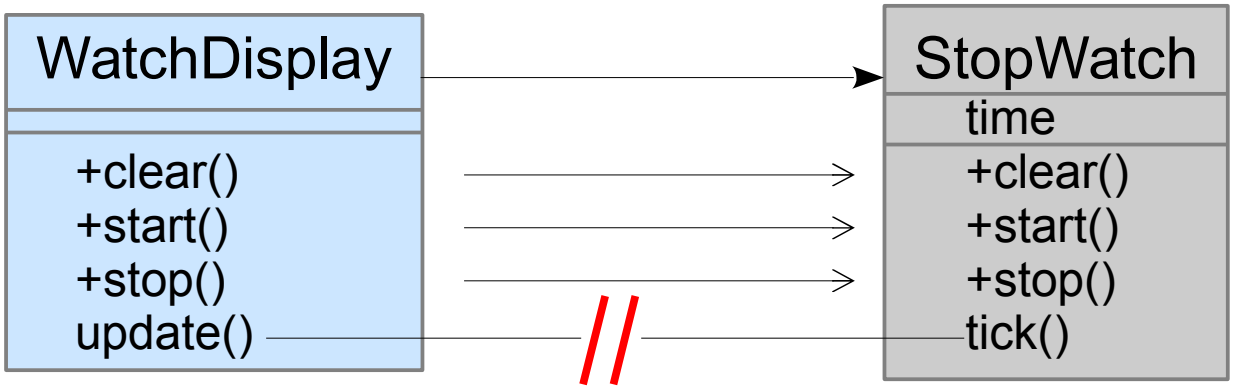


Near Miss

- **Adjust requirements to the building block?**
  - not competitive
- **Drop existing, build your own**
  - expensive
- **Why do we have the problem?**
  - ~~technical impossibility~~
  - rules about boundaries
- **Envy those who don't have these rules**
  - unlimited adaptation
  - unanticipated adaptation

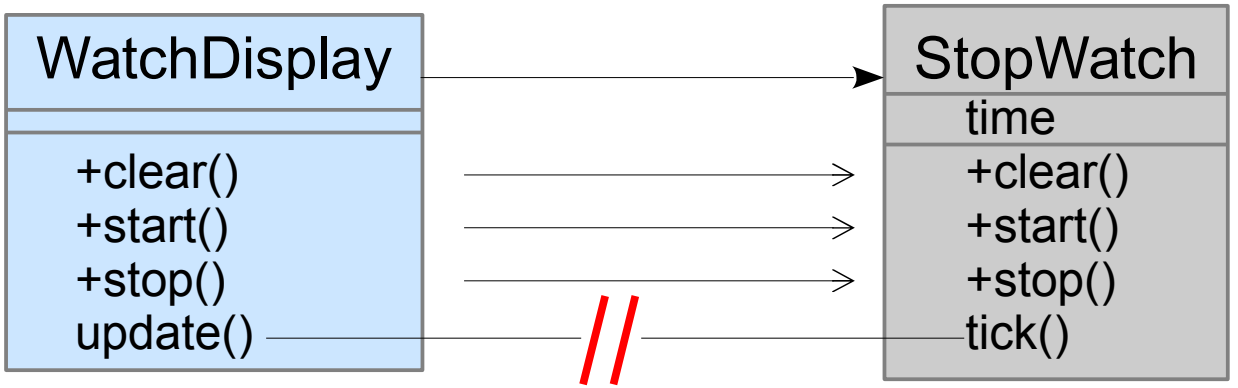
Unanticipated Adaptation

- **Example: Missing Listener Infrastructure**



Unanticipated Adaptation

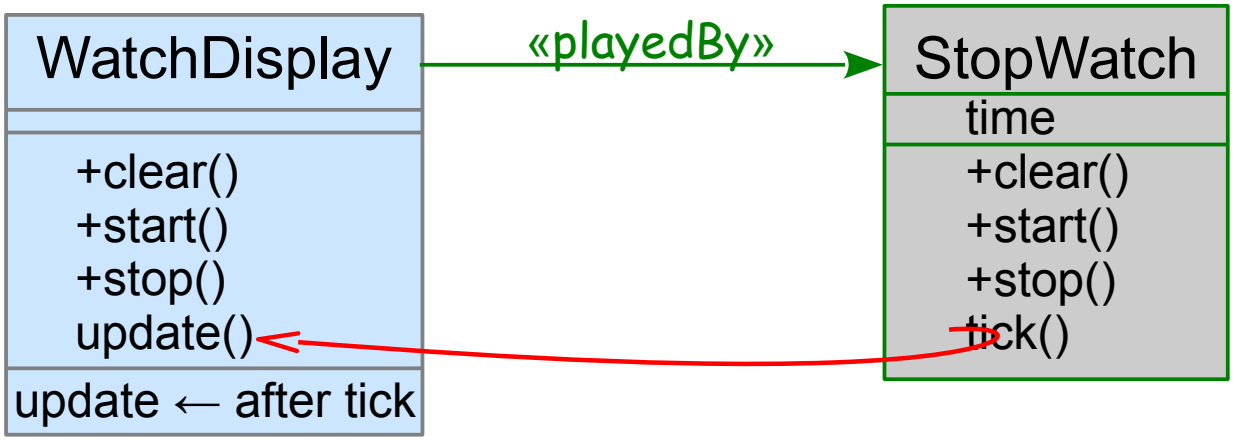
- **Example: Missing Listener Infrastructure**



WANTED:
Method Call Interception

- ○ infra structure solution
- ◉ language solution

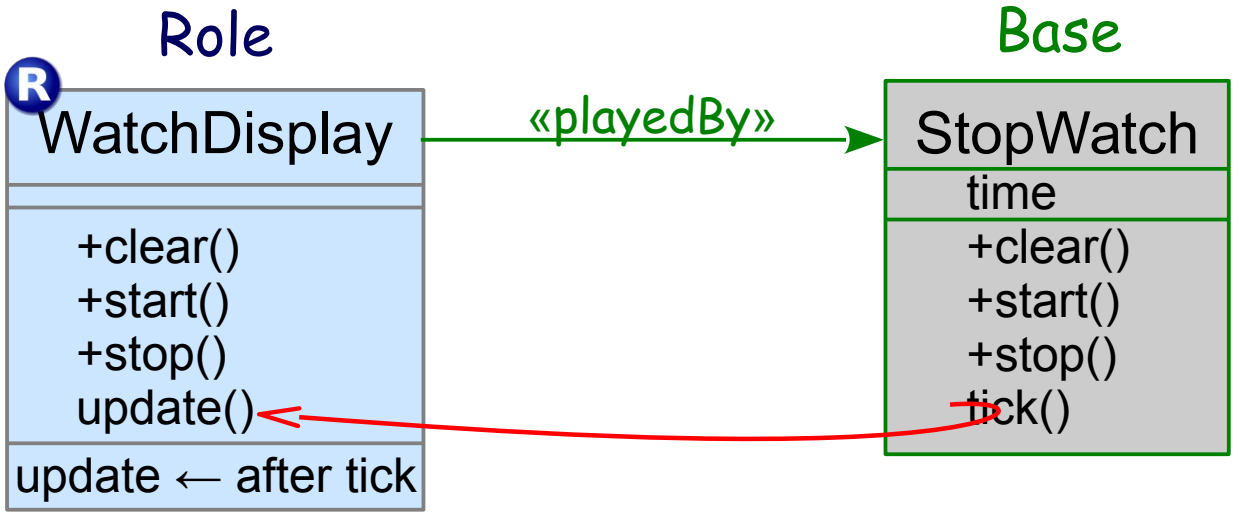
Unanticipated Adaptation



- **Method call interception in Object Teams:**
  - „callin“ method binding
  - flavors: before, after, replace

Demo: Stop Watch

Unanticipated Adaptation

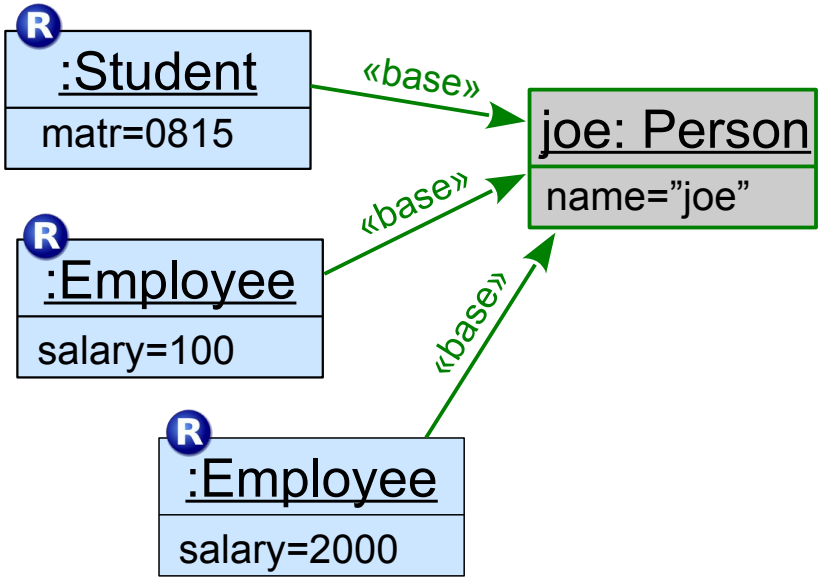


- **Method call interception in Object Teams:**
  - „callin“ method binding
  - flavors: before, after, replace

The Role Playing Metaphor

`playedBy` Relationship

- **Properties**
  - **Dynamism**:
    roles can come and go
    (same base object)
  - **Multiplicities:**
    one base can play several roles
    (different/same role types)



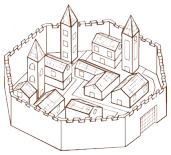
Summary Role Playing

- **playedBy**
  - similar to inheritance
  - connects instances
- **callin**
  - method call interception
- **callout**
  - method forwarding
  - regardless of visibility

```
protected class WatchDisplay
    extends JFrame
    playedBy StopWatch {

    update <- after advance;

    void start() -> void start();
    void stop()  -> void stop();
    void clear() -> void reset();

}
```

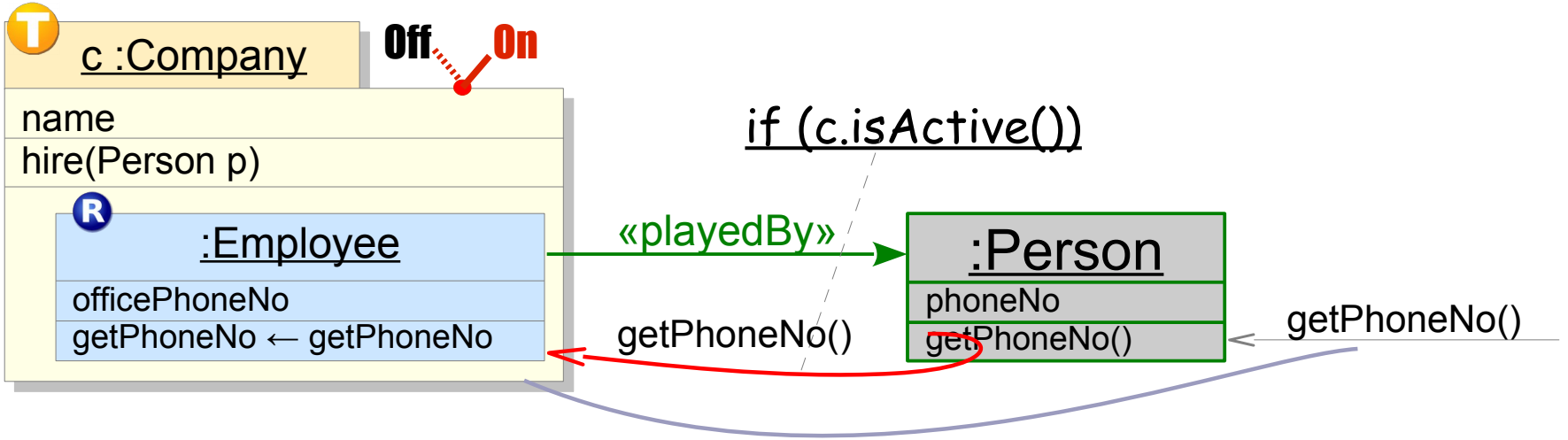
Good or Bad?

- **Moralists**
  - City: boundaries must not be violated!
  - Fire site: boundaries are bad!



- **With power comes responsibility**
- **Give developers all the means necessary**
  - For controlling roles
  - In a modular way

Controlling Roles



- **Roles** depend on context
  - contexts are reified as **Teams**
- Each team instance can be (de)activated
  - active team instances contribute to the **system state**
  - dispatch considers system state

Controlling Roles (2)

- **Globally active teams**
  - part of the top-level system definition
    - plain application: -Dot.teamconfig=...
    - OT/Equinox: extension point aspectBinding
- **Scope of activation**
  - per-thread
  - ALL_THREADS
  - while a certain code block is executing
  - ...

Demo: Flight Booking

Integration – Technically

- **Weave hooks into the byte code**
  - build time
  - load time
  - runtime
- **Execute on a standard JVM**
  - application
    - -javaagent:
  - OT/Equinox
    - WeavingHook (OSGi standard)

Phases

- **Development of building blocks**
  - variability model
  - extension points
- **Composition to an application**
  - wire & configure building blocks
  - **unanticipated adaptation**

New in OT/J

- **Deployment**
  - more wiring & configuration
- **Operation**
  - **runtime adaptation**

New in OT/J Neon

→ http://www.eclipse.org/objectteams