Just-In-Time Patching

Java Applications

Stefan Mandel, Jan Müller

Szenario

- Geschafft – wir sind live
- … und plötzlich tritt ein kritischer Fehler auf
  - Beim Testen war der noch nicht da
  - Der Fehler tritt nur manchmal auf
  - Und die Ursache ist völlig unklar
- Wir können das System nicht unterbrechungsfrei neu starten
- Was jetzt?

Post-Mortem-Analyse

- Debugging
- Logs
- Codeanalyse


```
Starting weather application
...
Weather application started
```

leer

```
SimulatedWeatherSource.java
44      }
45
46      public static Precipitation[] precipitationForSeason(Season season) {
47          switch (season) {
48          case SPRING:
49              return new Precipitation[] { DRY, DRY, DRY, DRY, MIST, MIST, MIST, DRIZZLE, DRIZ
50          case SUMMER:
51              return new Precipitation[] { DRY, DRY, DRY, DRY, DRY, DRY, DRY, DRY, DRIZZLE, RA
52          case FALL:
53              return new Precipitation[] { DRY, DRY, DRY, MIST, MIST, MIST, DRIZZLE, DRIZZLE,
54          case WINTER:
55          default:
56              return new Precipitation[] { DRY, DRY, DRY, MIST, MIST, DRIZZLE, RAIN, SNOW, SNO
57          }
58      }
59
60      public static Temperature[] temperatureForSeason(Season season) {
61          switch (season) {
62          case SPRING:
63              return new Temperature[] { WARM, MODERATE, COOL, COLD };
64          case SUMMER:
65              return new Temperature[] { HOT, WARM, MODERATE, COOL };
66          case FALL:
67              return new Temperature[] { WARM, MODERATE, COOL, COLD };
68          case WINTER:
69          default:
```

Legacy

Probleme

- Der Fehler
  - ist live
  - ist nicht lokalisierbar
  - ist nicht reproduzierbar
- Für die Lösung dieser Probleme müssen wir das Programm ändern, ggf. mehrfach
  - erfordert jeweils einen Systemneustart
- Jede Änderung
  - stört aktive Benutzer
  - kostet den Auftraggeber Geld und/oder Image
- Aber der Status Quo … ebenfalls

Das wäre nicht passiert / nicht so schlimm ...

- wenn wir konsequent loggen würden
- wenn wir bessere Tests geschrieben hätten
- wenn wir debuggen könnten
- wenn wir unterbrechungsfrei deployen könnten

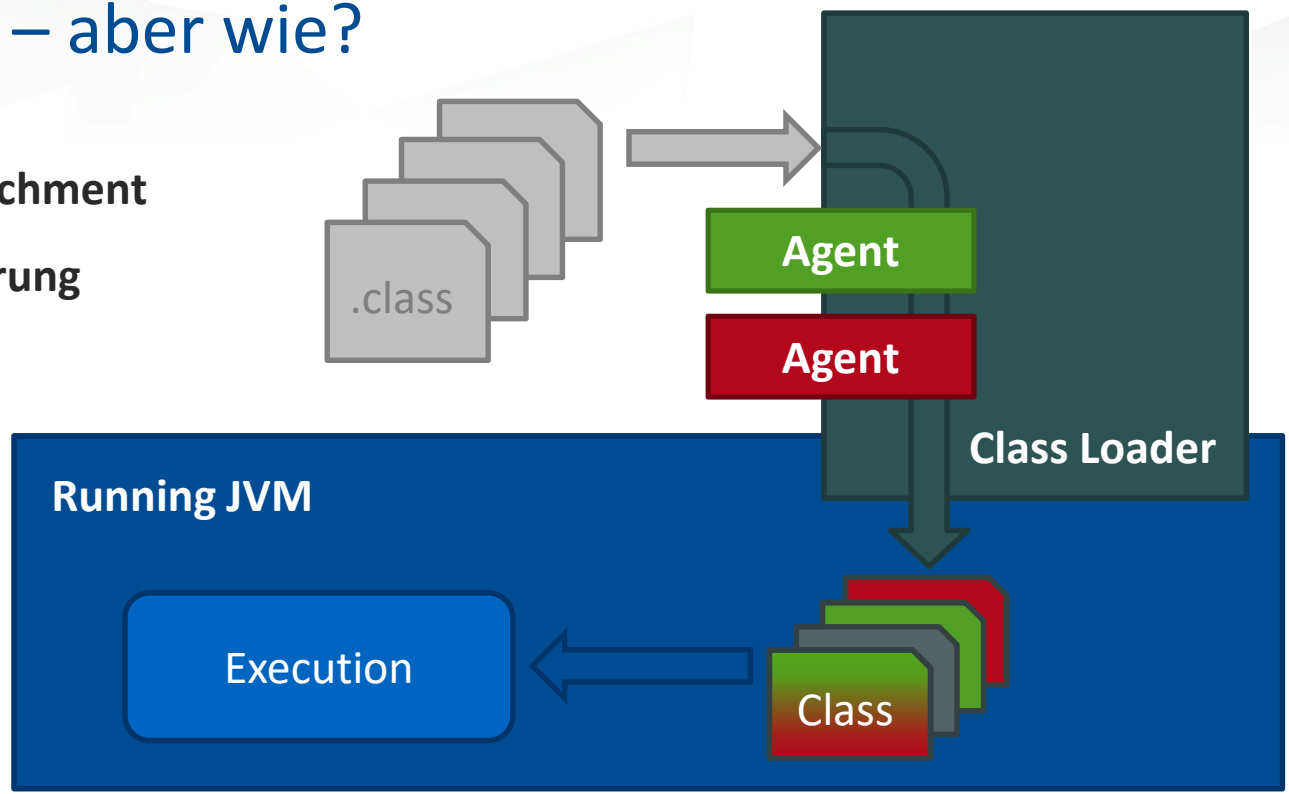
… oder es nachträglich ergänzen könnten

- Nachträglich Logging einfügen?
- Nachträglich Tests schreiben, um den Fehler zu isolieren?
- Anhand der Tests debuggen und den Fehler finden?
- Den korrigierten Code nachträglich einspielen?

Das geht – aber wie?

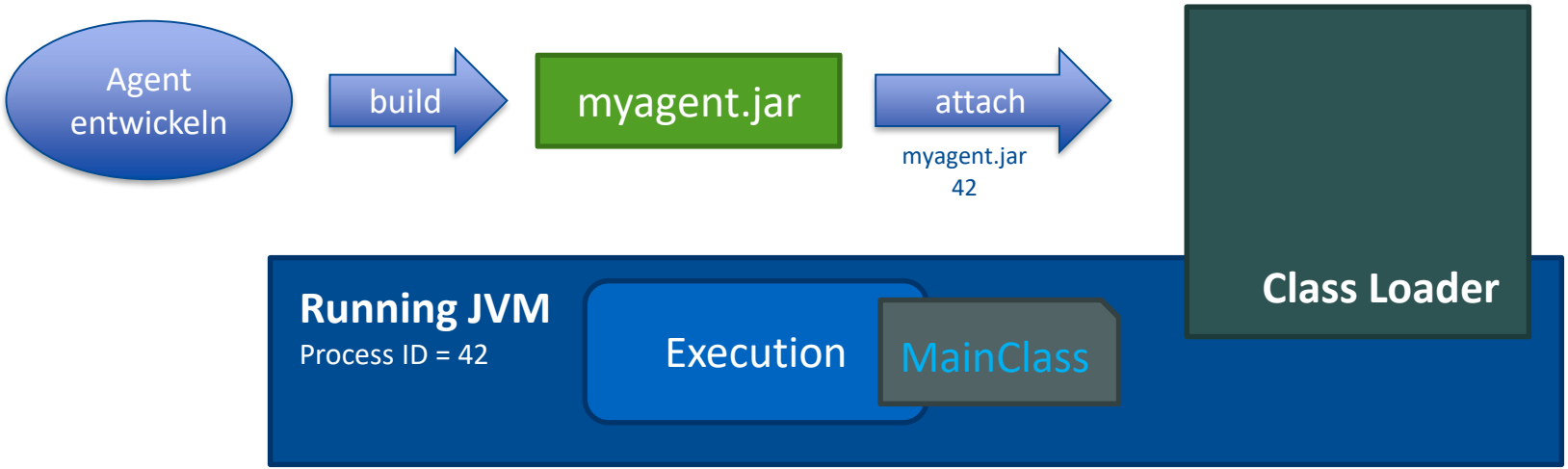
Runtime Attachment

Instrumentierung



Das geht – aber wie?

```
java […].RemoteAgentAttacher "myagent.jar" "MainClass" "attach"
```



Das geht – aber wie?

```
java […].RemoteAgentAttacher "myagent.jar" "MainClass" "attach"
```

```
public static void main(String[] args) throws Exception {
  try {
    File agentJar = Paths.get(args[0]).toFile();
    String processId = processIdForMainClass(args[1]);
    String action = args.length > 2 ? args[2] : "attach";

    System.out.println("Connecting for " + action + ": " + processId);
    ByteBuddyAgent.attach(agentJar, processId, action);
    System.out.println("Disconnecting from " + action + ": " + processId);
  } catch (...) {...}
}

private static String processIdForMainClass(String mainClass) throws Exception {...}
```


Demo – Agenda

Code: https://github.com/andrena/just-in-time-patching

1. Anwendung mit Bug
2. Fehlende Fehler-Logs einfügen (LogAgent)
3. Anwendungsverlauf aufzeichnen (RecordingAgent)
4. Aufzeichnung analysieren und Patch entwickeln
5. Patch einspielen (PatchAgent)

Was bieten uns Runtime-Agents?

- Unterbrechungsfreies Inspect & Adapt
  - Keine Beeinträchtigung der Benutzer (unterbrechungsfrei)
  - Erheben diagnostischer Informationen (Inspect)
  - Korrigieren fehlerhaften Codes (Adapt)
- Unabhängig von der Architektur
  - Kein Konzept für unterbrechungsfreies Ausrollen nötig
- Unabhängig von der Programmierung
  - Funktioniert auch mit Legacy-Code

Und der Haken?

- Problem
  - Patches entstehen als Deltas
  - Deltas sind zudem noch dynamisch
  - Dadurch wenig Überblick über den laufenden Code
- Alternative:
  - Architektur unterstützt unterbrechungsfreies Ausrollen
    - Systemneustarts werden günstig
  - Clean Code
    - Fehlersuche geht schneller

Was nehmen wir mit?

- Instrumentierung erlaubt Klassen um Aspekte zu erweitern, z.B. in
  - Code Coverage in der IDE
  - AspectJ
  - Project Lombok
  - Spring
  - Testrecorder
- Runtime Attachment erlaubt Instrumentierung zur Laufzeit, z.B. für
  - Hot-Swap in der IDE
  - Profiling
  - Logging
  - Fehlerkorrektur
- Viel Spaß

Code: https://github.com/andrena/just-in-time-patching

Vielen Dank!