12.–15.09.2010
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Aber sicher!
Strategien für Typ-System-Erweiterungen

Michael Wiedeking

MATHEMA Software GmbH

Typ-Annotationen in Java

- Eine Typ-Annotation kann vor jeden Typ geschrieben werden

  @NonNull String

- Das Objekt einer Methode (*this*) wird unmittelbar nach der Parameter-Liste annotiert

  void print() @ReadOnly { … }

Positionen von Typ-Annotationen

List<@NonNull String> *strings*;

myGraph = (@Immutable Graph) *tmpGraph*;

class UnmodifiableList<*T*>
  implements @Readonly List<@Readonly *T*> { ... }

Typ-Annotationen und Flussanalyse

```
@Nullable Integer jsr;

…

// @NonNull Integer valueOf(String);

jsr = Integer.valueOf("308");

… jsr.toString() … // Keine Null-Dereferenz-Warnung
```

Problematik bei generischen Typen

```
@PolymorphicQualifier
public @interface PolyNull { }
```

```
@PolyNull T cast(@PolyNull Object obj)
```

Internalisierte Strings

String *s*;
@Interned String *is*;

if (*s* == *is*) { … } // Warnung: Unsicherer Vergleich!

Sicheres Casten

@Readonly Object *x*;

… (@Readonly Date) *x* …

@Readonly Object *x*;

… (Date) *x* … // Warnung!

Sicheres Casten

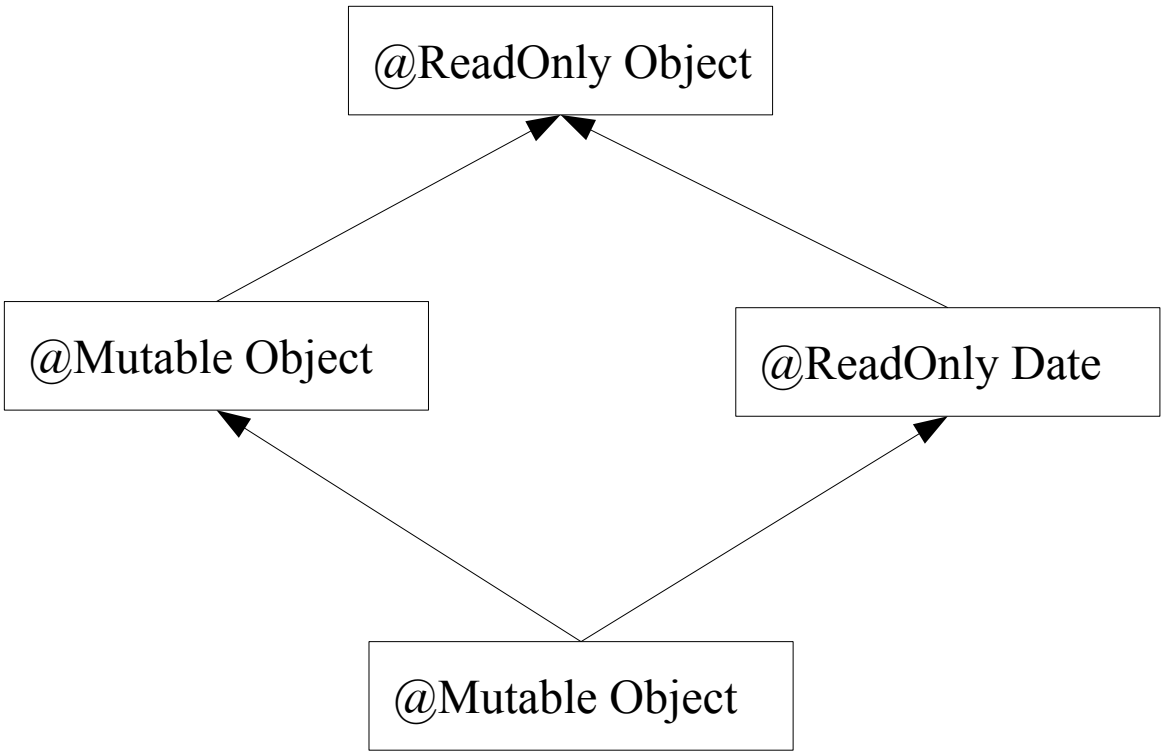
final Object *x* = **new** Object();

… (@NonNull Object) *x* …

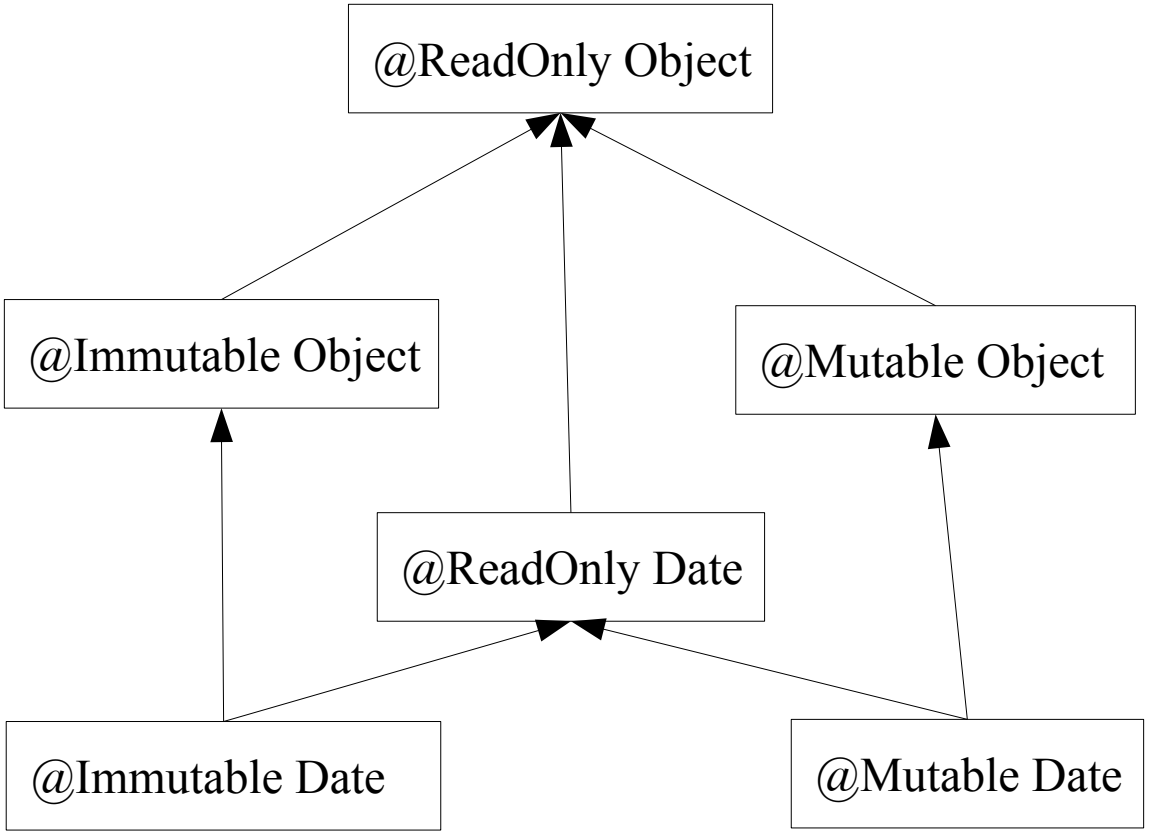
Abfragen von Typen

```
@Readonly Object x;

if (x instanceof Date) …
                                        // Fehler: inkompatibel
if (x instanceof @Readonly Date) ...
                                        // OK


Object y;

if (y instanceof Date) …
                                        // OK
if (y instanceof @NonNull Date) …
                                        // Fehler: inkompatibel
```

Typ-Hierarchie (1/2)



Typ-Hierarchie (2/2)



Kompliziertere Typen

```
final class MyStringMap
    extends
        @Readonly Map<
            @NonNull String,
            @NonEmpty List<
                @NonNull @Readonly String
            >
        >
{
    …
}
```

Was nicht funktioniert

```
class ArrayList<E> {

    public
    @PolyNull Object[]
        toArray() ArrayList<@PolyNull T> { … }

    public
    Object[]
        toArray() @Readonly { … }

}
```

Was auch nicht funktioniert

```
interface List<T> {

    // size ändert weder this noch eines der Elemente
    public int size() @Readonly List<@Readonly T> { ... }

}
```

Funktioniert leider auch nicht

```
class MyMap<T, U> {

   // Werte sind nicht-null, Schlüssel sind beliebig
   public
   void requiresNonNullKeys()
        MyMap<T, @NonNull U> { … }

}
```

Alternative (1/2)

```
public class Patient {

    private int patientId;

    @RolesAllowed({"Doctor", "Patient"})
    public static Patient getPatient(int pid) { … }

    @RolesAllowed({"Doctor", "Patient"})
    public List<String> getHistory() { … }

    @RolesAllowed({"Doctor"})
    public void addPrescription(String prescription) { … }
    …
}
```

Alternative (2/2)

```
public class PatientServlet {

    void displayHistory(int pid, Request rq, Response rsp) {
        if (rq.isUserInRole("Patient")) {
            if (rq.userId != pid) {
                throw new AccessError("Cannot access");
            }
        }
        Patient p = Patient.getPatient(pid);
        List<String> hist = p.getHistory();
        …
    }
}
```

Typisierte Methoden (1/2)

```
public class Patient {

    @RoleParam public final int patientId;

    @Requires(
-       roles={"DoctorOf", "Patient"},
        params={"pid", "pid"})
    @Returns(
        roleparams="patientId",
        vals="pid")
    public static
    Patient getPatient(@RoleParam final int pid) { … }

    …
```

Typisierte Methoden (2/2)

```
…

@Requires(
    roles={"DoctorOf", "Patient"},
    params={"this.patientId", "this.patientId"})
public List<String> getHistory() { ... }

@Requires(
    roles="DoctorOf",
    params="this.patientId")
public void addPrescription(String prescription) { … }

…
```

Beispiel 2 (1/3)

```
public class PatientServlet {

    @Requires(
        roles={"DoctorOf", "Patient"},
        params={"pid", "pid"})
    public void displayHistory(
        @RoleParam final int pid,
        Request req, Response resp
    ) {
        Patient p = Patient.getPatient(pid);
        List<String> hist = p.getHistory();
        …
```

Beispiel 2 (2/3)

```
public class Request {

    @RolePredicate(roles="Patient", params="pid")
    public
    boolean hasPatientRole(@RoleParam final int pid) {
        …
    }

    @RolePredicate(roles="DoctorOf", params="pid")
    public
    boolean hasDoctorOfRole(@RoleParam final int pid) {
        …
    }

}
```

Beispiel 2 (3/3)

```
public class PatientServlet {
   void displayHistory(
        @RoleParam final int pid, Request rq, Response rsp
   ) {
        if (!
             (rq.hasPatientRole(pid)
             ||
             rq.hasDoctorOfRole(pid))
        )) {
             throw AccessError("Cannot access this patient");
        }
        Patient p = Patient.getPatient(pid);
        List<String> hist = p.getHistory();
        …
```

Fragen?

Vielen Dank!

michael.wiedeking@mathema.de