Technik: Annotations

- **Definition einer Annotation analog eines Interfaces**

```
public @interface ClassInfo {
    String author();
    int revision() default 1;
    String[] reviewers() default {};
}
```

- **@interface (statt interface)**
- **Default-Werte (alle anderen müssen angegeben werden)**

Technik: Annotations

- **Anwendung**

```
@ClassInfo(
    author = "Jens",
    revision = 2,
    reviewers = { "Hubert", "Steffi" }
)
public class TestReflect {
    …
}
```

Technik: Annotations

RWTH AACHEN UNIVERSITY

- **Meta-Annotations**
  - → @Retention (Gibt an, wieweit die Informationen aufgehoben werden)
    - →RetentionPolicy.Source (Nur im Source)
    - →RetentionPolicy.Class (Im Class-File, aber nicht zur Laufzeit)
    - →RetentionPolicy.Runtime (Zur Laufzeit per Reflection auslesbar)
  - → @Target (Gibt an, auf welche Elemente die Annotation angewendet werden kann, mehrere sind möglich)
    - →ElementType.TYPE (Klassen, Interfaces, etc.)
    - →ElementType.METHOD (Methoden)
    - →Etc.
  - → @Documented (Annotation erscheint in JavaDoc der Klasse)

Technik: Annotations

- **Beispiel von vorhin für Klassen und zur Laufzeit per Reflection auslesbar:**

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface ClassInfo {
    String author();
    int revision() default 1;
    String[] reviewers() default {};
}
```

Technik: Annotations

- **Aufgabe:**
  - → Eigene Annotation für eine Klasse schreiben
  - → In der Reflection-Tool-Klasse eine neue Methode einführen: printAnnotations, die die Annotations einer Klasse ausgibt. Für die eigene Annotation auch die angegebenen Werte ausgeben

    Beispiel mit eigener Annotation @ClassInfo und @Deprecated:

```
Alle Annotations von de.rz.reflect.TestReflect
->Deprecated
->ClassInfo.author=Jens
->ClassInfo.revision=2
->ClassInfo.reviewers:Hubert Steffi
```

Technik: Serialisierung

- **Implementieren des Marker-Interfaces `Serializable` reicht, wenn alle Daten der Klasse ebenfalls `Serializable` (oder primitive Datentypen) sind**
- **Statische Daten werden nicht serialisiert**
- **Mit `transient` kann man Daten aus der Serialisierung heraus nehmen**
- **Default-Serialisierung schreibt recht viel**
- **Anpassung ist notwendig, wenn**
  - → Basisklasse nicht serialisierbar ist
  - → Nicht-serialisierbare Daten referenziert werden

Technik: Serialisierung

RWTH AACHEN UNIVERSITY

- **Anpassung der Serialisierung**
  - → Durch Überschreiben von `writeObject` und `readObject` (Alle Attribute müssen selber behandelt werden)
  - → Benutzen von `defaultWriteObject` und `defaultReadObject` in `writeObject` und `readObject` (schreibt/liest alle nicht-transient Attribute)
  - → Für Platz-Optimierung definiert man das betreffende Attribut als transient und schreibt eine platzsparende Variante in `writeObject` und liest sie entsprechend in `readObject` wieder ein.

Technik: Serialisierung

- **Zur Identifizierung dient der Klassenname und eine Versionskennung**
  - → Gibt man nichts vor, wird eine Versionskennung beim Kompilieren errechnet: Hier gehen unter anderem die Methoden und der Zeitpunkt der Kompilierung ein! D.h.: Hinzufügen einer Methode ändert die Versionskennung → eine „alte“ Deserialisierung geht schief.
  - → Abhilfe: Manuelles Angeben einer serialVersionUID, die nur geändert wird, wenn sich die Daten für die Serialisierung ändern
  - → Zusätzlich kann eine eigene Versionierung nötig sein. Auf die eigene Version muss man beim Schreiben/Lesen entsprechend reagieren (Default-Werte beim Lesen neuer Attribute, die in der alten Version nicht gespeichert wurden)

Technik: Serialisierung bei Beans

- **Aufgabe: Ändern der Klasse Position durch Hinzufügen einer Methode (die ausgelieferte Bean unberührt lassen)**
- **Geht jetzt ein Aufruf von greet(„you“,new Position(…)) schief?**
- **Was passiert , wenn man eine `serialVersionUID` einführt?**
- **Versuche dasselbe mit Speichern in und Lesen aus einer Datei**

Technik: Serialisierung bei Beans

- **Aufgabe: Die Klassen:**

```
class Person {
    Position pos;
    void setPos(Position p);
}
```

```
class Position {
    int x;
    void setX(int x);
}
```

- **Methode in einer Bean:**

```
void doIt( Person per, Position pos ) {
    pos.setX( 12 );
    per.setPos( pos )
}
```

- **Was kommt davon auf dem Client an?**

Technik: SessionBeans

- **Aufgabe:**
  - → Auf Client-Seite per Reflection die Klassenhierarchie der per lookup erzeugten Implementation des Hello-Interfaces erfragen.
  - → Auf Server-Seite die Call-Hierarchie der implementierten Methode greet ausgeben.