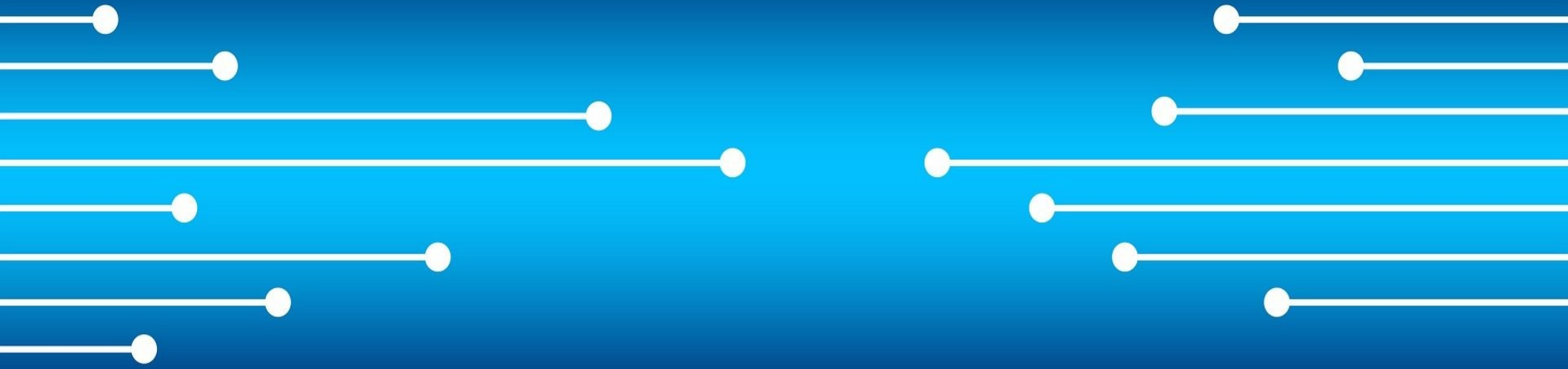
Datenverarbeitung

Teil des Moduls 5CS-DPDL-20



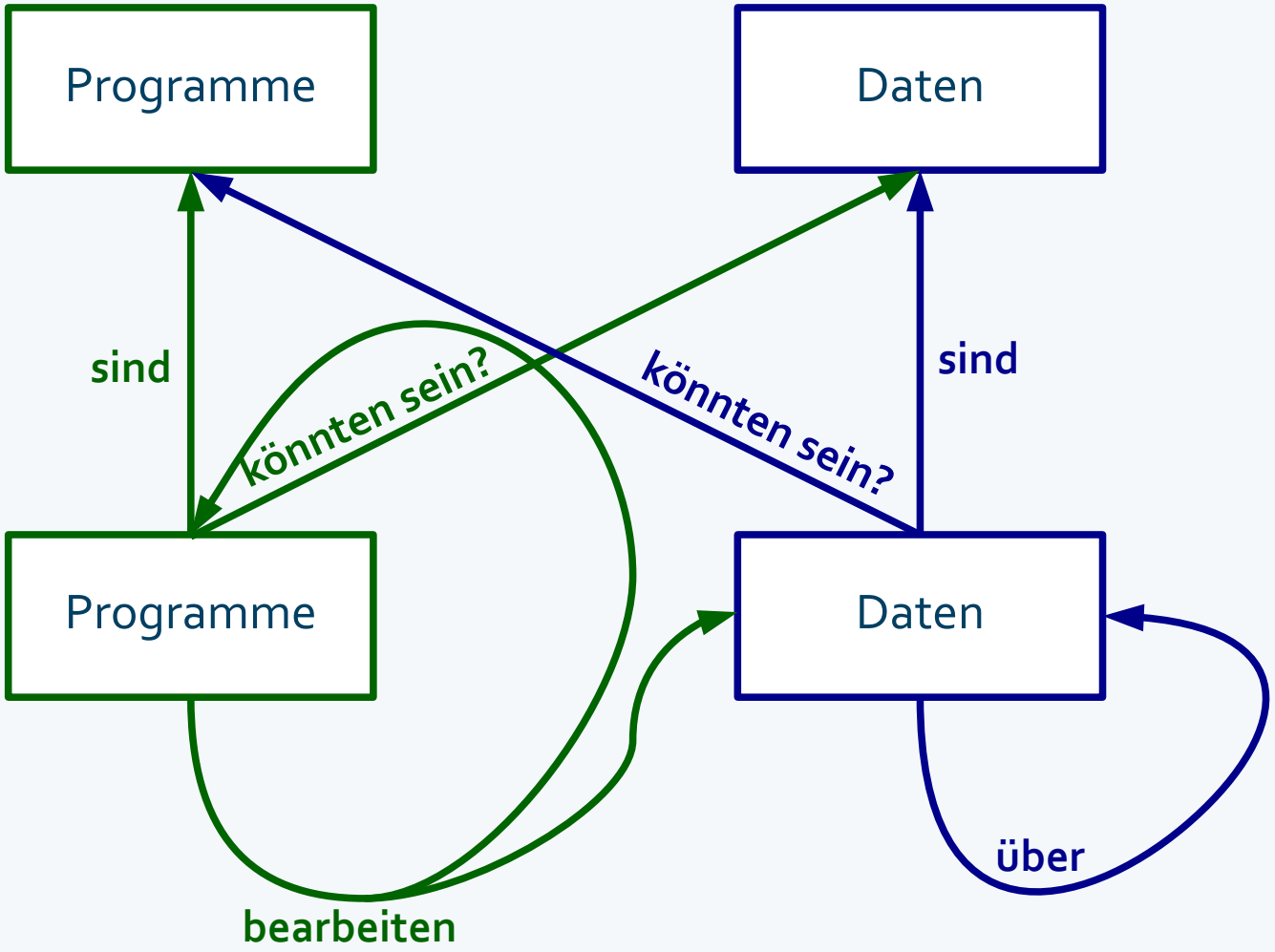
Prof. Dr. Deweß

Thema 7

Metaprogrammierung, insb. Reflexion

- Was ist das?
- Wie kann man es in Java realisieren?
- ein paar Beispiele zur Übung

Ansichtssache



„Meta-"

„The word meta is Greek and means ‚among, with, after,' but we can thank New Latin, the language of scientific nomenclature, for its use prefixing the names of certain disciplines. In its most basic use, meta- describes a subject in a way that transcends its original limits, considering the subject itself as an object of reflection.

Consequently, metatheory is theory about theory; metacriticism is criticism about how we critique a work of art; metafiction is fiction in which the author takes steps to acknowledge the artifice of fiction writing. And metadata is data that helps you further analyze other data—for example, the number of surveys that were taken on a given subject.

So many things, in fact, can be viewed through the lens of self-reference that meta has taken on its own identity as an adjective."

That's so meta (2017) *Merriam-webster.com*. Merriam-Webster. Verfügbar unter: https://www.merriam-webster.com/words-at-play/meta-adjective-self-referential (Zugegriffen: 19. April 2023).

Metaprogramme

- **Metaprogramm**: Programm, welches andere Programme bearbeitet
- bearbeitetes Programm entspricht damit den Daten des Metaprogramms

- Sprache, in der Metaprogramm geschrieben ist, ist dem entsprechend eine **„Metasprache"**, also eine Sprache, die benutzt wird, um über eine Sprache zu sprechen
- Sprache des bearbeiteten Programms ist passend dazu eine **„Objektsprache"**, also eine Sprache, über die gesprochen wird

Reflexion

Mit „Reflexion“ wird allgemein die grundsätzliche Fähigkeit von etwas bezeichnet, sich selbst zu repräsentieren, sich selbst zu bearbeiten und damit mit sich selbst analog dazu wie mit den Dingen, die eigentlich sonst betrachtet werden, umzugehen.

Reflexion (Programmierung)

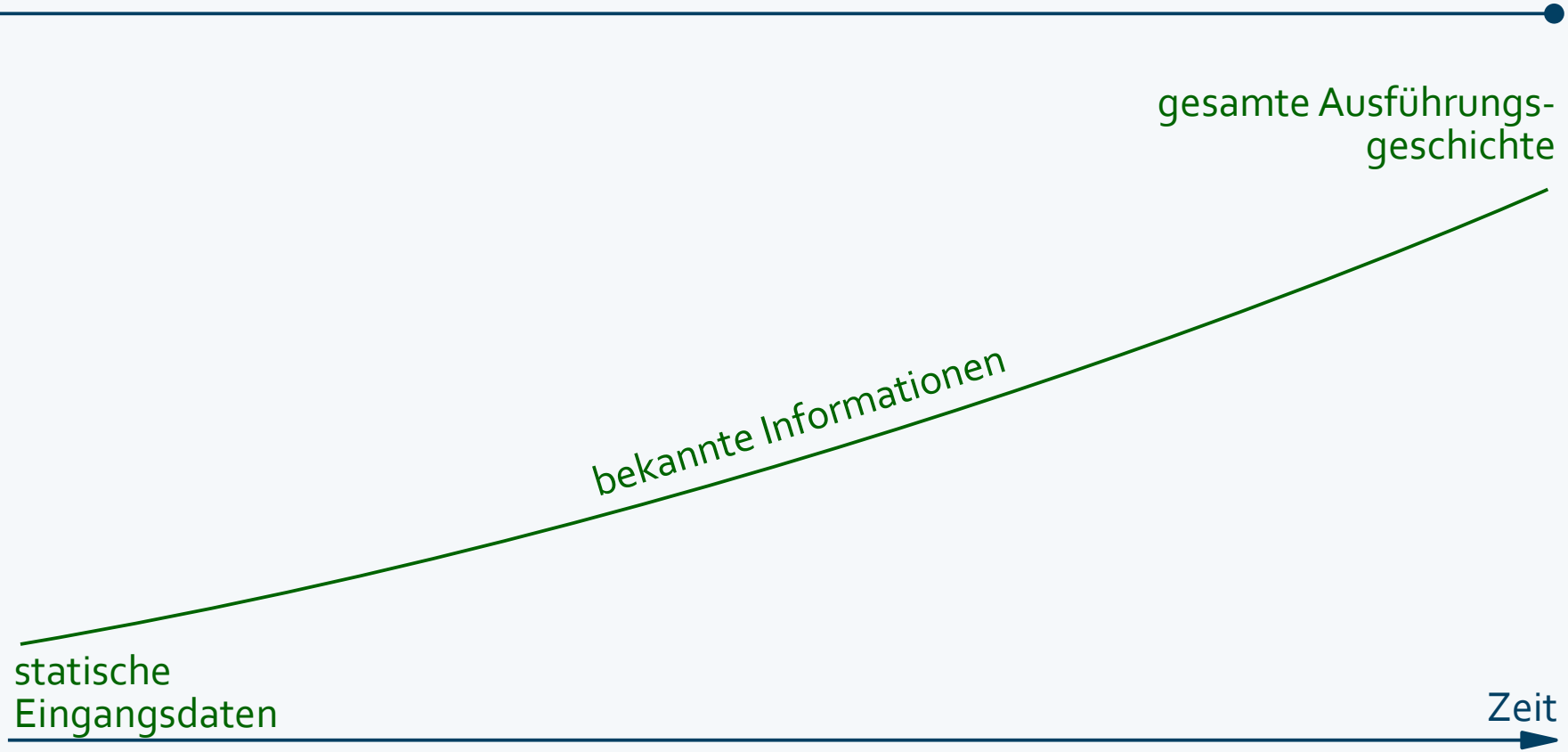
- Fähigkeit eines Programms, mit sich selbst während der Laufzeit so wie mit Daten umzugehen.
- Programmiersprache kann damit die eigene Metasprache sein
- erfordert Mechanismus zur Kodierung des Ausführungszustandes als Daten (Reifikation)

Reflexion

Aspekte von Reflexion

- Introspektion: Fähigkeit zur Beobachtung des eigenen Zustands und Ableitung daraus folgender Erkenntnisse.
- Interzession: Fähigkeit zur Veränderung des eigenen Ausführungszustandes bzw. der eigenen Interpretation

Reflexion – Warum?



Vor-der-Kompilierung | Kompilierung | Verlinkung | Ladezeit | Laufzeit | Nach-der-Laufzeit

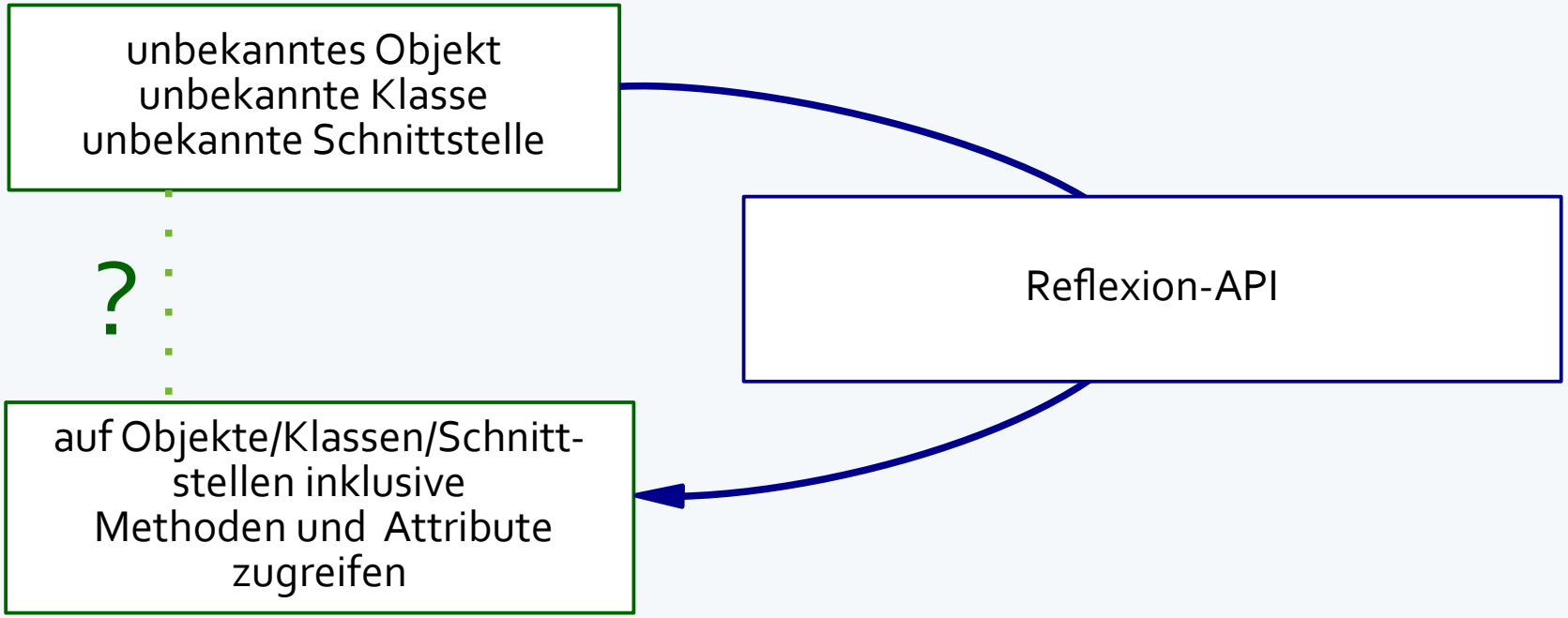
Je mehr Informationen verfügbar sind, desto besser kann man reagieren :-)

Reflexion – Level der Sprachunterstützung

- sehr hoch
  - rein objektorientierte Sprache „Smalltalk", bei der alles als Objekte, also auch Klassen als Klassenobjekte, repräsentiert werden, die analog zu Objekten während der Laufzeit bearbeitet werden können
  - „Common Lisp Object System" als objektorientierte Erweiterung von „Common Lisp", die eine Veränderung des Metaobjektmodells erlaubt
- mittelmäßig
  - Sprachen wie Java und C#
  - Methoden und Attribute können während der Laufzeit ermittelt werden
  - Objekterzeugung zuerst bei Laufzeit namentlich bekannter Klassen ist möglich
  - hauptsächlich Introspektion; Klassen-Metaobjekte werden nicht zur Laufzeit geändert
- niedrig
  - einfache Introspektion, z.B. C++ Runtime type identification (RTTI)

Reflexion – Umsetzung

- API-basierter Programmzugriff (z.B. Java)



- spezielle Sprach- und Kompilerunterstützung (z.B. C++ Templates)

Anwendungen - Metaprogrammierung und Reflexion

- Java-Kompiler (Programmiersprache, in der Kompiler geschrieben ist, ist dann die „Metasprache“) verarbeitet Java-Code (hier Java als „Objektsprache“) und produziert Bytecode

Aufgabe 1

Wann handelt es sich bei einem Java-Kompiler „einfach nur“ um ein Metaprogramm und wann um ein reflexives Metaprogramm?

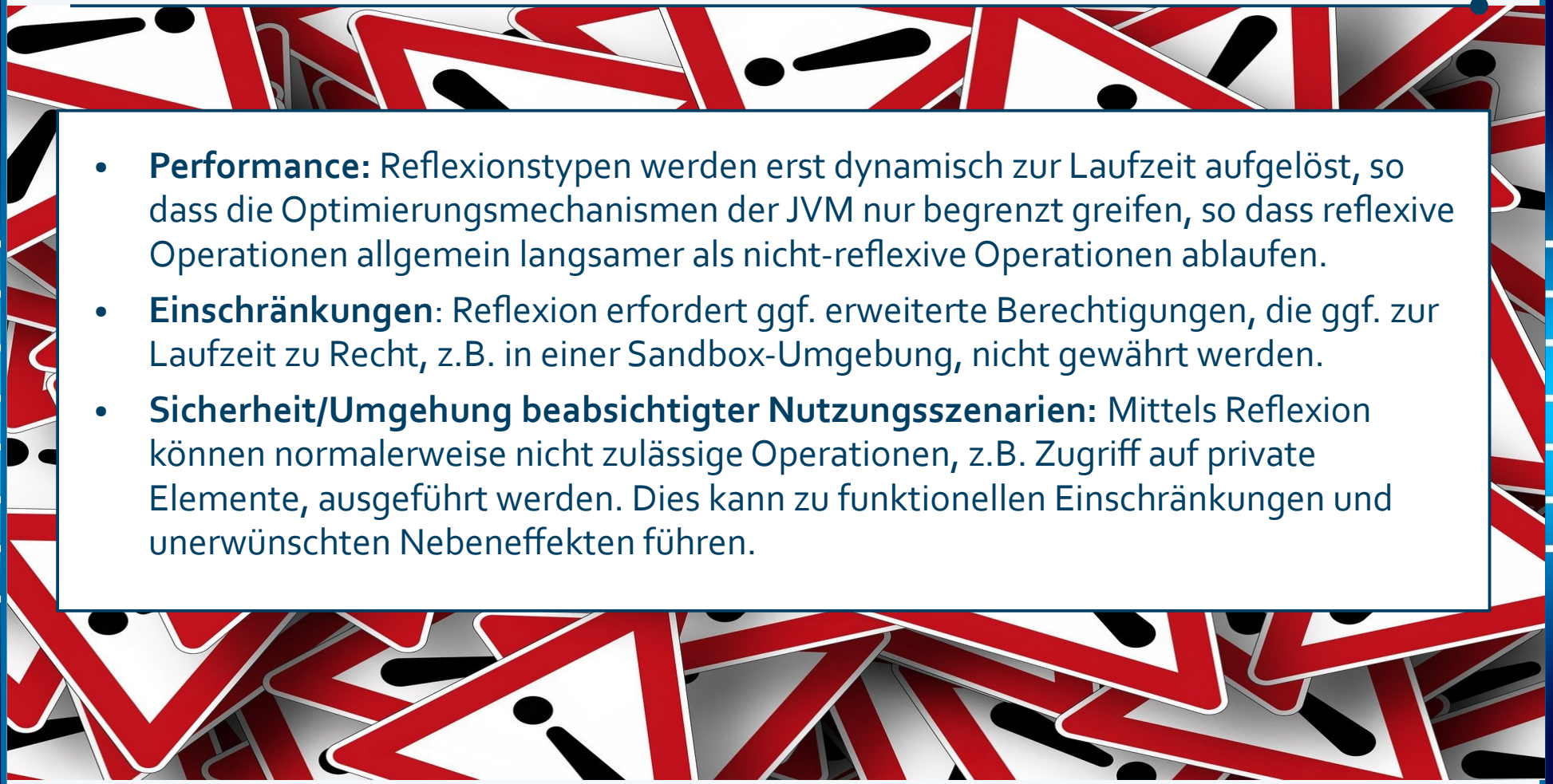
- entsprechende Programme zur Analyze, Erzeugung und Manipulation von Byte-Code (eine interessante Bibliothek dazu ist die „Apache Commons BCEL“, zu finden unter https://commons.apache.org/proper/commons-bcel/)
- Debugger, Interpreter, Entwicklungsumgebungen
- Klassenbrowser für objektorientierten Quellcode
- Quellcode-Werkzeuge (z.B. CheckStyle), Testwerkzeuge
- Anpassung laufzeitumgebungsunabhängiger Programme an spezielle Laufzeitumgebungen
- Umsetzung nutzerbasierter Klassen, bei Nutzung fremder Bibliotheken
- ...

Java Reflexion

- Programmiersprache Java unterstützt Reflexion an sich und bietet speziell dafür geeignete Standard-Bibliotheken an:
  - java.lang.reflect
  - java.lang.Class, java.lang.Package, java.lang.Module, ...
- angebotene Möglichkeiten
  - Klassennamen usw. zur Laufzeit ermitteln
  - Klassen usw. zur Laufzeit in den Speicher laden
  - Methoden, Felder, Konstruktoren usw. untersuchen
  - zur Laufzeit Objekte (Konstruktoren aufrufen) und Instanzen von Schnittstellen (Stichwort „Proxy"-Klassen) erstellen
  - zur Laufzeit Methoden aufrufen und auf Felder zugreifen

Wir schauen uns nur einen ganz kleinen Teil davon an, weil Metaprogrammierung eigentlich etwas für Senior-Entwickler und nichts für Anfänger ist.

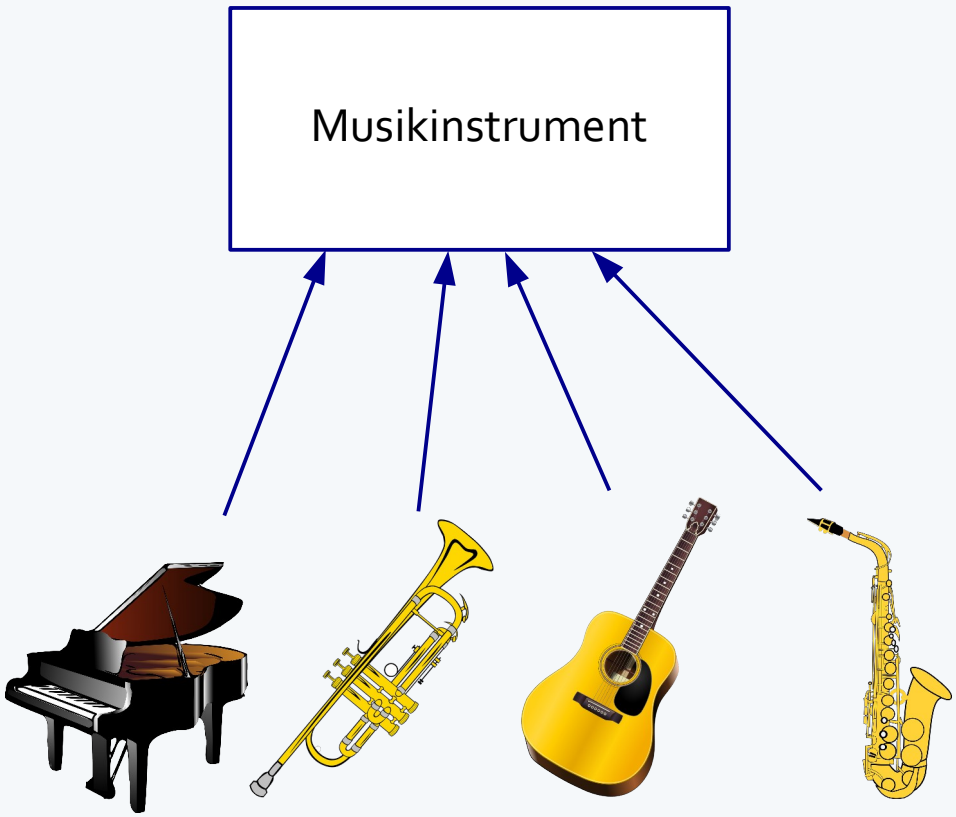
Java Reflexion – **Warnungen**

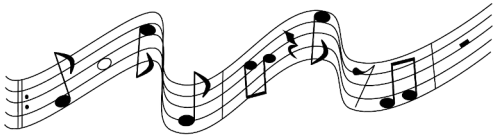


- **Performance:** Reflexionstypen werden erst dynamisch zur Laufzeit aufgelöst, so dass die Optimierungsmechanismen der JVM nur begrenzt greifen, so dass reflexive Operationen allgemein langsamer als nicht-reflexive Operationen ablaufen.
- **Einschränkungen**: Reflexion erfordert ggf. erweiterte Berechtigungen, die ggf. zur Laufzeit zu Recht, z.B. in einer Sandbox-Umgebung, nicht gewährt werden.
- **Sicherheit/Umgehung beabsichtigter Nutzungsszenarien:** Mittels Reflexion können normalerweise nicht zulässige Operationen, z.B. Zugriff auf private Elemente, ausgeführt werden. Dies kann zu funktionellen Einschränkungen und unerwünschten Nebeneffekten führen.

Was unproblematisch ohne Reflexion umgesetzt werden kann, sollte ohne Reflexion umgesetzt werden!

Java Reflexion – Vorwissen

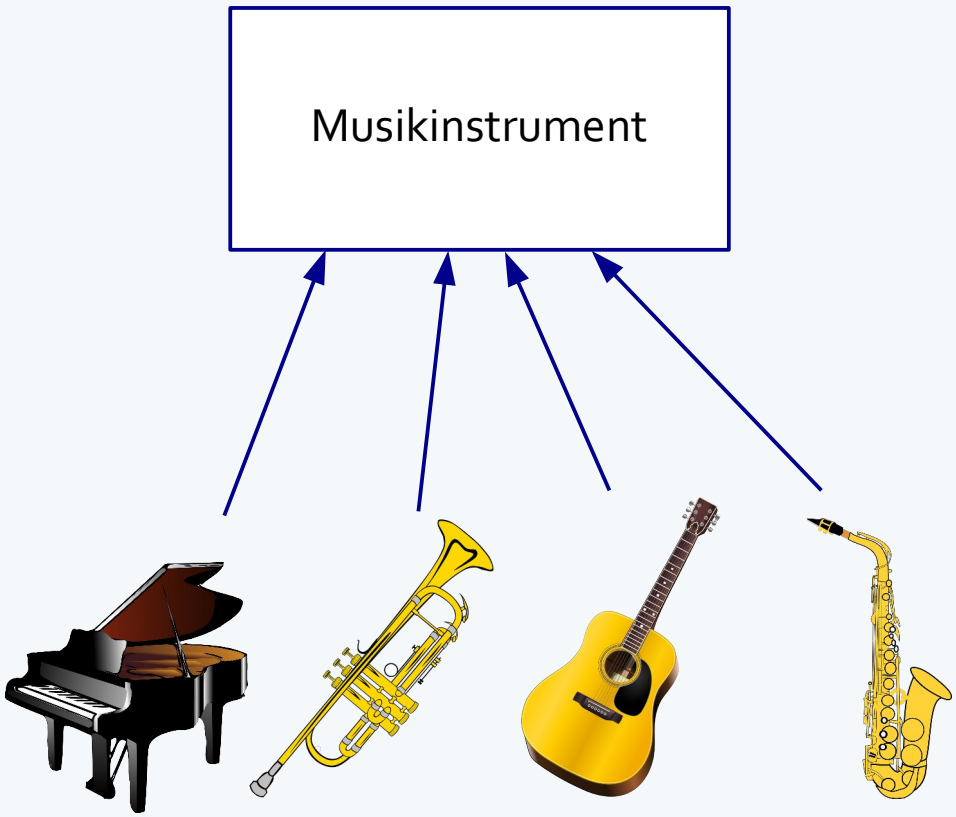




Was muss beim Aufruf einer Methode „spieleMelodie(…)" für ein Musikinstrument geschehen?

Per „Polymorphie" wird sich darum gekümmert, dass die Melodie gezupft, geblasen oder per Tastendruck erzeugt wird.

Java Reflexion – Vorwissen



Aber was ist, wenn ich ein Konzertprogramm geschrieben habe und der Nutzer z.B. alle Bläser spielen lassen darf?

Dann muss ich zur Laufzeit irgendwie bestimmen können, von welchem Typ ein Musikinstrument ist.
Per Laufzeittypinformationen (Run-Time Type Information, RTTI) ist genau das möglich.

Java Reflexion – Vorwissen

Wie funktioniert Laufzeittypinformation (RTTI)?

Wie wird „Typ-Information" in Java zur Laufzeit überhaupt repräsentiert?

- für jede Klasse/Schnittstelle/Enumeration … gibt es ein „Klassenobjekt" (Objekt der Klasse „Class"), welches Informationen über die Klasse/Schnittstelle/Enumeration … enthält und für die Erstellung der eigentlichen Objekte benutzt wird
- „Klassenobjekt" wird nach dem Kompilieren in gleichnamiger „.class"-Datei gespeichert und bei Bedarf geladen
- Laufzeittypinformation wird über dieses „Klassenobjekt", auf das zugegriffen werden kann, realisiert

Ein wichtiger Teil der Java-Laufzeitumgebung (JRE) sind neben der virtuellen Maschine (JVM) die „Class-Loader" (interessant, aber nicht klausurrelevant), die alle benötigten Klassen usw. (als „Klassenobjekte") bei Bedarf in den Speicher der JVM laden.

- Dank der „Class-Loader" muss sich die JVM selbst nicht mit dem plattformabhängigen Dateisystem beschäftigen
- „Class-Loader" laden nicht einfach nur irgendwelche Klassen, sondern dienen auch dem Sicherheitsmanagement in Java

Java Reflexion – Zwischenbemerkung

Wir werden uns auf ganz normale „Klassen“ und primitive Typen beschränken und auch da nur einen kleinen Teil betrachten – Sie behalten bitte im Hinterkopf, dass man viele Dinge natürlich auch für „Klassenobjekte“ machen kann, die Schnittstellen usw. repräsentieren.

Java Reflexion – Erste Versuche

Wann ist „bei Bedarf", so dass Klassen von einem „Class-Loader" geladen werden?

Wann stehen also „Klassenobjekte" und damit „Typ-Information" zur Verfügung?

- beim ersten Verweis auf ein statisches Mitglied der Klasse, wobei Konstruktoren implizit als statische Mitglieder der Klasse zählen
- „Klassenobjekte" sind dann wie normale „Objekte" und können dementsprechend referenziert und verwendet werden

Aufgabe 2

Bitte erstellen Sie für unser Experiment in einem neuen Paket 3 Klassen „ClassA", „ClassB" und „ClassC", die jeweils lediglich enthalten:

- Standardkonstruktor, der eine individuelle Nachricht auf der Konsole ausgibt
- (einen nicht klausurrelevanten) Klasseninitialisierer (entspricht einem Konstruktor für das „Klassenobjekt"), der eine individuelle Nachricht auf der Konsole ausgibt (siehe https://docs.oracle.com/javase/tutorial/java/javaOO/initial.html)

Java Reflexion – Erste Versuche

Lösung 2 – nur „ClassA", andere Klassen analog

```
package de.baleipzig.meta;

public class ClassA {

  static {
    System.out.println("Hello from A");
  }

  public ClassA() {
    System.out.println("New A");
  }

}
```

Java Reflexion – Erste Versuche

Aufgabe 3

Bitte erstellen Sie für unser Experiment und zukünftige Aufgaben im gleichen Paket wie Ihre 3 Klassen „ClassA", „ClassB" und „ClassC" eine Klasse „AgentY" folgender Art:

```
package de.baleipzig.meta;
public class AgentY {
  public static void main(String[] args) {
    System.out.println("Agent Y starts ...");
    new ClassA();
    System.out.println("After creating ClassA");
    try {
      Class.forName("de.baleipzig.meta.ClassB");
    } catch (ClassNotFoundException e) {
      System.out.println("ClassB missing ...");
    }
    System.out.println("After trying something ...");
    new ClassC();
    System.out.println("After creating ClassC");
  }
}
```

Lassen Sie die Klasse laufen und schauen Sie sich an, was der Reihe nach passiert.

Java Reflexion – Erste Versuche

Lösung 3 - Konsolenausgabe

```
Agent Y starts ...
Hello from A
New A
After creating ClassA
Hello from B
After trying something ...
Hello from C
New C
After creating ClassC
```

Hier haben wir über die statische Methode „forName“ von „Class“ eine Referenz auf unser entsprechendes „Klassenobjekt“ (Objekt der Klasse „Class“) für „ClassB“ erhalten, wofür natürlich die Klasse „ClassB“ geladen werden musste …

wir sehen gut, wann welche Klassen verfügbar sind

Java Reflexion – „Klassenobjekte“

Wie kann man z.B. eine Referenz auf ein „Klassenobjekt“ einer Klasse erhalten?

- ausgehend von einer Instanz über Methode „getClass“ von „Object“, z.B.:

```
String aString = "test";
Class classTest1 = aString.getClass();
System.out.println(classTest1);
```

- ausgehend vom Datentyp über sein „Klassenobjekt“, z.B.:

```
Class classTest2 = Integer.class;
System.out.println(classTest2);
```

- über einen Klassennamen mittels Methode „forName“ von „Class“, z.B.:

```
Class<?> classTest3 = null;
try {
  classTest3 = Class.forName("de.baleipzig.meta.ClassB");
} catch (ClassNotFoundException e) {
  e.printStackTrace();
}
System.out.println(classTest3);
```

Java Reflexion – „Klassenobjekte“

Wie kann man z.B. eine Referenz auf ein „Klassenobjekt“ einer Klasse erhalten?

- ausgehend von einer Instanz über Methode „getClass“ von „Object“, z.B.:

```
String aString = "test";
Class classTest1 = aString.getClass();
System.out.println(classTest1);
```

- ausgehend vom Datentyp über sein „Klassenob

```
Class classTest2 = Integer.class;
System.out.println(classTest2);
```

- über einen Klassennamen mittels Methode „fo

```
Class<?> classTest3 = null;
try {
  classTest3 = Class.forName("de.baleipzig.meta.ClassB");
} catch (ClassNotFoundException e) {
  e.printStackTrace();
}
System.out.println(classTest3);
```

Aufgabe 4

Testen Sie in der main-Methode von „AgentY“ alle 3 Varianten, indem Sie jeweils passend ausgehend vom Typ bzw. Namen der Klasse „ClassA“ bzw. einem konkreten Objekt der Klasse „ClassA“ jeweils eine Referenz auf das jeweils zugehörige Klassenobjekt erzeugen. Nutzen Sie bei jeder Variante wieder die Konsolenausgabe wie im gegebenen Beispiel.

Zusatzaufgabe

Warum funktioniert die erste Variante nicht bei „`int i = 3`“?

Java Reflexion – Lösung 4

- ausgehend von einer Instanz über Methode „getClass“ von „Object“:

```
ClassA aClassA = new ClassA();
Class classTest1 = aClassA.getClass();
System.out.println(classTest1);
```

- ausgehend vom Datentyp über sein „Klassenobjekt“:

```
Class classTest2 = ClassA.class;
System.out.println(classTest2);
```

- über einen Klassennamen mittels Methode „forName“ von „Class“:

```
Class<?> classTest3 = null;
try {
  classTest3 = Class.forName("de.baleipzig.meta.ClassA");
} catch (ClassNotFoundException e) {
  e.printStackTrace();
}
System.out.println(classTest3);
```

Java Reflexion – „Klassenobjekte“

Einige Methoden, die auch Referenzen auf ein „Klassenobjekt“ zurückgeben:

- Class.getSuperclass(), z.B.:

```
Class classTest4 = classTest1.getSuperclass();
System.out.println(classTest4);
```

- Class.getEnclosingClass(), z.B.:

```
Class classTest5 = classTest1.getEnclosingClass();
System.out.println(classTest5);
```

- Class.getClasses() (liefert öffentliche Mitglieder-Klassen, -Schnittstellen, ...)
- Class.getDeclaredClasses() (liefert Mitglieder-Klassen, -Schnittstellen ...)
- Class.getDeclaringClass()
  - java.lang.reflect.Field.getDeclaringClass()
  - java.lang.reflect.Method.getDeclaringClass()
  - java.lang.reflect.Constructor.getDeclaringClass()

Java Reflexion – Ein Experiment

Aufgabe 5 - Vorbereitung

Bitte erstellen Sie in Ihrem Paket eine Klasse „AgentX“. Jeder „Agent X“ soll (wie üblich mit Zugriffsmodifizierer „private“) einen Namen („name“ vom Typ „String“) und ein Alter („age“ vom Typ „int“) haben.

Implementieren Sie mit Hilfe Ihrer Entwicklungsumgebung alle Getter und Setter sowie einen Konstruktor, der beide Eigenschaften als Parameter übernimmt und einen Standardkonstruktor.

Java Reflexion – Ein Experiment

Lösung 5 - Vorbereitung

```
package de.baleipzig.meta;

public class AgentX {

  private String name;
  private int age;

  public AgentX() {
    this.name = "John Doe";
    this.age = 30;
  }

  public AgentX(String name, int age) {
    this.name = name;
    this.age = age;
  }

  //automatische Getter, Setter
  //...

}
```

Java Reflexion – Ein Experiment

Aufgabe 6 – das Experiment

Wir wollen herausfinden, von welcher Klasse Meta-Klassen sind.

a) Erstellen Sie einen neuen Agent X mit Referenzvariable „whois".

b) Ermitteln Sie mittels der Methode „getClass" das zugehörige „Klassenobjekt" von „whois", nehmen Sie die gelieferte Information in einer neuen Variable namens „metaTest1" vom Typ „Class" entgegen und erzeugen Sie eine Konsolenausgabe mit „System.out.println(metaTest1)".

c) Wiederholen Sie b) für „metaTest1" und verwenden Sie entsprechend eine neue Variable namens „metaTest2" für die neuen Informationen.

d) Wiederholen Sie c) für „metaTest2" und verwenden Sie entsprechend eine neue Variable namens „metaTest3" für die neuen Informationen.

Java Reflexion – Ein Experiment

Aufgabe 6 – Lösung

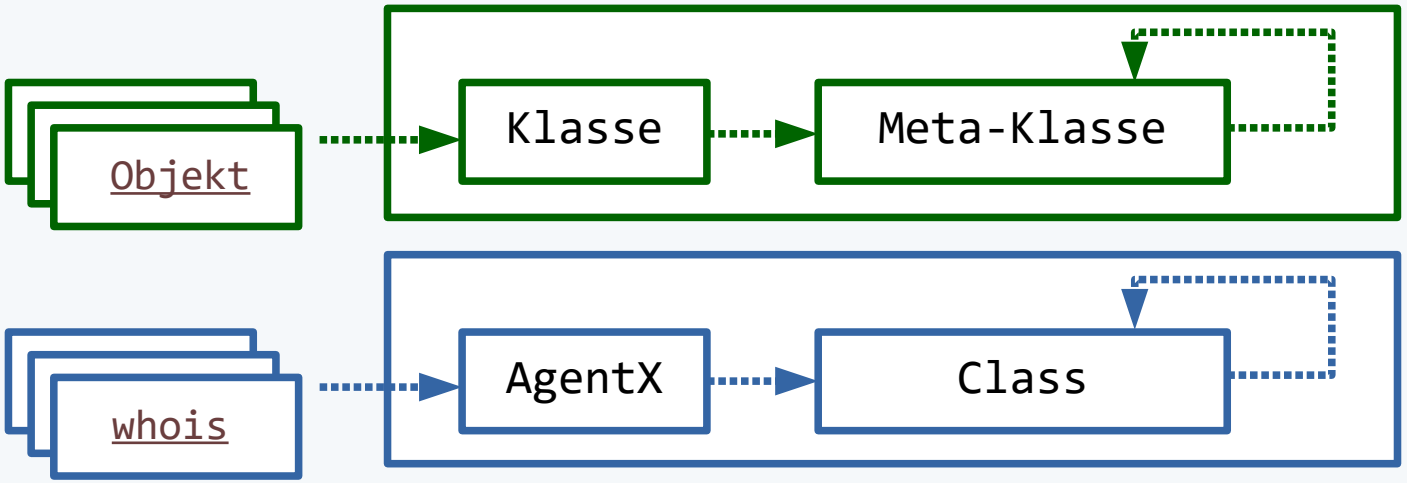
- Objekte sind Instanzen von Klassen
- Klassen sind Instanzen der Meta-Klasse „Class“
- die Meta-Klasse „Class“ ist eine Instanz von sich selbst

```
AgentX whois = new AgentX();

Class metaTest1 = whois.getClass();
System.out.println(metaTest1);

Class metaTest2 = metaTest1.getClass();
System.out.println(metaTest2);

Class metaTest3 = metaTest2.getClass();
System.out.println(metaTest3);
```



Java Reflexion – „Klassenobjekte“

java.lang.Class<T>

- sofern man von einem „Object“ die Klasse ermittelt hat oder anderweitig eine Klasseninformation erhalten hat, bietet diese API viele Möglichkeiten, um mehr über eine Klasse zu erfahren
- Link zur Dokumentation:

  https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Class.html

java.lang.reflect

- Paket mit Klassen und Schnittstellen, um reflexiv Informationen über Klassen, Schnittstellen und deren Mitglieder zu erhalten und zu modifizieren, z.B.:
  - java.lang.reflect.Field
  - java.lang.reflect.Method
  - java.lang.Constructor
  - java.lang.reflect.Modifier
  - Schnittstelle „Member“
- Link zur Dokumentation:

  https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/reflect/package-summary.html

Java Reflexion – Weitere Informationen

Aufgabe 7

Wir wollen Zugriffsmodifizierer, Instanzvariablen und Methoden von „AgentX" herausfinden.

a) Nutzen Sie die Methode „getModifiers()" aus „java.lang.Class<T>" und anschließend die Methode „toString(int mod)" aus „java.lang.reflect.Modifier", um den Zugriffsmodifizierer der Klasse „AgentX" zu ermitteln. Geben Sie diesen zur Kontrolle auf der Konsole aus.

b) Ermitteln Sie ebenfalls die „Fields" und „Methods" von „AgentX". Nutzen Sie dafür passende Methoden aus „Class". Dort finden Sie auch die passenden Typbezeichnungen für die erhaltenen Rückgabewerte der jeweiligen Methoden.

c) Iterieren Sie jeweils über die erhaltenen Arrays aus b) und geben Sie auf der Konsole unter Nutzung von „getName()" auch die Namen der erhaltenen „Fields" und „Methods" an.

Java Reflexion – Weitere Informationen

Lösung 7

```
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
```

```
Class aClass = AgentX.class;

System.out.println(Modifier.toString(aClass.getModifiers()));

Field fields[] = aClass.getDeclaredFields();
for (Field f : fields) {
  System.out.println(f.getName());
}

Method methods[] = aClass.getDeclaredMethods();
for (Method m : methods) {
  System.out.println(m.getName());
}
```

Java Reflexion – Geheime Werte

Aufgabe 8

Wir wollen schauen, ob wir von einem Agent X ohne Nutzung der Methoden der Klasse „AgentX" an alle Eigenschaftswerte kommen. (Würde Sinn machen, wenn die Klasse „AgentX" gar nicht von uns wäre und wir ohne weitere Kenntnisse diese Daten erhalten möchten – zu Übungszwecken geht es nicht anders als mit einer uns bekannten Klasse.)

a) Erzeugen Sie einen neuen Agent X „firstAgent" mit Namen „James", der 30 Jahre alt ist.

b) Nutzen Sie die Methode „getClass()" von „Object", um das zugehörige „Klassenobjekt" zu erhalten.

c) Ermitteln Sie ausgehend davon zunächst einen Array, der alle Eigenschaften eines Agent X enthält.

d) Iterieren Sie über diesen Array und geben Sie so für jede Eigenschaft den Eigenschaftsname gefolgt vom Eigenschaftswert unseres Agent X „firstAgent" aus.

   Hinweis: Für den Zugriff auf die eigentlichen Eigenschaftswerte können Sie zwei Methoden aus „java.lang.reflect.Field" nutzen – eine, mit der Sie den Zugriff überhaupt erlauben und eine, mit der Sie dann den entsprechenden Wert erhalten.

Zusatzaufgabe: Bauen Sie in Ihren Quellcode Befehle derart ein, dass jeder Eigenschaftswert „James" auf den Eigenschaftswert „Bond" geändert wird.

Java Reflexion – Geheime Werte

Lösung 8 (inkl. Zusatzaufgabe)

```
//...

AgentX firstAgent = new AgentX("James",30);
Class<?> agentClass = firstAgent.getClass();
Field agentFields[] =  agentClass.getDeclaredFields();
for (Field f : agentFields) {
  System.out.print(f.getName() + ": ");
  try {
    f.setAccessible(true);
    System.out.println(f.get(firstAgent));
    if (f.get(firstAgent)=="James") {
      f.set(firstAgent, "Bond");
    }
  } catch (Exception e) {
    //...
  }
}

//...
```

Java Reflexion – „Fake Agents“

Aufgabe 9

Ein Agent ist kein Problem, aber wie sieht es mit einem kompletten Geheimdienst aus?

a) Deklarieren Sie typsicher eine Menge zur Aufnahme von Agenten vom Typ „AgentX“ und initialisieren Sie diese mit einem neuen „HashSet“.

b) Fügen Sie als Erstes Ihren „firstAgent“ aus der letzten Aufgabe dieser Menge hinzu.

c) Deklarieren Sie eine Variable „fakeAgent“ für einen neuen Agent X und initialisieren Sie diese zunächst mit „null“.

d) Nutzen Sie das „Klassenobjekt“ für derartige Agenten aus der letzten Aufgabe und:

   1. eine passende Methode von „java.lang.Class“, um einen Konstruktor zu erhalten sowie
   2. diesen nutzend eine Methode aus „java.lang.reflect.Constructor<T>“, um einen neuen Standard-Agent-X (Standardkonstruktor ohne Parameter wird genutzt) zu erzeugen.
   3. Weisen Sie die Referenz darauf dem „fakeAgent“ zu.

e) Nehmen Sie nun den „fakeAgent“ mit in Ihre Agenten-Menge auf und lassen Sie sich die Menge auf der Konsole anzeigen.

Zusatzaufgabe: Nehmen Sie statt dem Standard-Agent-X den 25-jährigen Agent namens „Hunt“ auf.

Java Reflexion – „Fake Agents“

Lösung 9 (inkl. Zusatzaufgabe)

```
//...

Set<AgentX> ourAgents = new HashSet<>();
ourAgents.add(firstAgent);
AgentX fakeAgent = null;

try {
  fakeAgent = (AgentX) agentClass.getConstructor().newInstance();
  //Zusatzaufgabe stattdessen:
  //fakeAgent = (AgentX) agentClass.getConstructor(String.class,
                int.class).newInstance("Hunt", 25);
} catch (Exception e) {
  // mindestens das, eigentlich mehr
  e.printStackTrace();
}
ourAgents.add(fakeAgent);
System.out.println(ourAgents);

//...
```

Java Reflexion – identische Agenten

Aufgabe 10

Nutzen Sie Ihre Entwicklungsumgebung, um in Ihrer Klasse „AgentX“ noch die geerbten Methoden „hashCode“ und „equals“ von „Object“ passend zu überschreiben.

Nutzen Sie dabei alle verfügbaren Optionen.

Innerhalb von „equals“ können Sie den binären Operator „instanceOf“ entdecken, mit dem getestet werden kann, ob ein Objekt eine Instanz einer bestimmten Klasse ist.

Dieser erinnert an reflexive Programmierung, setzt allerdings auf der rechten Seite zur Kompilierzeit Typkenntnis voraus.

Welche Methode aus „java.lang.Class<T>“ ist das reflexive Pendant dazu?

Java Reflexion – identische Agenten

Lösung 10

Die Methode „isInstance“, mit der zur Laufzeit derartige Überprüfungen dynamisch möglich sind.

```
obj instanceof AgentX
```

wäre zur Laufzeit damit abbildbar über

```
Class testClass = Class.forName("AgentX");
boolean isInstanceRetrunValue  = testClass.isInstance(obj);
```

Java Reflexion – Abschlussbemerkungen

Wir haben nur einen minimalen Einblick in das Thema gewagt, aber trotzdem ist hoffentlich das Potential ersichtlich:

- mit Java Reflexion können Klassen, Schnittstellen, Methoden usw. ohne jeweilige Namenskenntnis zur Laufzeit untersucht werden
- selbst eigentlich „private" Dinge sind nicht mehr unantastbar

- über „ClassLoader" können externe Klassen eingebunden und mittels Reflexion zur Laufzeit analysiert und genutzt (z.B. Erzeugung neuer Objekte davon) werden, so dass Programme mit der Möglichkeit potentieller Erweiterungen programmierbar sind
- außerdem können Datenbank-Objekt-Zuordnungen und die Untersuchung fremder Klassen allgemein relativ einfach umgesetzt werden, so dass z.B. auch Monitoring-Programme mit Java gut programmierbar sind

- damit können Vorteile objektorientierter Sprachen voll ausgespielt werden und die Reflexion in Java zählt daher zu einer der mächtigsten Eigenschaften von Java

Vielen Dank für Ihre Aufmerksamkeit.