Object-oriented Programming & Subversion

Object-**O**riented
Software**C**onstruction

Armin B. Cremers,
Tobias Rho, Daniel Speicher, Holger Mügge

Objectives of this lecture

What is ...

- an object?
- a class?
- an interface?
- polymorphism?

Where do I find these concepts in Java?

How do I use SVN to commit my solutions?

General: More tutorials on Java needed?

Object-oriented Programming

Why OOP?

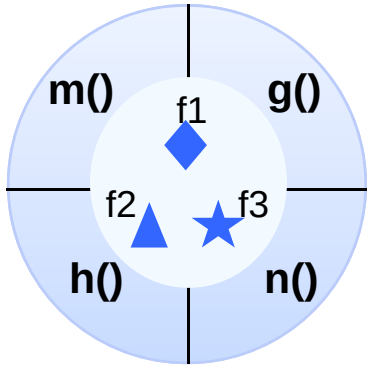
- The world can be thought of as interacting objects

Therefore

- Organize programs into smaller entities (objects)
- Use the same basic mechanisms on that level that we already know from the "real world"

OOP: Objects

- Objects = Unity of data and code
  - fields → state
  - operations → behaviour



- encapsulation
  - the language ensures that the state is only modified via the specified interface

➔ maintainance!

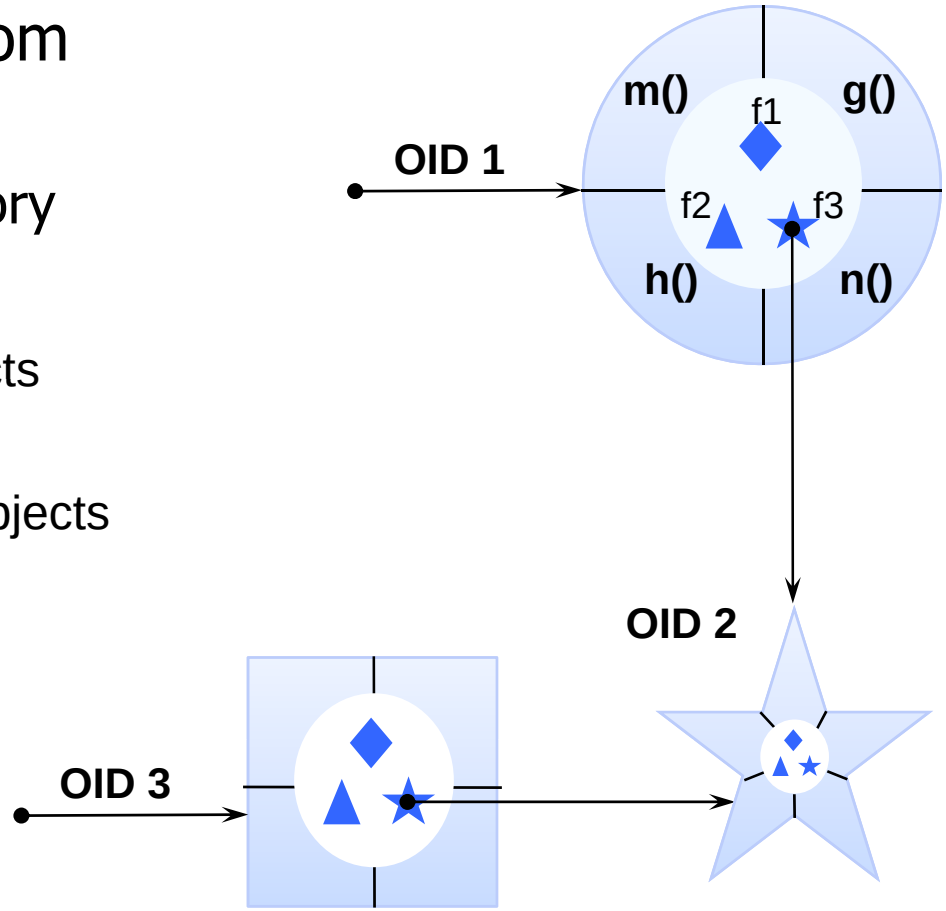
➔ access synchronisation!

OOP: Object-Identity

- OID = Object reference, which is independent from
  - the state of the object
  - of the location im memory

- Variables contain OIDs, not objects

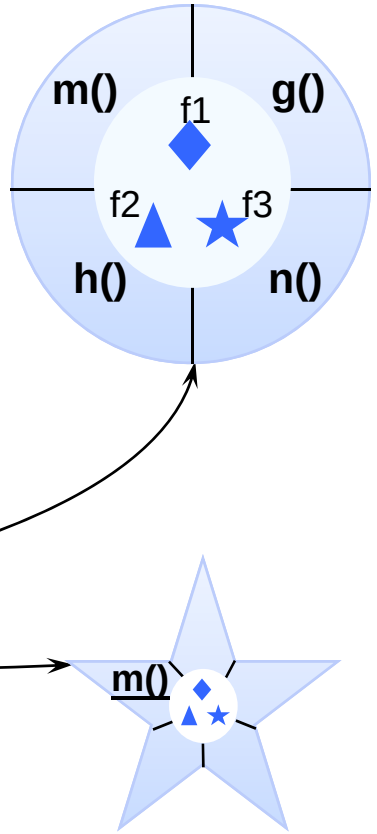
- OIDs enable the shared use of objects



OOP: Messages and „Dynamic Binding“

- Message
  - tells an object to call a certain method
- Message ≠ procedure call
  - the address of the called method is not known
  - may also be used in a distributed environment

```
anObject.m(arg1,...,argN)
```



- Dynamic Binding
  - determination of the code to call is done at runtime and depends on the receiver of the message

Classes, Interfaces, Inheritance

Concepts and Phenomena 1

Phenomenon

- An object in the world of a domain as you perceive it
- *Example:* The lecture you are attending
- *Example:* My black watch

Concept

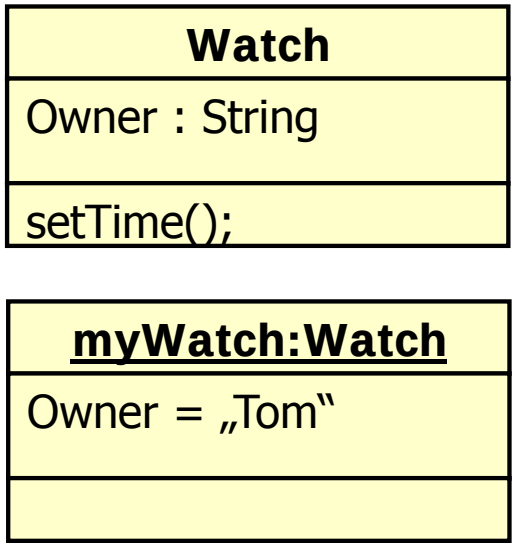
- A set of phenomena with common properties
- *Example:* Lectures on software engineering
- *Example:* watches

A concept described by:

- Name (To distinguish it from other concepts)
- Purpose (Properties that determine if a phenomenon is a member of a concept)
- Members (The set of phenomena which are part of the concept)

Concepts and Phenomena in SE:
Classes and Objects

- Class defines
  - Attributes
  - Operations (Methods)
  - = *Concept*
- Object
  - Instance of a class
  - = *Phenomenon*

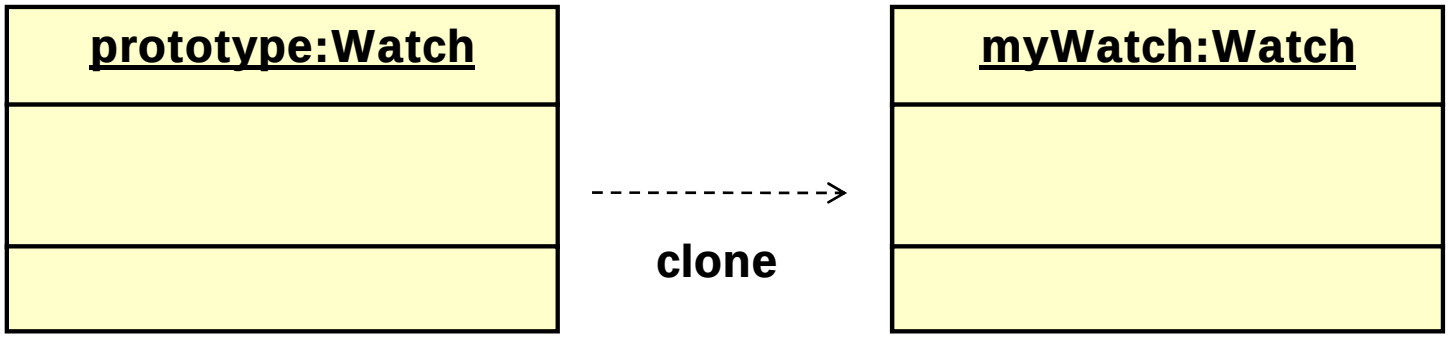



```
// Java Syntax:
public class Watch {


}

Watch watch =  new Watch();
```

- Example
  - Watch (Class)
  - Tom’s watch, Sheila’s watch (Objects)

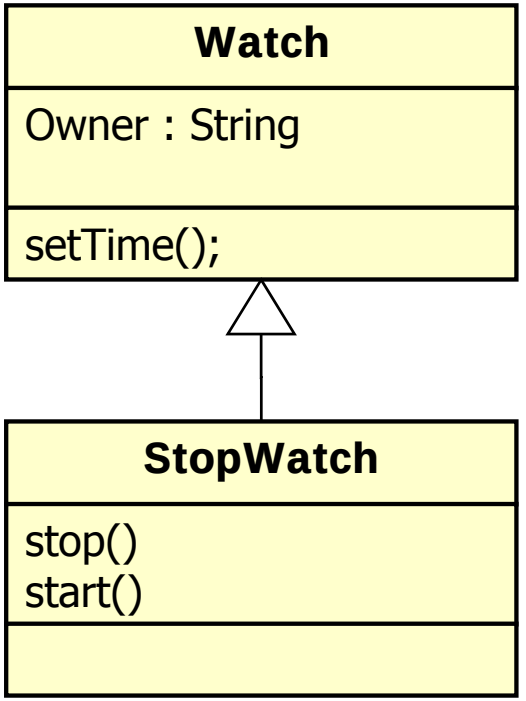
Classes vs. Prototype-based languages

- Most object-oriented languages use classes to define the (static) structure of its instances (objects)
- Alternative approaches: use prototypes
  - E.g.: Self, Javascript
  - Copy objects instead of class instantiation
  - Inheritance by prototype chain
- In this lecture we only considered *class-based object-orientation*



Inheritance

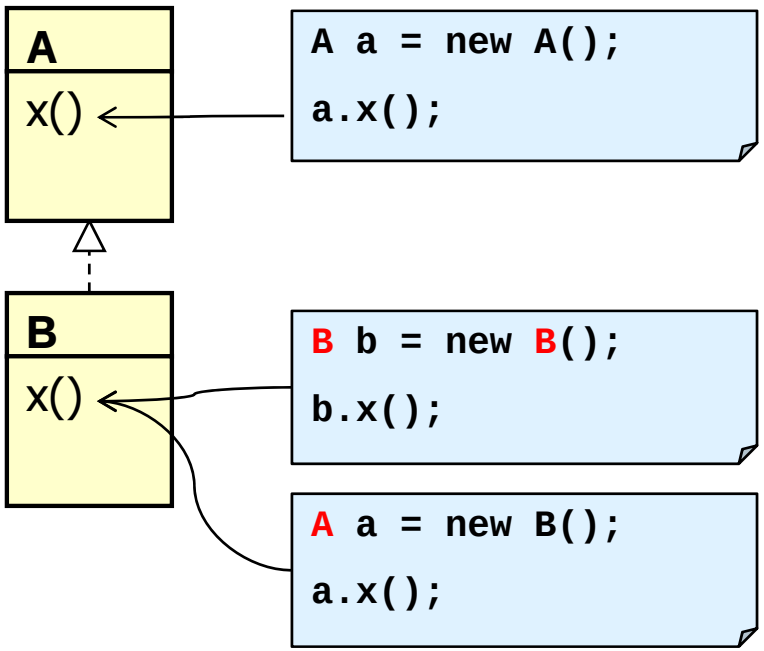
- Form new classes by sub-classing / extending existing classes
  - Take over (inherit) attributes and operations from an existing class
- Advantages
  - Sub classes can be treated like super types (Specification Inheritance)
  - Overriding
  - Code sharing (Implementation Inheritance)

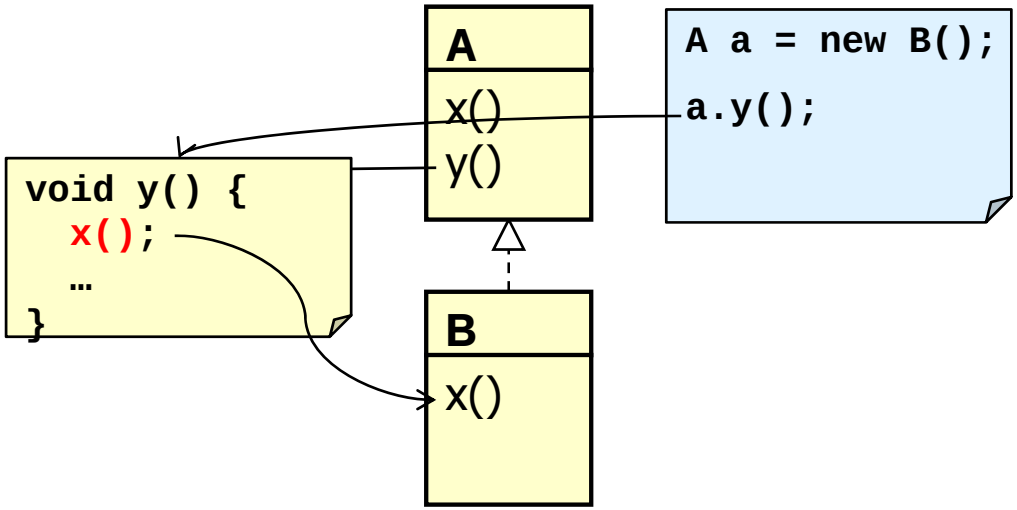



```
// Java Syntax:
public class Watch {
}
class StopWatch extends Watch {
}
```

Overriding

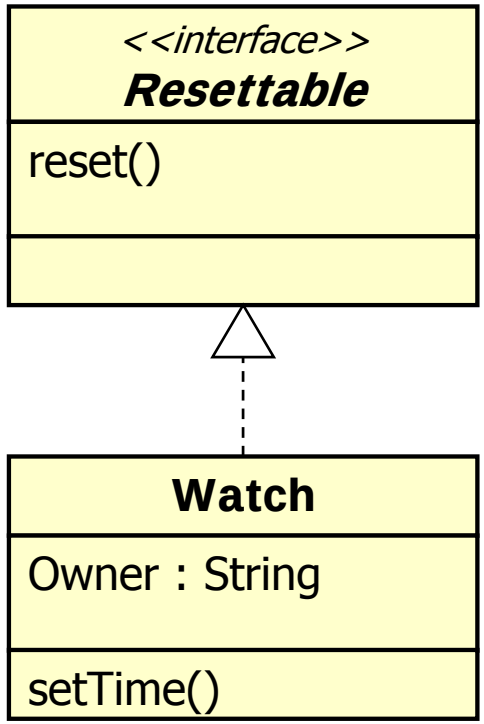
- Redefine methods in the subtype
- Polymorphism
  - Different behavior of an operation for different sub classes (types)
- Example





Interfaces

- Interface
  - Pure Specification
  - No implementation
- Interface defines
  - Abstract Operations (Methods)
  - = *abstract Concept*
- Example
  - Watch (Class)
  - Resettable (Interface)

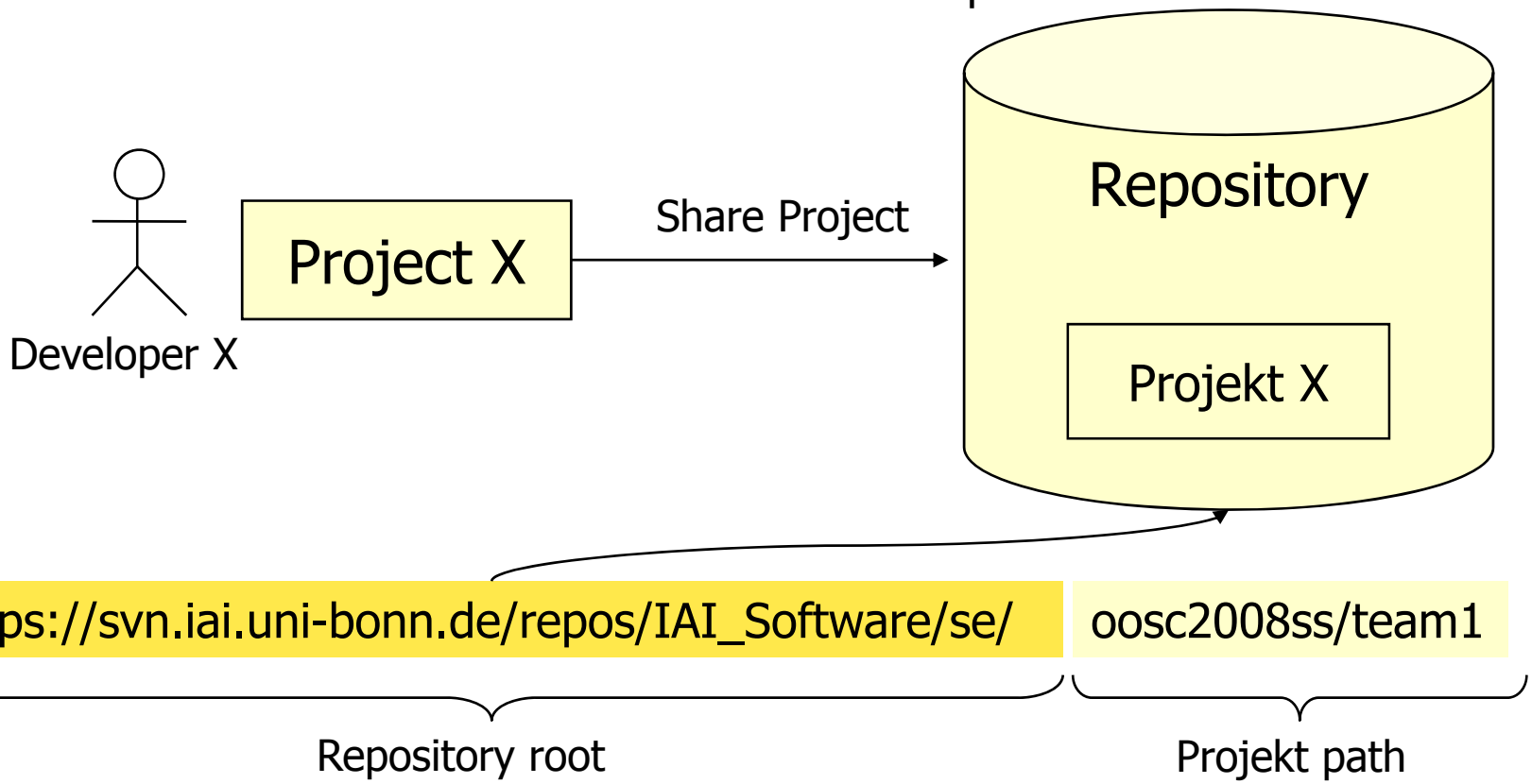



```
// Java Syntax:
public interface Resettable {
   void reset();
}
class StopWatch implements Resettable {
   public void reset(){ … }
}
```

Configuration Management with Subversion

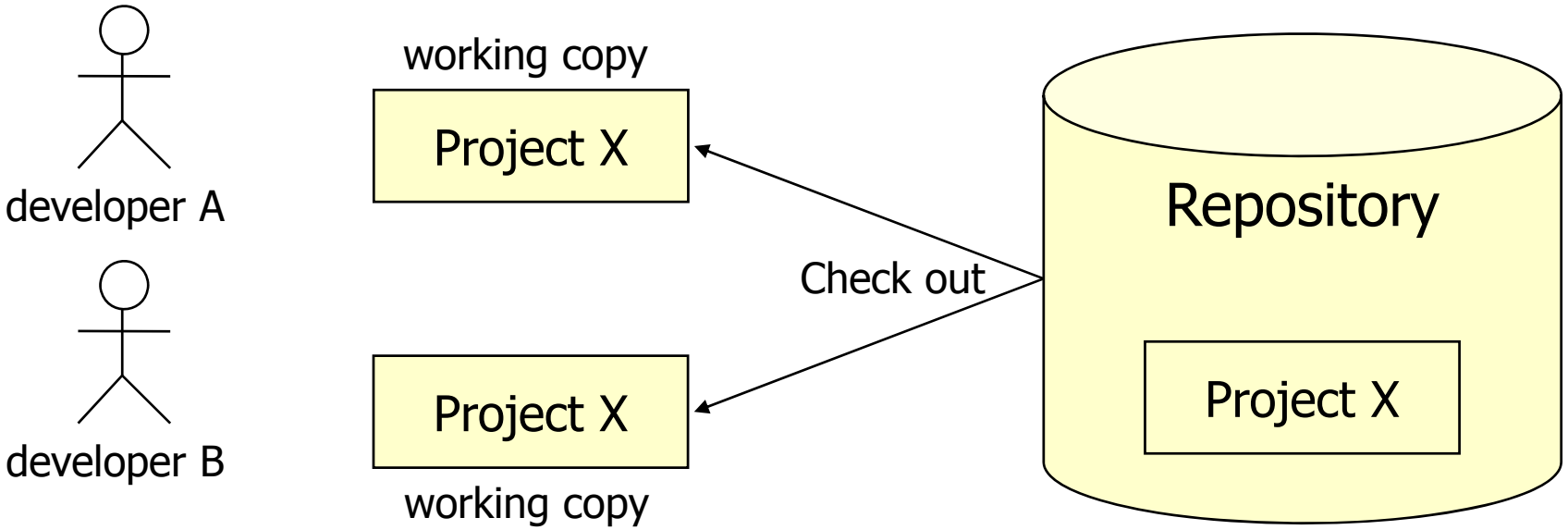
Sharing / Checkout
Put Project Under Version Control

- A repository exists
- Your project exists but you didn't share it.
- **Sharing:** The project is added to the repository
  - ➔ It is now available for all authorized developers



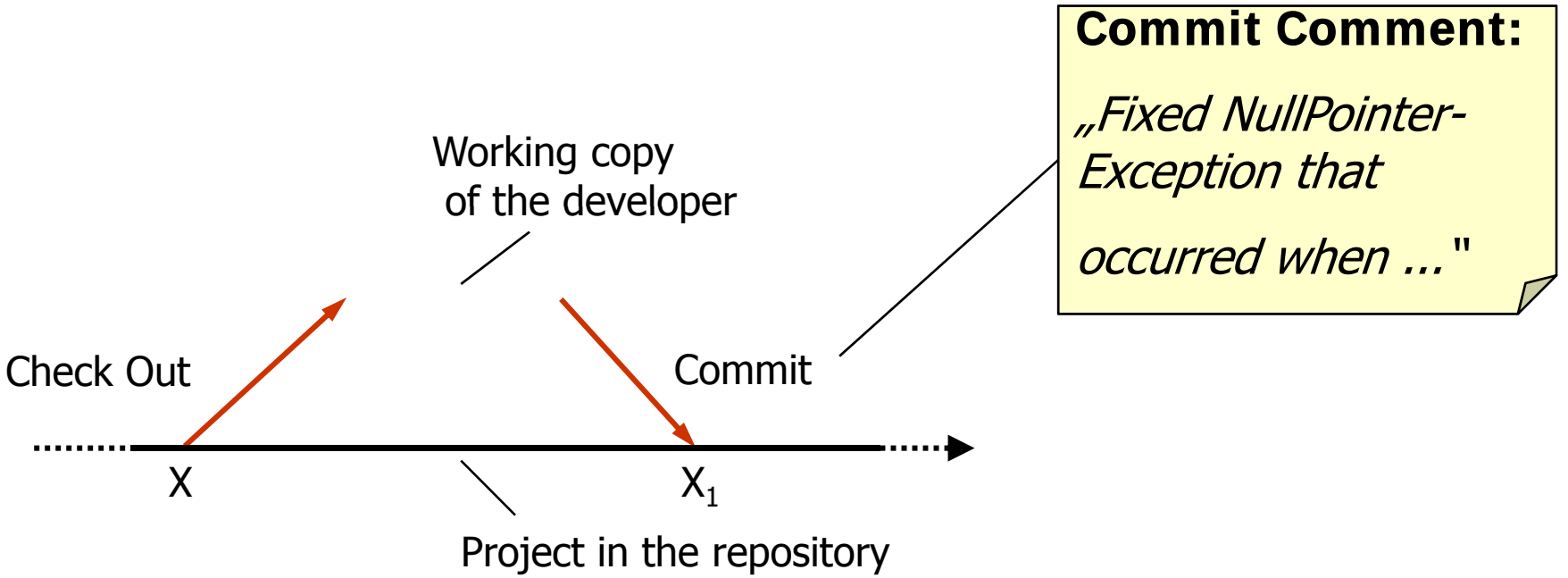
Check-Out
Initial Project Download

- Developers check-out the project
  - get a local **working copy**
- From now on they can work on the project
- Check-out is only done once!
  - Get new versions → **Update**



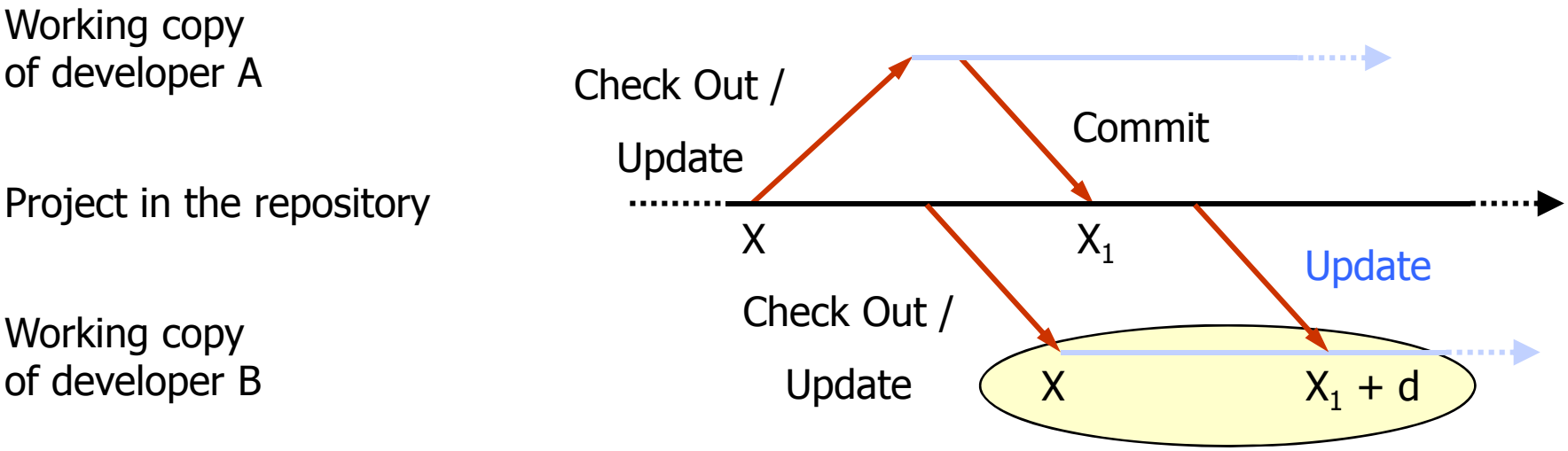
Commit
Writing Changes To The Repository

- The developer changes his working copy
- He adds his changes to the repository (Commit)
- A Commit Comment describes what has changed and why



Update
Get Latest Changes From the Repository

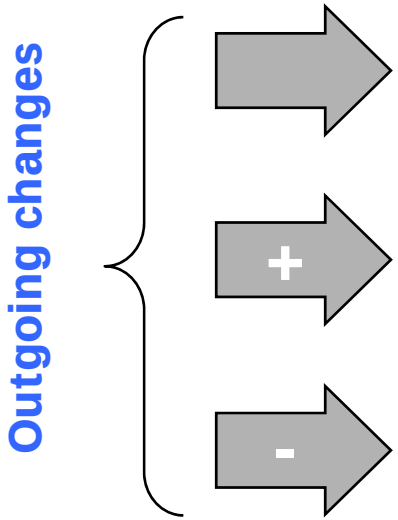
- Updates the working copy with the **current state** of the repository
- Problems
  - Changes are not recapitulated
  - Conflicts are not merged (well)
- Better: → Use „**Synchronize**“



Synchronize
Synchronize The Project

- Right-click on project(s) in the Eclipse navigator
  - Team->Synchronize with Repository
- Motivation
  - Compare the local version and the repository version
    - automated overview comparison
    - automated detailed comparison
  - Decide yourself what to copy
  - Perform selective update or commit
- Automated overview comparison

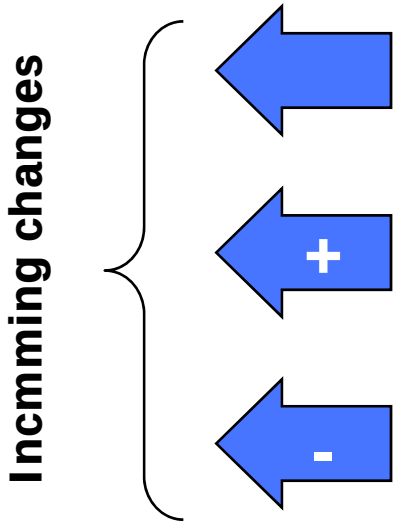
Synchronize: Meaning of Symbols in „Synchronize View“



File has been changed locally and exists in the repository
→ Copy local version into the repository

File has been added locally and doesn‘t exist in the repository
→ Add the file to the repository

File has been deleted locally and exists in the repository
→ Delete the file from the repository

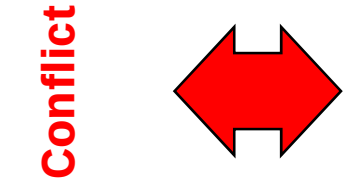


File has been changed in the repository and exists locally
← Copy repository version into the working copy

File has been added to the repository and doesn‘t exist locally
← Add the file to the local working copy

File has been deleted from the repository and exists locally
← Delete the file from the local workingcopy

Synchronize: Synchronize The Project

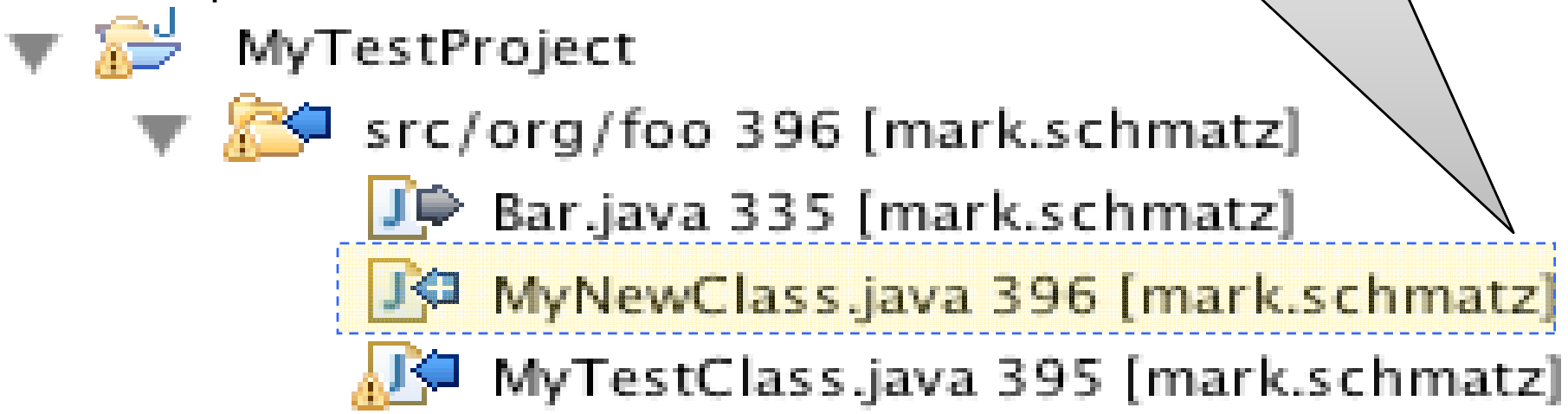


File has been changed in the local working copy **and** in the repository
←→ Manual conflict resolution necessary

- Conflict resolution needs detailed comparison at file level
  - see next 2 slides

Synchronize: Synchronize The Project

- Motivation
  - Compare the local version and the repository version
    - automated overview comparison
    - automated detailed comparison
  - Decide yourself what to copy
  - Perform selective update or commit
- Automated detailed comparison
  - Comparison at file level

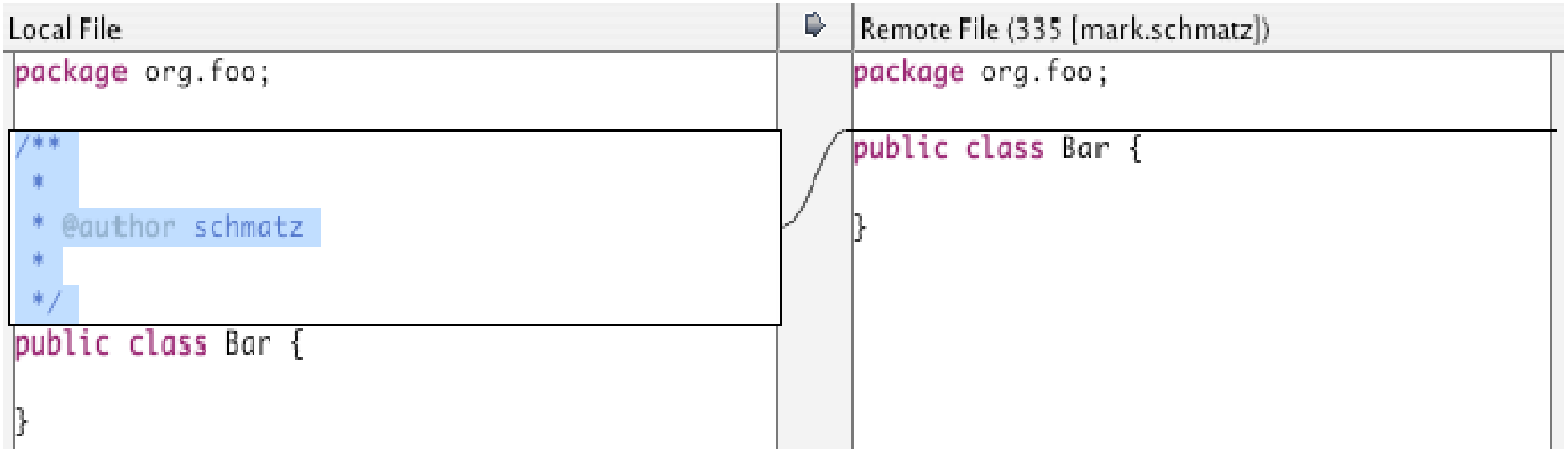


Start detail comparison

by double-clicking on a file ...

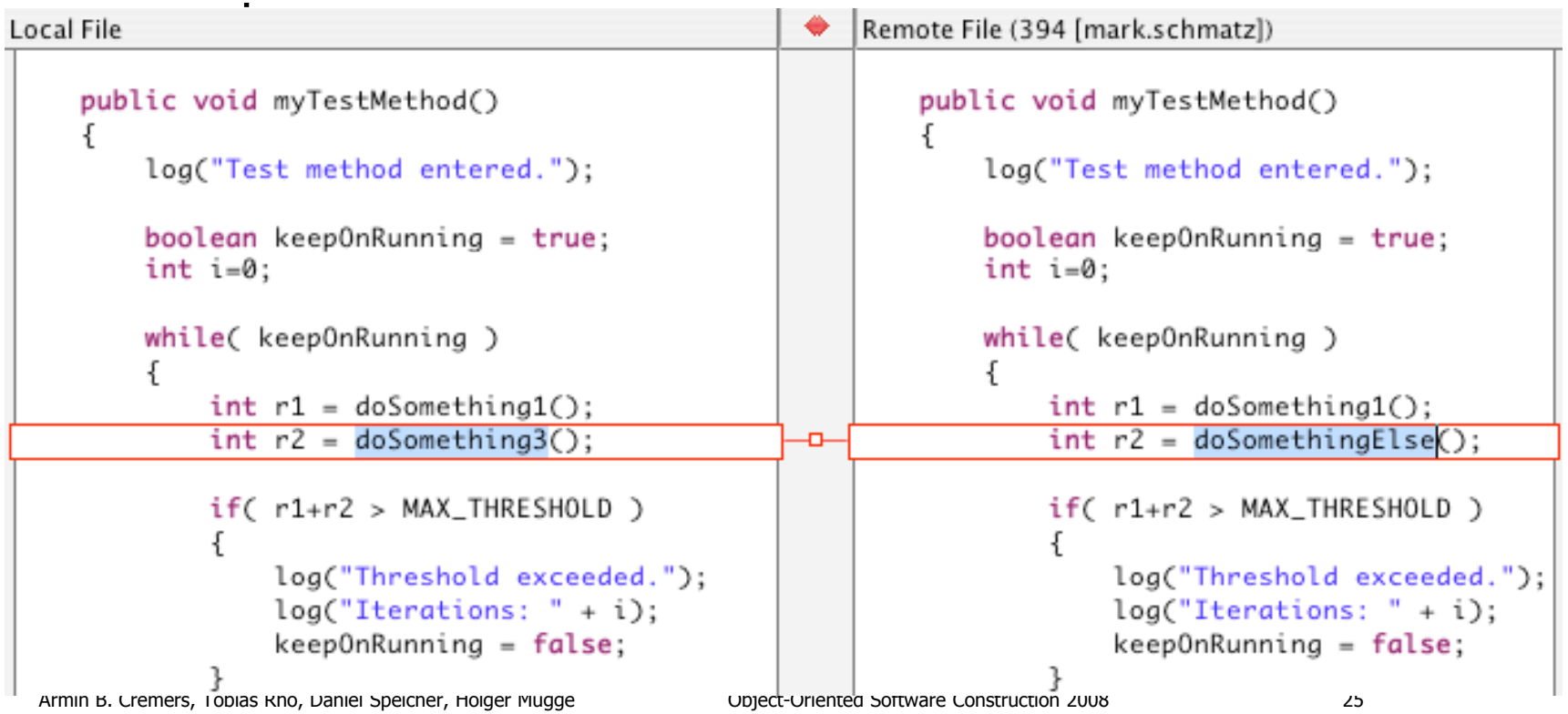
Synchronize: Automated Detailed Comparison

- ... opens the **Compare** window
  - The local workingcopy and the repository versions are compared
  - Selective updates are possible now
    - fine-grained: individual change



Solving Conflicts

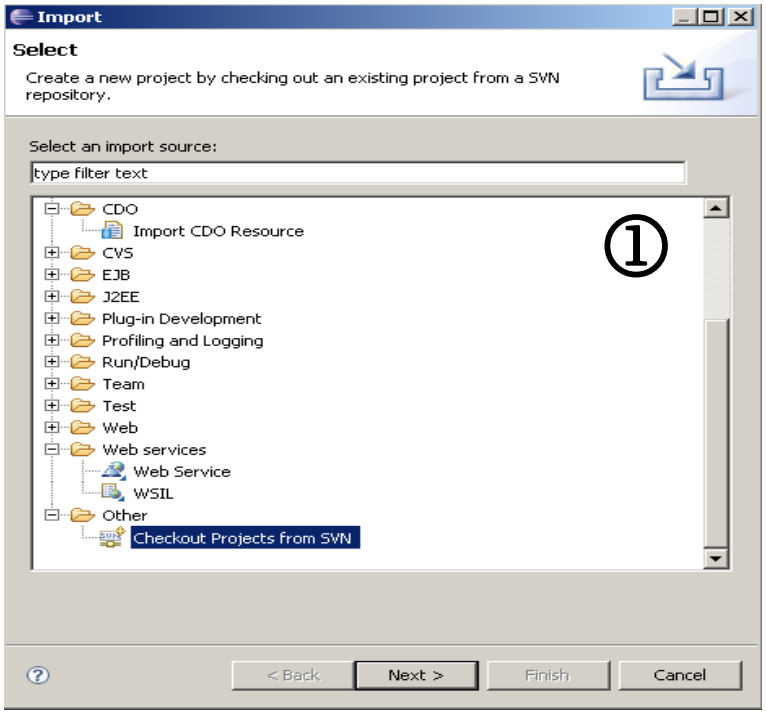
- SVN / CVS helps comparing versions
- Programmer decides what to do
  - **„Overwrite and Update“**: overwrite local changes with repository version
  - **„Overwrite and Commit“:** overwrite repository changes with local version
    - normally requires communication with the author of the other

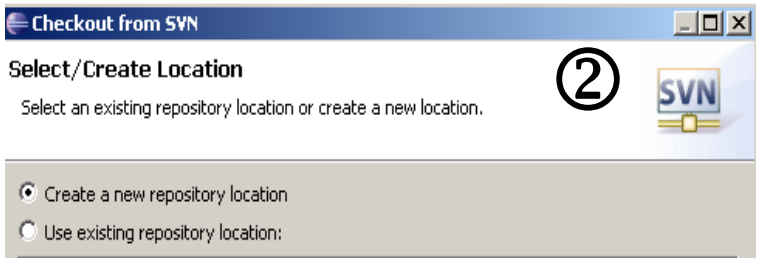


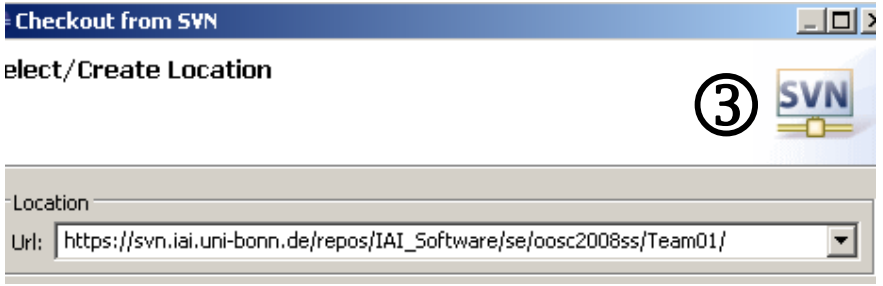
Exercise

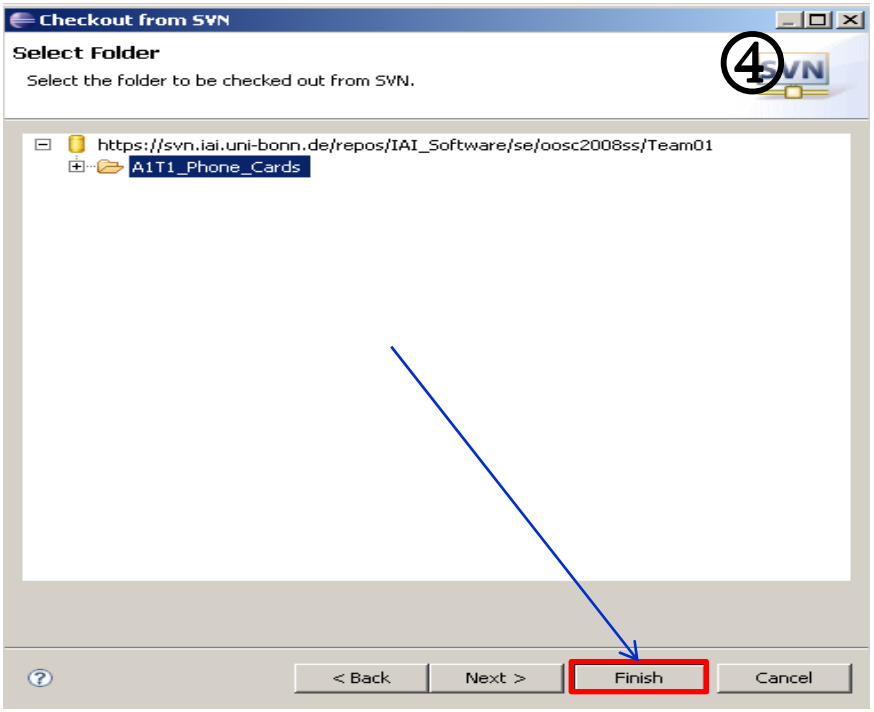
Java Exercise

- Task 0: Check out our uncompleted implementation from the *SVN*.
  - https://svn.iai.uni-bonn.de/repos/IAI_Software/se/oosc2008ss/teamX, X = 1,…,6









Java Exercise

b-it

- DeutschTel Inc. sells 3 different types of phone cards for making cheap long distance calls around the world:

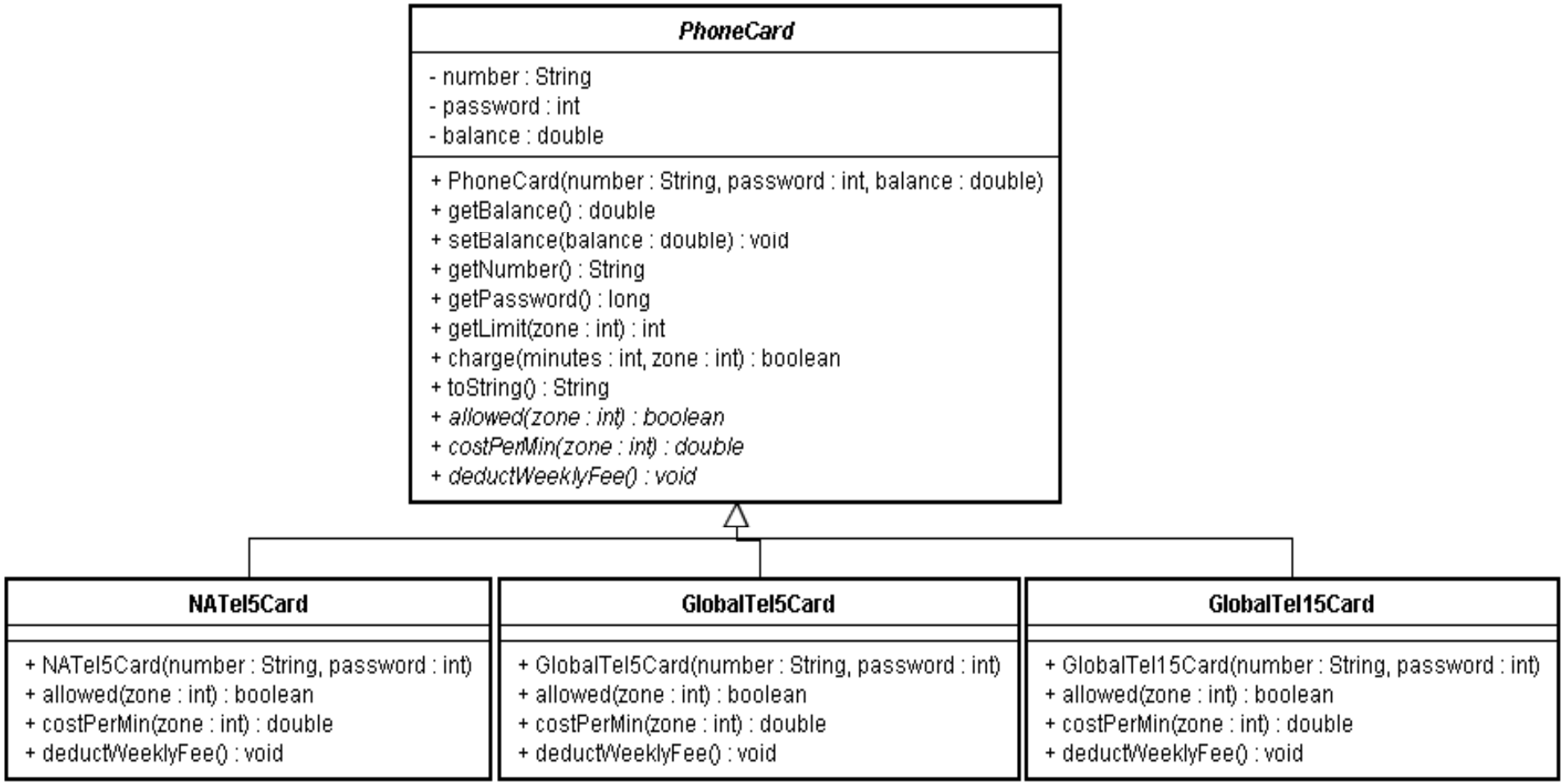
	NATel5	GlobalTel5	GlobalTel15
Africa	N/A	€0.50	€0.35
Asia	N/A	€0.50	€0.35
Australia	N/A	€0.40	€0.25
Canada	€0.05	€0.10	€0.08
Europe	N/A	€0.30	€0.20
Latin America	N/A	€0.40	€0.25
USA	€0.08	€0.15	€0.10

Per minute rates and call zones

	NATel5	GlobalTel5	GlobalTel15
Initial balance	€5.00	€5.00	€15.00
Weekly fee	€0.25	€0.35	€0.50

The initial balance and weekly maintenance fee

Class Diagram for the tutorial



Java Exercise

b-it

- Task 1: Complete the implementation: Go through the classes in the given order, you can find 18 small tasks that you have to accomplish.
  - PhoneCard.java
  - NATel5Card.java
  - GlobalTel5Card.java
  - GlobalTel15Card.java
  - TestApplication.java

Java Exercise

Optional Part

Java Exercise

b-it

- Task 2 (optional): Convert the CallZone class into an enumeration. Modify other classes' code accordingly.

- Hints:
  - See the following slides on Java enumerations
  - Convert the zone type from integer to a CallZone object.
  - Add a function in the CallZone enumeration type to convert from a string to an enumeration object (not mandatory).

Enumerations (Java 5)

- Exhaustively enumerates an inventory of constant objects
- Replaces interfaces containing string/int constants
- Example:

```
public enum AccessKind {
    READ, WRITE, READ_WRITE, NO_ACCESS;
}
```

```
...

void m() {

  AccessKind kind = AccessKind.READ;
  switch(kind) {
    case READ:
      …
    case WRITE:
      …
  }
```

Enumeration members

- Enum constants are objects
- You may add methods and state to enums

```
public enum AccessKind {
    READ(0),
    WRITE(1),
    READ_WRITE(2),
    NO_ACCESS(4);

    int state;

    AccessKind(int state) {
        this.state = state;
    }

    public int getState() {
        return this.state;
    }
}
```

Not accessible for the programmer