Software Engineering

Lecture 1(c):

Design validation & Coding

Outline

(A) Basic class design with annotation — Lect 1(a,b)

(B) Collection class design with annotation

(C) Design validation & Coding — Lect 1(c)

(D) Type hierarchy — Lect 2

References

- Course book: **Chapters 7**

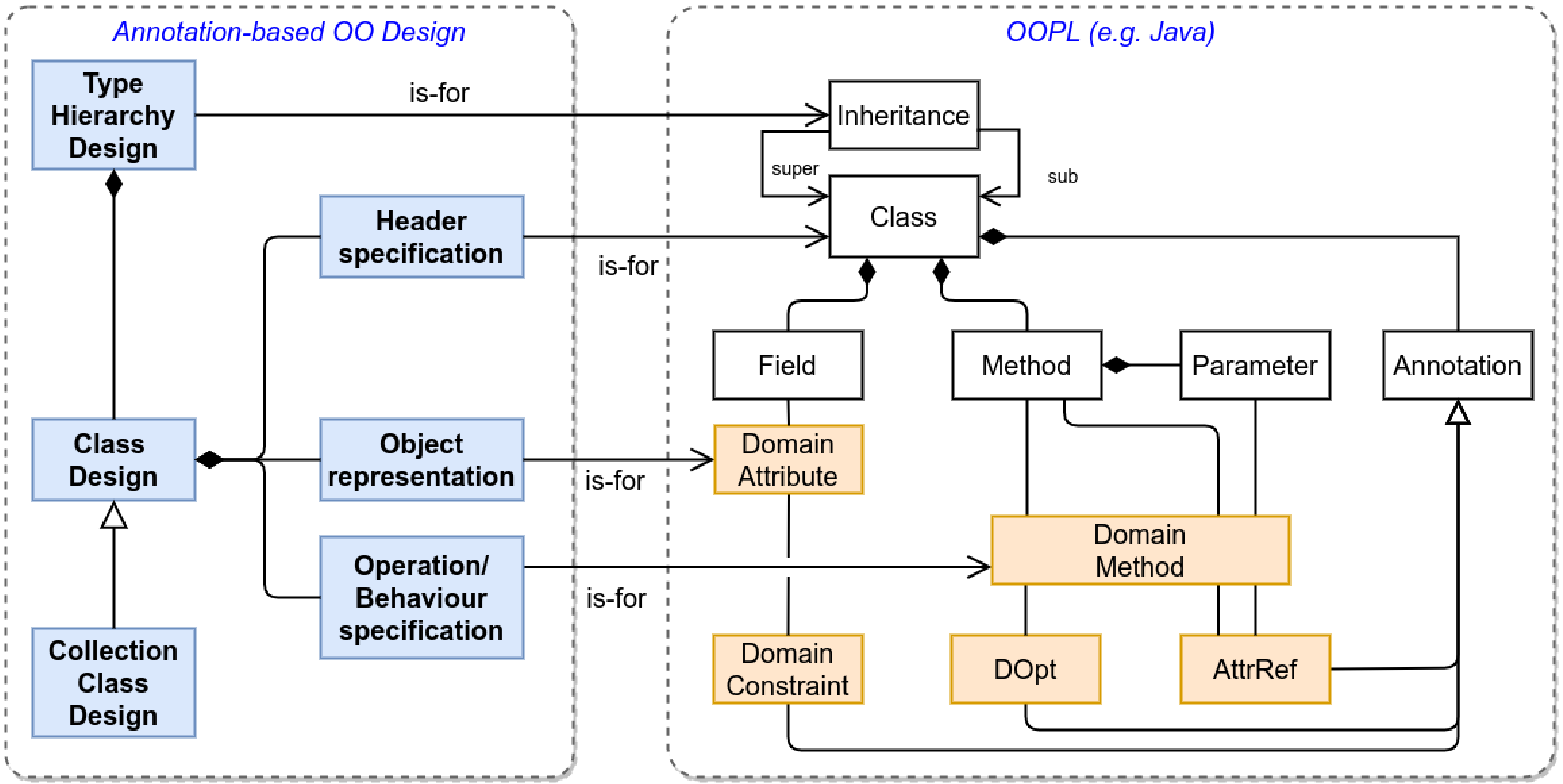
(C.1) Design validation

1) Design review
2) OOPChecker: a design validation tool

Review the design

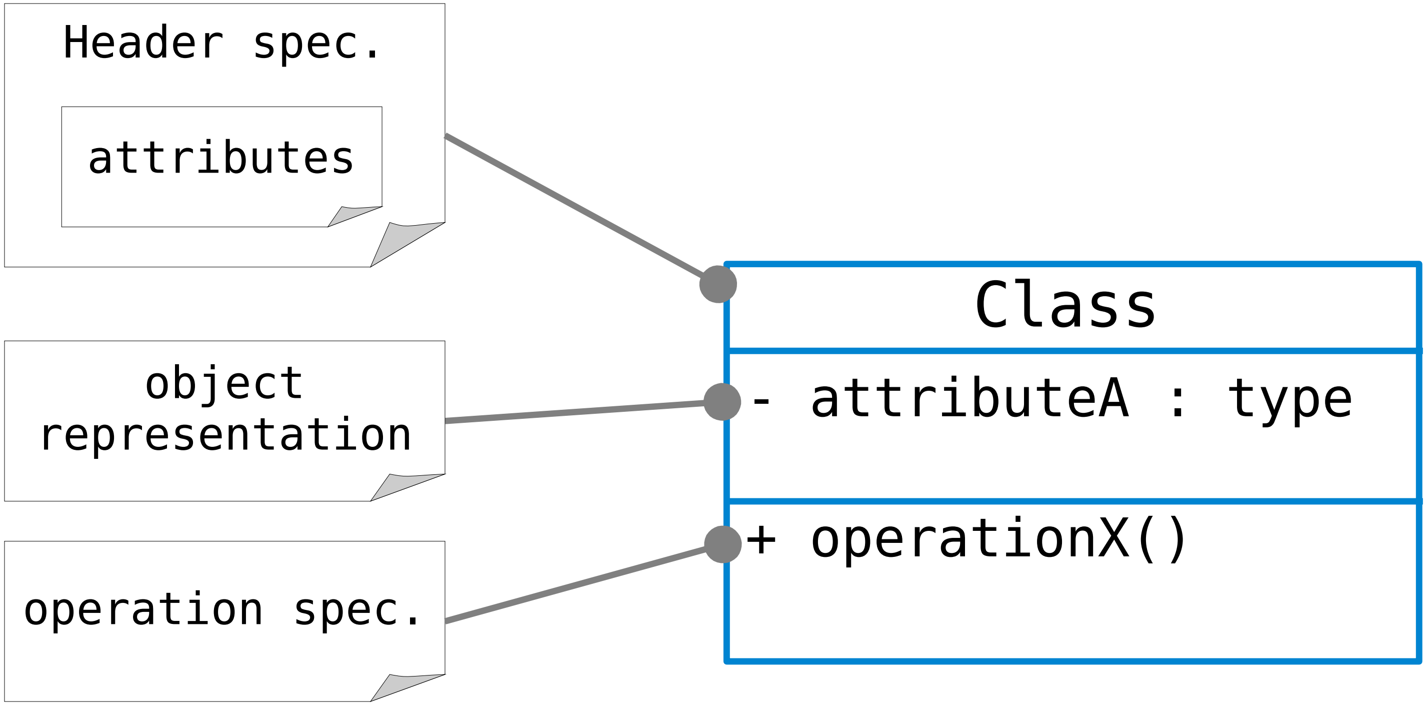
- Helps fix logic errors or make the code compact, before implementation commences:
  - more costly to fix errors once code is written
- Consists in the following checks:
  - check header specification
  - check attribute definitions
  - check object representation
  - check operational specification

Design Method (recap)



(UML class diagram: https://www.uml-diagrams.org/class-diagrams-overview.html)

Review elements



Check header specification

- @overview: states what the abstract concept is
- @attributes: list correct attributes and types
- @object: definition is based on the attribute(s)
- @abstract_properties: domain rules on the attributes are correct

Check attribute definitions

- Formal types are correct
- Concrete types (if any):
  - must be supported by Java
  - must match the formal ones

Check object representation

- Object variables match attribute definitions
- Domain constraint tags match abstract properties

Check operational specification

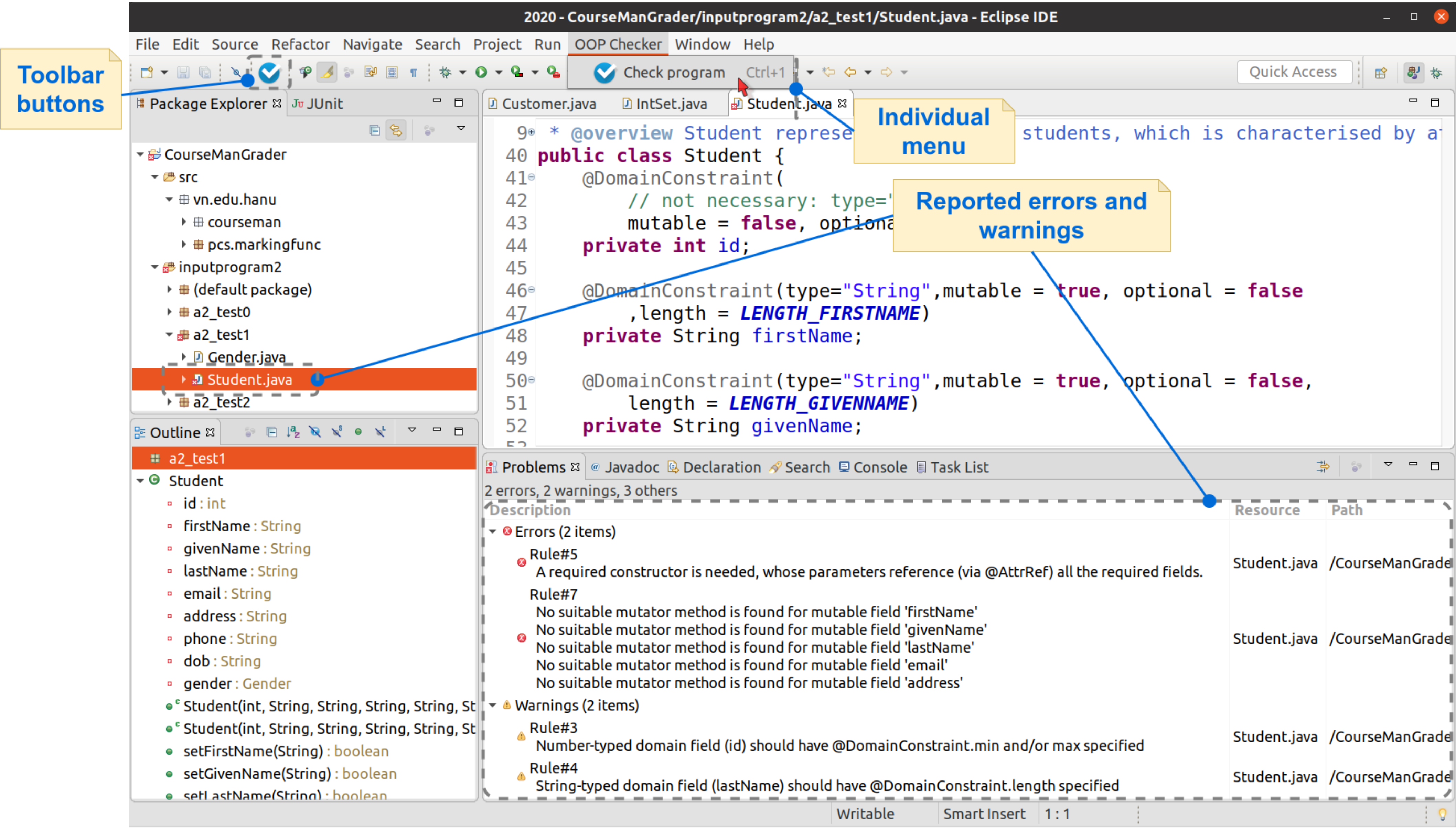
- Ensure that operations result in valid objects
- For each operation, check that:
  - its behaviour is defined with abstract properties in mind
  - it is tagged with suitable annotation(s)
  - it refers to the correct attribute(s)
  - if helper then it needs to be used by other operation(s)

Tool: OOPChecker

- Check the **essential design** of an OOP using its annotation elements
  - Used for the tutorial exercises and assignments
- Design scope:
  - class header
  - fields (attributes)
  - operation header
    - does not check the operation code
- Display design errors and/or warnings at compile time
- Integrated into Eclipse IDE as a plugin:
  - the “Problems” tab displays errors and warnings

Demo

OOPChecker as an Eclipse plugin



Toolbar buttons

Individual menu

Reported errors and warnings

```
 * @overview Student represe                   students, which is characterised by a
public class Student {
    @DomainConstraint(
        // not necessary: type='
        mutable = false, optiona
    private int id;

    @DomainConstraint(type="String",mutable = true, optional = false
        ,length = LENGTH_FIRSTNAME)
    private String firstName;

    @DomainConstraint(type="String",mutable = true, optional = false,
        length = LENGTH_GIVENNAME)
    private String givenName;
```

2 errors, 2 warnings, 3 others

Description	Resource	Path
Errors (2 items)		
Rule#5 A required constructor is needed, whose parameters reference (via @AttrRef) all the required fields.	Student.java	/CourseManGrade
Rule#7 No suitable mutator method is found for mutable field 'firstName' No suitable mutator method is found for mutable field 'givenName' No suitable mutator method is found for mutable field 'lastName' No suitable mutator method is found for mutable field 'email' No suitable mutator method is found for mutable field 'address'	Student.java	/CourseManGrade
Warnings (2 items)		
Rule#3 Number-typed domain field (id) should have @DomainConstraint.min and/or max specified	Student.java	/CourseManGrade
Rule#4 String-typed domain field (lastName) should have @DomainConstraint.length specified	Student.java	/CourseManGrade

Quick user guide

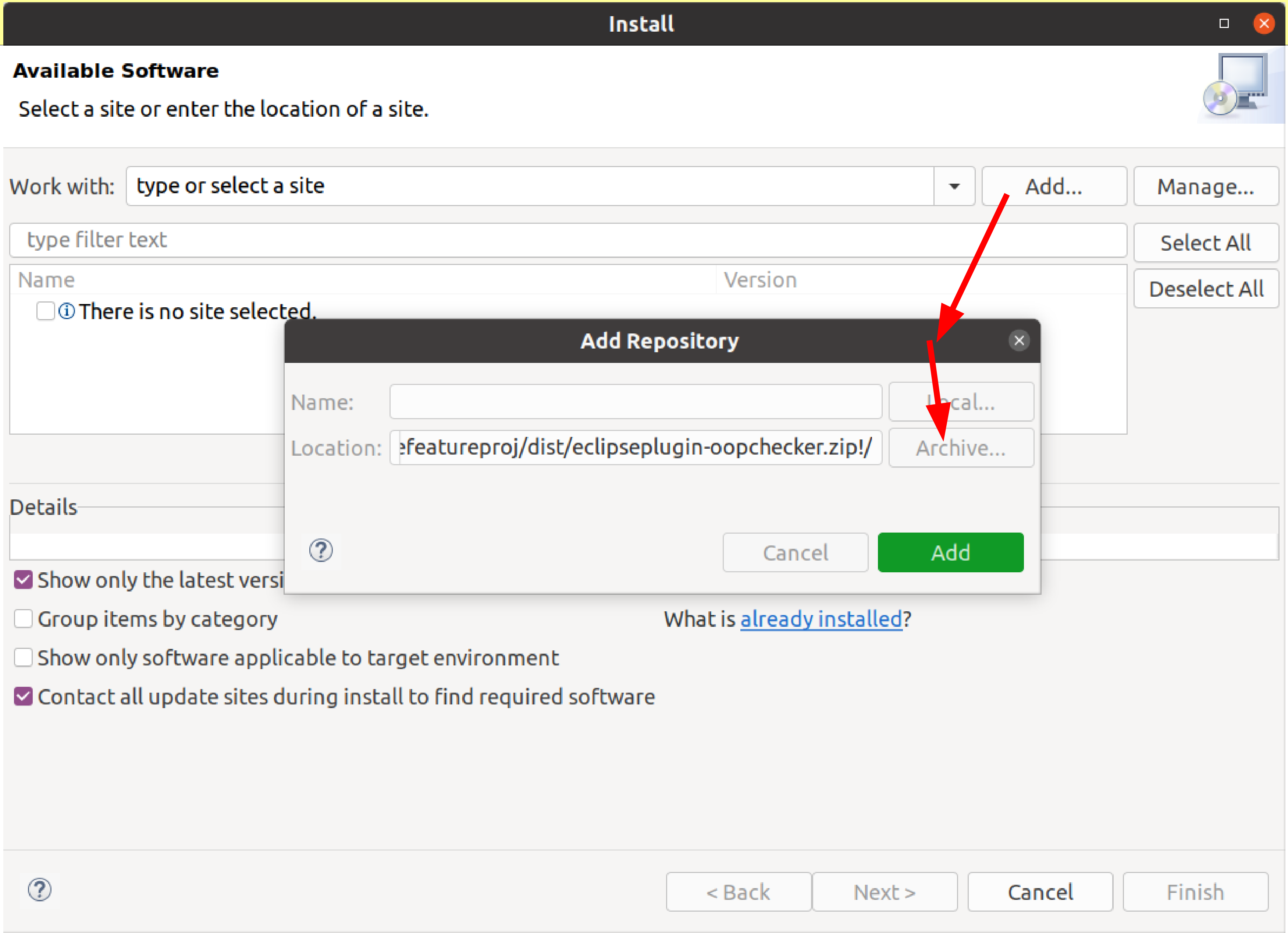
- Select a class of a package
- Click the toolbar button ✓
  - or the menu item “Check program” in the “OOP Checker” menu
- Check the dialog to see if any problems/warnings are reported:
  - If so, go to the “Problems” view to see and fix them
- Subsequent runs on the same file remove the previous problems/warnings (if any)

EXer
cise

Set up and usage

- Download file `eclipseplugin-oopchecker.zip`
- Eclipse: Help/Install New Software...
- Follow the dialogs to complete...

Browse to the plugin archive file



Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

□ ⓘ There is no site selected.

Add Repository

Name:

Location: efeatureproj/dist/eclipseplugin-oopchecker.zip!/

☑ Show only the latest versi

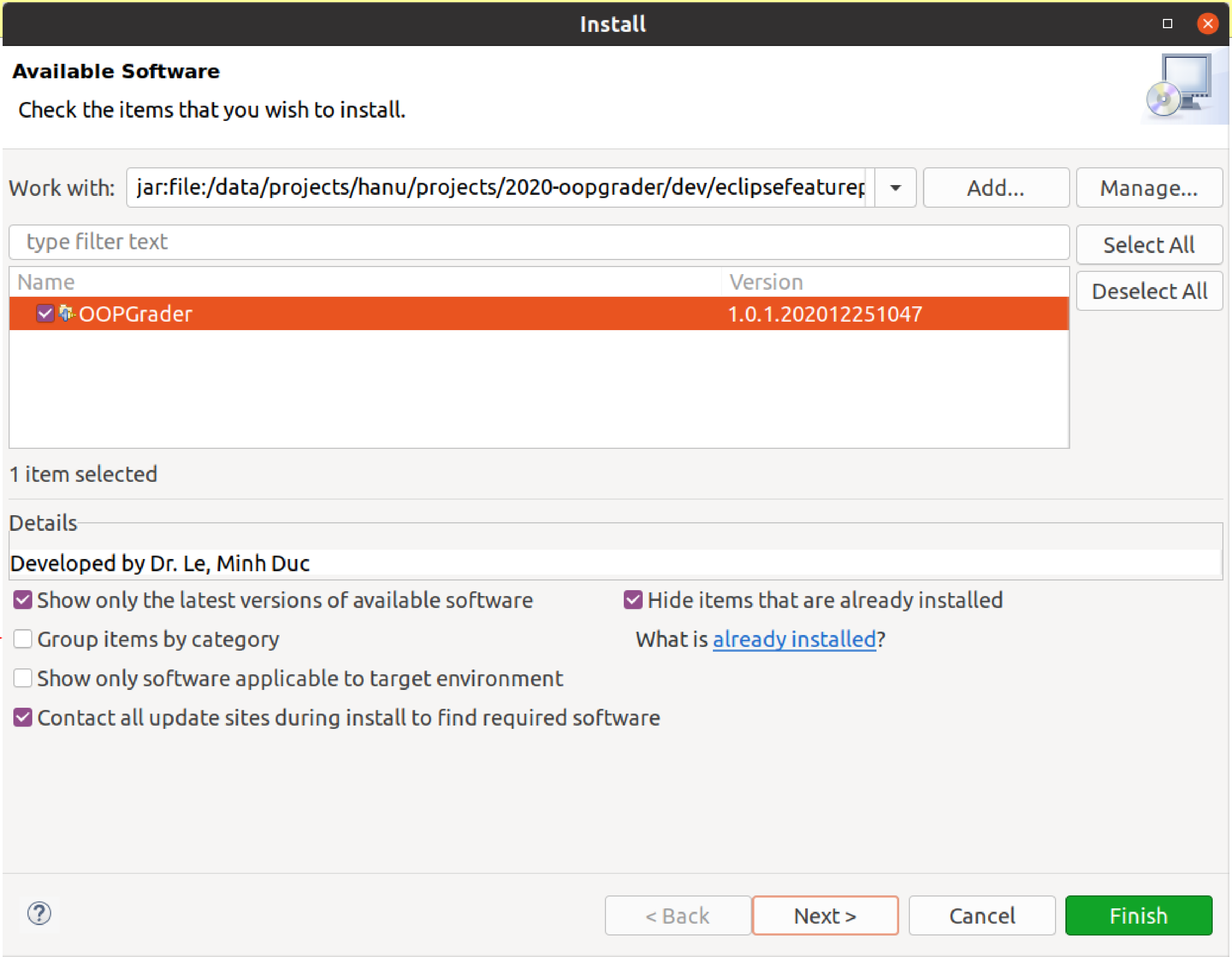
□ Group items by category

What is already installed?

□ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

Untick “Group items by category”



Available Software

Check the items that you wish to install.

Work with: jar:file:/data/projects/hanu/projects/2020-oopgrader/dev/eclipsefeaturep

Name	Version
☑ OOPGrader	1.0.1.202012251047

1 item selected

Details

Developed by Dr. Le, Minh Duc

☑ Show only the latest versions of available software

☑ Hide items that are already installed

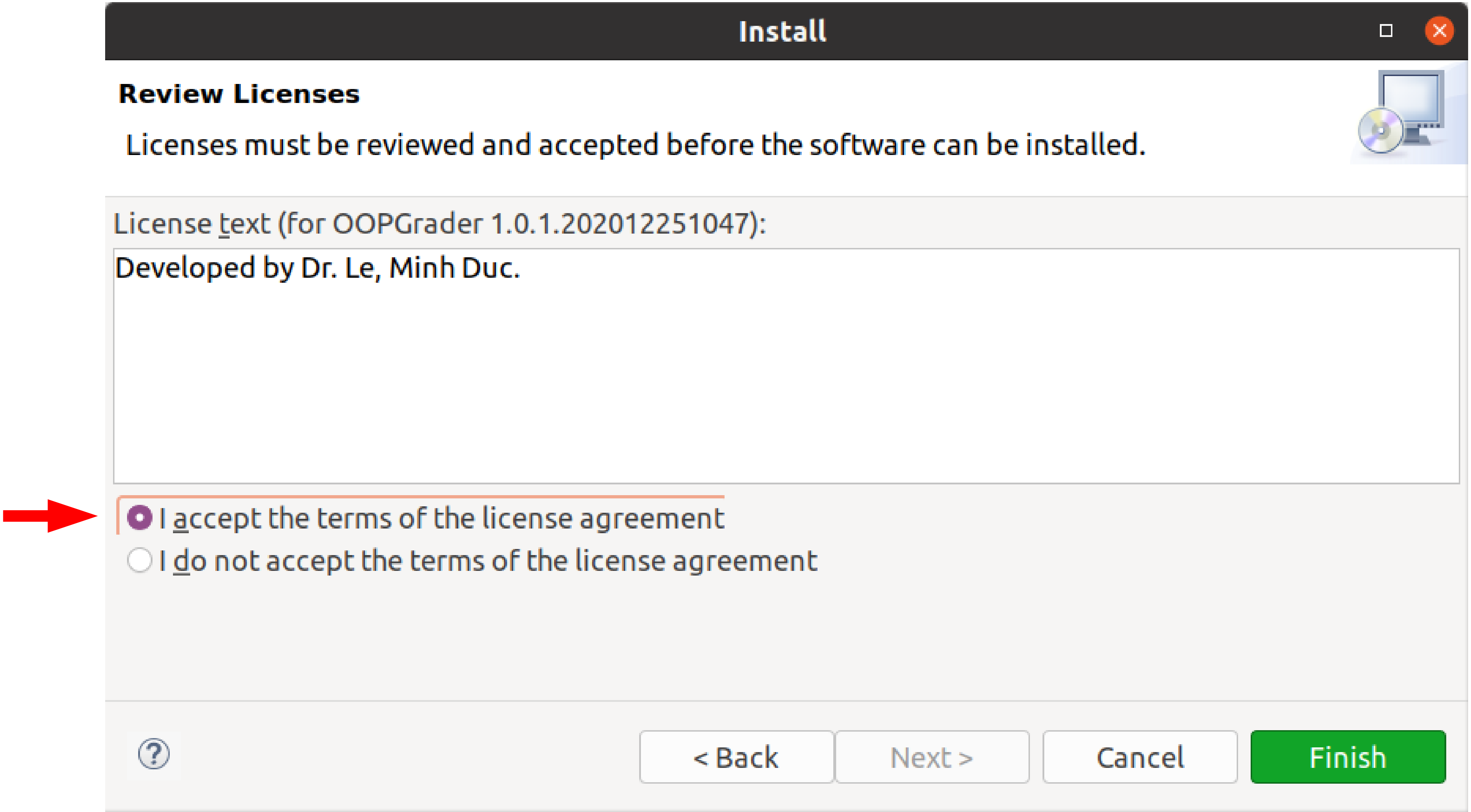
□ Group items by category

What is already installed?

□ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

Accept license agreement



Install

Review Licenses

Licenses must be reviewed and accepted before the software can be installed.

License text (for OOPGrader 1.0.1.202012251047):

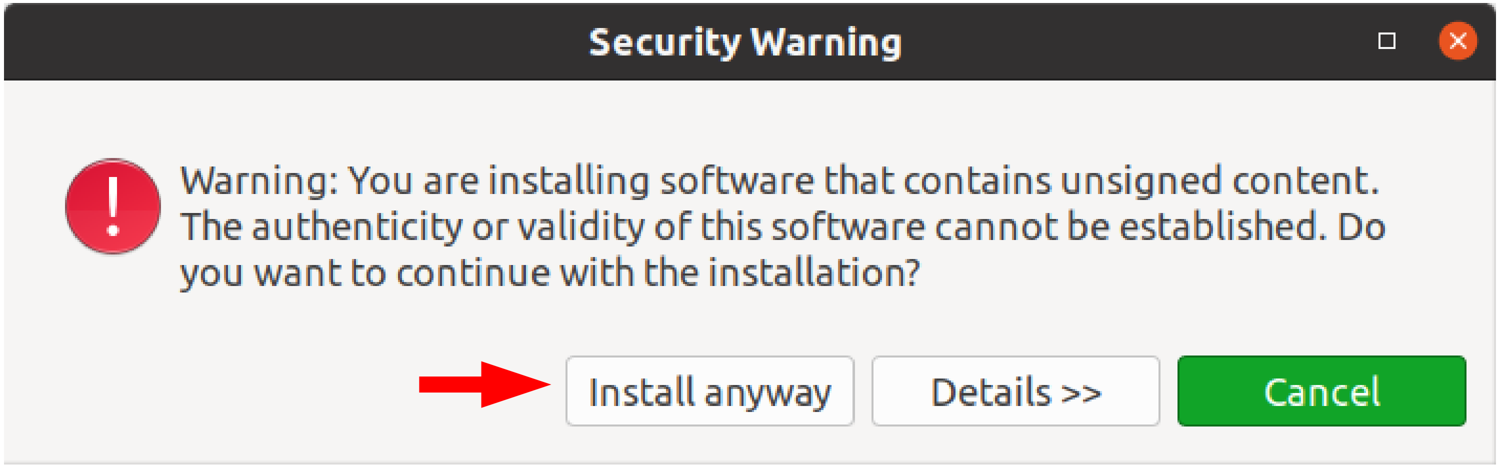
Developed by Dr. Le, Minh Duc.

● I accept the terms of the license agreement

○ I do not accept the terms of the license agreement

Follow the instructions to install...

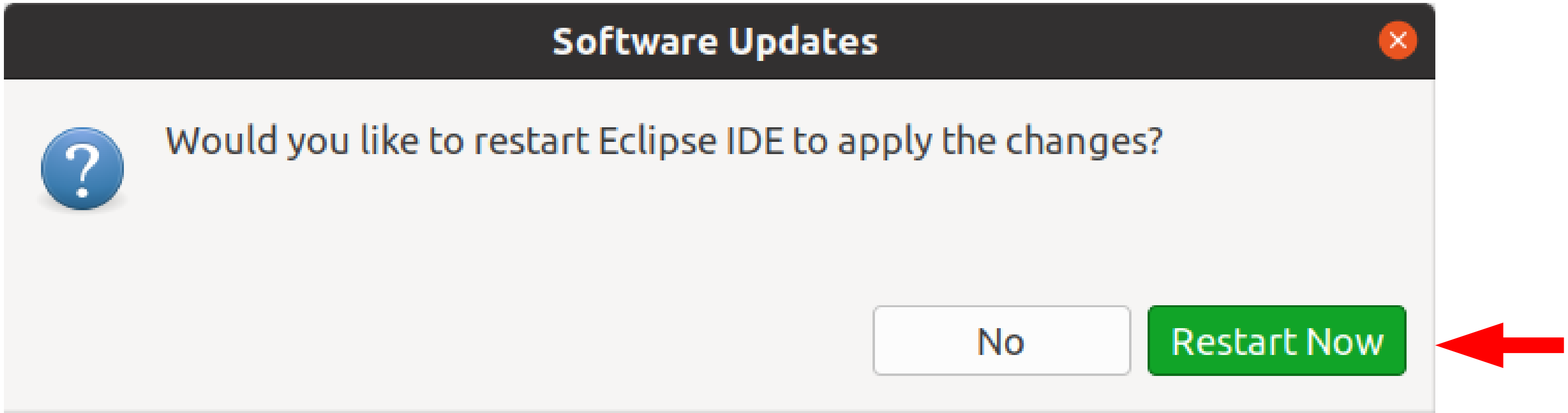
- “Install anyway”



Security Warning

Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the installation?

- “Restart now”



Software Updates

Would you like to restart Eclipse IDE to apply the changes?

(C.2) Coding (implementation)

1) General guidelines
2) Constructors
3) Mutators
4) Observers
5) Default
6) Helpers
7) Examples

General guidelines

- Write code that conforms to the behaviour description
- Make the most of the built-in operations of the chosen data types:
  - e.g. use `Vector` operations to implement `IntSet`
- Use helper operations where needed
  - e.g. to validate input data
- Use the `this` keyword to access other members that have the same name

Constructors

- Focus on the essential constructor
- If input validation fails for some input:
  - throws NotPossibleException with a message containing the constructor name and input value
  - exception is defined in the `utils` package

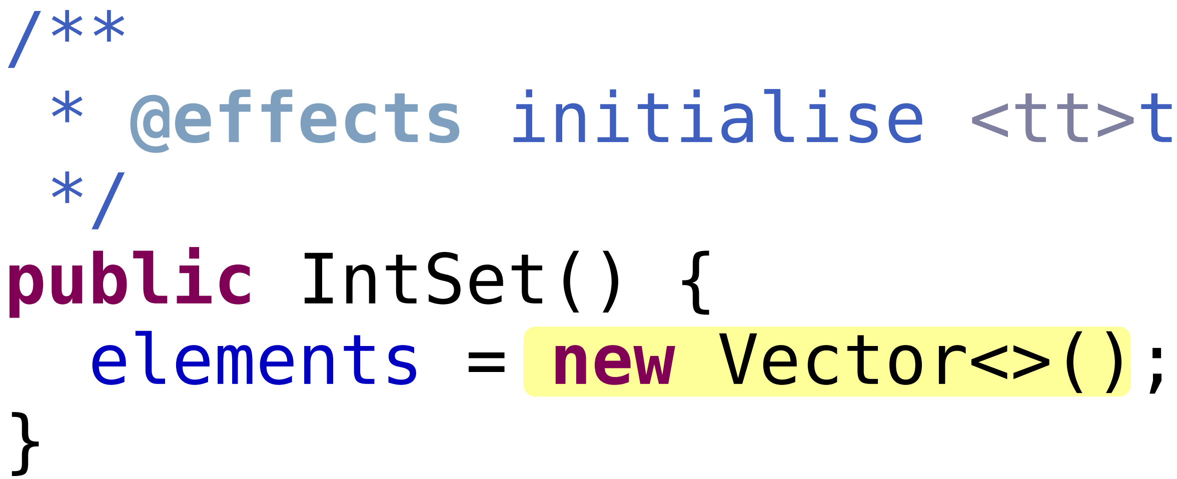
Example: Customer

```
/**
 * ...
 */
public Customer(@AttrRef(”id”) int custID,
       @AttrRef(”name”) String name)
       throws NotPossibleException {

  if (!validateId(custID)) {
    throw new NotPossibleException(
      "Customer.init: Invalid customer id: " + custID);
  }

  if (!validateName(name)) {
    throw new NotPossibleException(
      "Customer.init: Invalid customer name: " + name);
  }

  id = custID;
  this.name = name;
}
```

Using Customer()

```java
Customer c;
try {
  c = new Customer(id,name);
  System.out.println("Created customer: " + c);
} catch (NotPossibleException e) {
   e.printStackTrace();
}
```

Example: IntSet



```
/**
 * @effects initialise <tt>this</tt> to be empty
 */
public IntSet() {
  elements = new Vector<>();
}
```

Using IntSet()

```
IntSet s = new IntSet();
```

Mutators

- Customer
- IntSet

Example: Customer

```
/**
 * @effects <pre>
 *   if name is valid
 *     set this.name=name
 *     return true
 *   else
 *     return false</pre>
 */
@DOpt(type=OptType.Mutator) @AttrRef(”name”)
public boolean setName(String name) {
  if (validateName(name)) {
    this.name = name;
    return true;
  } else {
    return false;
  }
}
```

Example: IntSet

```
/**
 * @modifies <tt>this</tt>
 * @effects <pre>
 *   if x is already in this
 *     do nothing,
 *   else
 *     add x to this, i.e., this_post = this + {x}
 *   </pre>
 */
@DOpt(type=OptType.MutatorAdd)
public void insert(int x) {
  if (getIndex(x) < 0)
    elements.add(x); // auto-boxing
}
```

```
/**
 * @modifies <tt>this</tt>
 * @effects <pre>
 *   if x is not in this
 *     do nothing
 *   else
 *     remove x from this, i.e.
 *   this_post = this - {x}</pre>
 */
@DOpt(type=OptType.MutatorRemove)
public void remove(int x) {
  int i = getIndex(x);
  if (i < 0)
    return;
  elements.set(i, elements.lastElement());
  elements.remove(elements.size() - 1);
}
```

Observers

- Customer
- IntSet

Example: Customer

```
/**
 * @effects return <tt>id</tt>
 */
@DOpt(type=OptType.Observer) @AttrRef(”id”)
public int getId() {
  return id;
}

/**
 *
 * @effects return <tt>name</tt>
 */
@DOpt(type=OptType.Observer) @AttrRef(”name”)
public String getName() {
  return name;
}
```

Example: IntSet

```
/**
 * @effects <pre>
 *  if x is in this
 *    return true
 *  else
 *    return false</pre>
 */
@DOpt(type=OptType.ObserverContains)
public boolean isIn(int x) {
  return (getIndex(x) >= 0);
}

/**
 * @effects return the cardinality of <tt>this</tt>
 */
@DOpt(type=OptType.ObserverSize)
public int size() {
  return elements.size();
}
```

cont'd

```
/**
 * @effects
 *  if this is not empty
 *    return Integer[] array of elements of this
 *  else
 *    return null
 */
@DOpt(type=OptType.Observer)
public Integer[] getElements() {
  if (size() == 0)
    return null;
  else
    return elements.toArray(new Integer[size()]);
}
```

Default

- `toString`:
  - to create a string representation similar to the typical object using the current object state
- `equals`: two techniques
  - compare references: use operator ==
    - the default behaviour of `Object.equals`
  - compare states (common): use the attribute values
    - If value is object then may also need to invoke `equals` on it
    - If value is a collection then need to compare size and elements

Example: Customer

```java
@Override
public String toString() {
  return "Customer:<" + id + "," + name + ">";
}
```

- Using `String.format`

```java
@Override
public String toString() {
  return String.format("Customer:<%d,%s>", id, name);
}
```

```
@Override
public boolean equals(Object o) {
  if (o == null || !(o instanceof Customer))
    return false;

  int yourID = ((Customer) o).id;
  return yourID == id;
}
```

Example: IntSet

```
@Override
public String toString() {
  if (size() == 0)
    return "IntSet:{ }";

  String s = "IntSet:{" +
         elements.elementAt(0).toString();
  for (int i = 1; i < size(); i++) {
    s = s + " , " + elements.elementAt(i).toString();
  }

  return s + "}";
}
```

Example: IntSet (using StringBuilder)

```
@Override
public String toString() {
  if (size() == 0)
    return "IntSet:{ }";

  StringBuilder s = new StringBuilder("IntSet:{");
  s.append(elements.elementAt(0).toString());
  for (int i = 1; i < size(); i++) {
    s.append(" , ")
     .append(elements.elementAt(i).toString());
  }
  s.append("}");
  return s.toString();
}
```

```
@Override
public boolean equals(Object o) {
  if (o == null || !(o instanceof IntSet))
    return false;

  // use Vector.equals to compare elements
  Vector<Integer> yourEls = ((IntSet)o).elements;
  return elements.equals(yourEls);
}
```

Helpers

- repOK
- Data validation
- Utility

Example: Customer.repOK

why?

```
/**
 * @effects <pre>
 *              if this satisfies abstract properties
 *                return true
 *              else
 *                return false</pre>
 */
public boolean repOK() {
  if (!validateId(id) || !validateName(name)) {
    return false;
  }

  return true;
}
```

Example: IntSet.repOK

why?

```
/**
 * @effects  ...
 */
public boolean repOK() {
  if (elements == null) return false;

  for (int i = 0; i < elements.size(); i++) {
    Integer x = elements.get(i);
    /* omitted due to the use of generic
      if (!(x instanceof Integer)) return false;
     */
    for (int j = i + 1; j < elements.size(); j++) {
      if (elements.get(j).equals(x)) return false;
    }
  }
  return true;
}
```

Example: Customer validation

```
/**
 * @effects <pre>
 *            if id is valid
 *              return true
 *            else
 *              return false
 *          </pre>
 */
private boolean validateId(int id) {
  if (id < MIN_ID) {
    return false;
  }
  return true;
}
```

```
/**
 * @effects <pre>
 *             if name is valid
 *               return true
 *             else
 *               return false
 *           </pre>
 */
private boolean validateName(String name) {
  if (name == null || name.length() > LENGTH_NAME) {
    return false;
  }
  return true;
}
```

Utility

- `IntSet.getIndex`
- [!] `Rat.reduce`

Example: IntSet.getIndex

```
/**
 * @effects <pre>
 *  if x is in this
 *    return the index where x appears
 *  else
 *    return -1</pre>
 */
private int getIndex(int x) {
  for (int i = 0; i < elements.size(); i++) {
    if (x == elements.get(i))
      return i;
  }

  return -1;
}
```

Application examples

- Wrapper classes: a bit more
- Integers: use IntSet
- CRM: use Customer

More about wrapper classes

- Wrapper class objects can be created using:
  - auto-boxing
  - constructor operation
  - `parse`*X* operation (*X* is the primitive type: Int, Long, ...)
- Conversion to primitive can be performed using:
  - auto-unboxing
  - *x*`Value` operation (*x* is the primitive type: int, long, ...)

Example

`chap5_2.apps.Wrappers`

- Create an Integer object
- Perform integer and conversion operations

Integers

`chap5_2.apps.Integers`

- Create an `IntSet` from a given array of integers
- Print set using `toString`

Customers

`chap5_2.apps.CRM`

- Create some Customer objects
- Use a static (class) variable to generate object ids
- Use try...catch to handle object creation error

Q & A