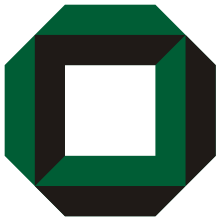
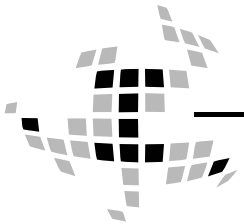


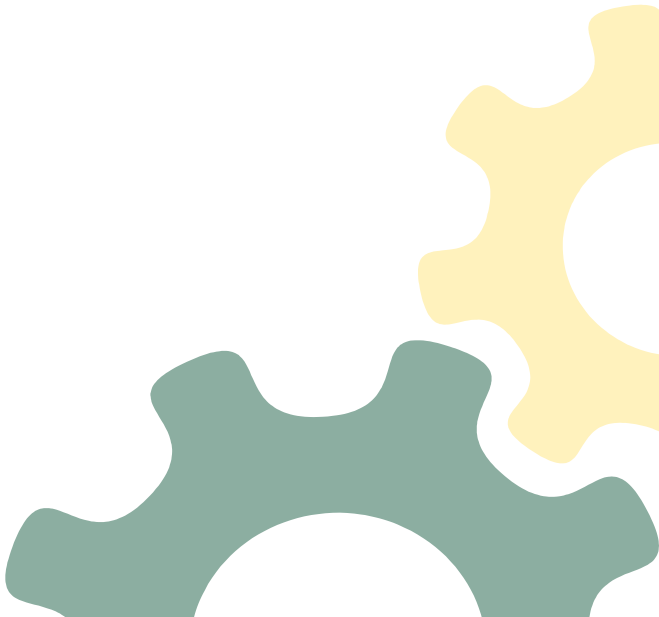
Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Softwaretechnik 1
Übung 2

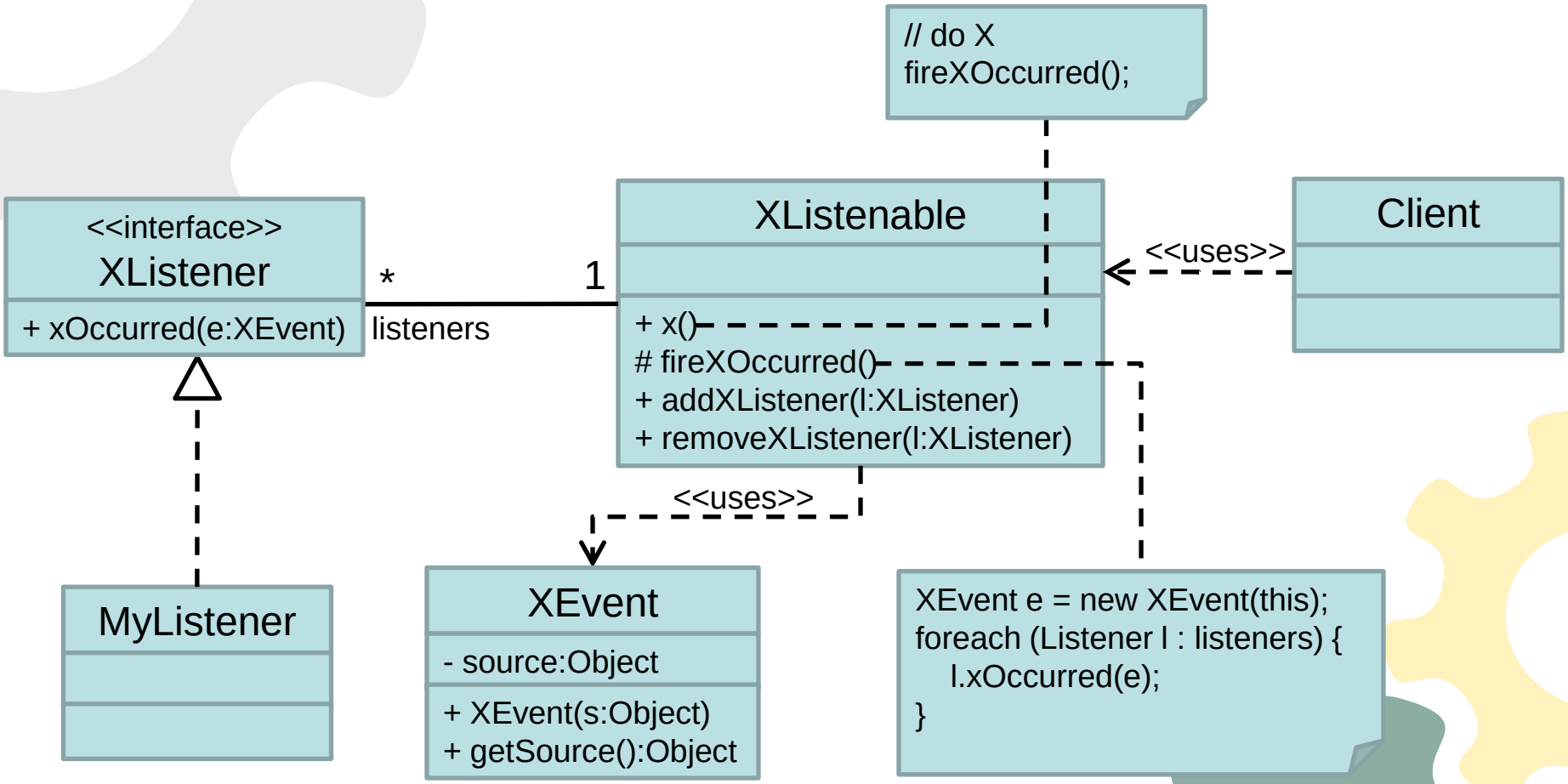
25.5.2009

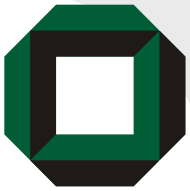


Fakultät für **Informatik**
Lehrstuhl für Programmiersysteme



Aufgabe 1 – Klassendiagram

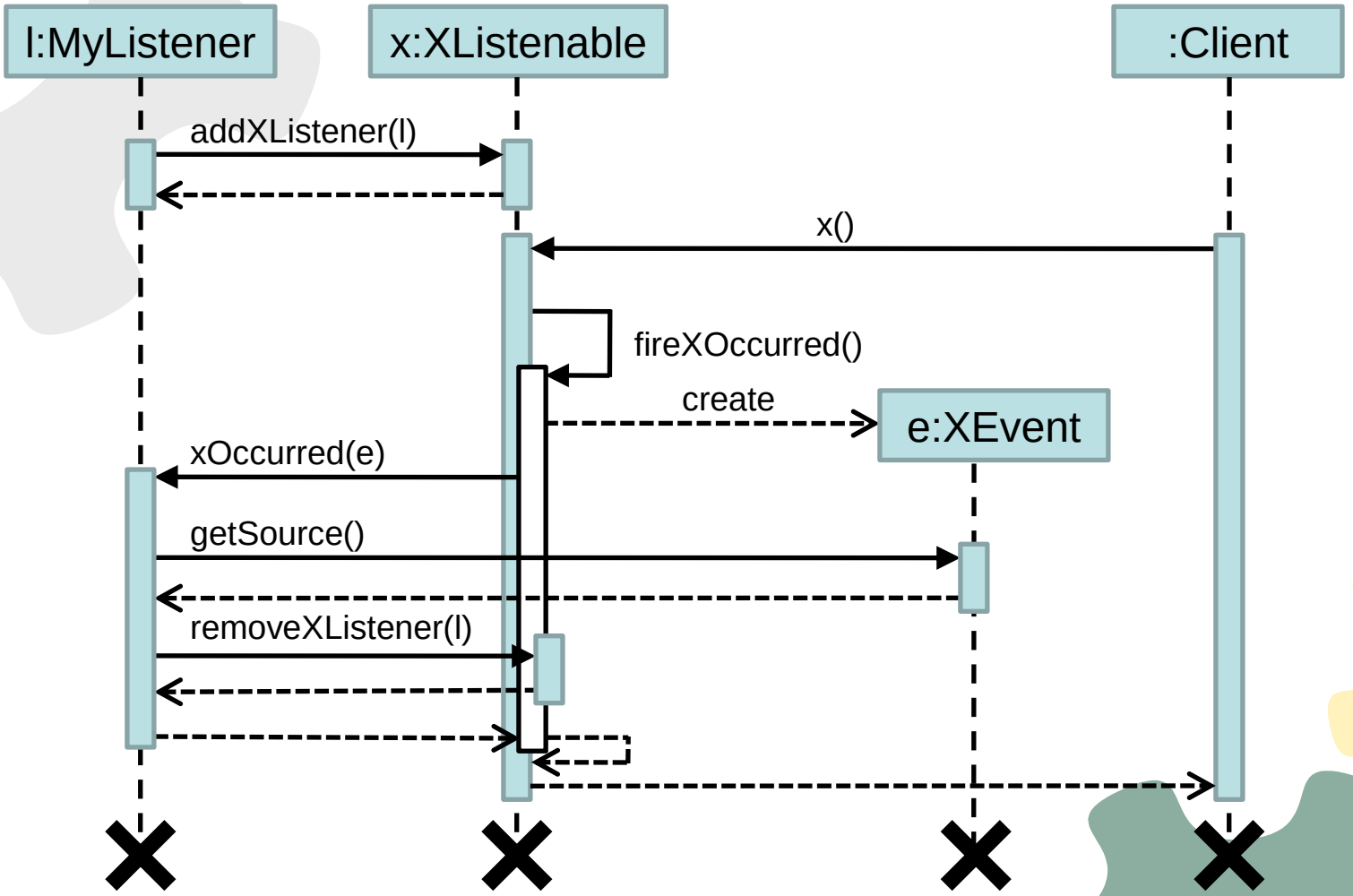




Aufgabe 1 – Szenario

Ein Objekt l der Klasse MyListener möchte exakt ein Mal darüber informiert werden, wenn in dem Objekt x vom Typ XListenable die Methode x() aufgerufen wird. Dazu meldet sich das Objekt l als Beobachter bei Objekt x an. Nach unbestimmter Zeit ruft ein Objekt der Klasse Client bei Objekt x die Methode x() auf. Daraufhin wird das Objekt l durch das Senden eines XEvents e über das Auftreten von x() informiert, greift auf die Quelle des Ereignis-Objekts e zu und meldet sich direkt im Anschluss als Beobachter von x ab.

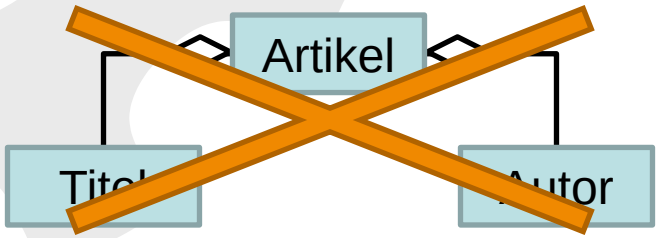
Aufgabe 1 – Sequenzdiagramm

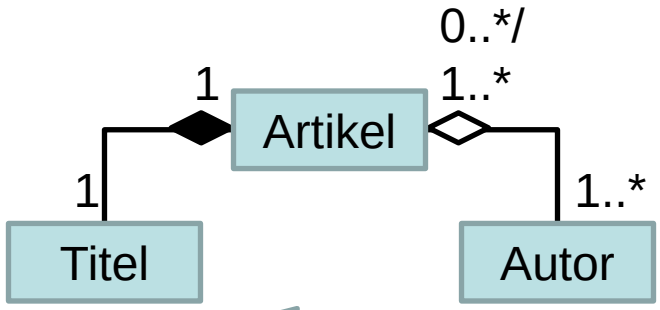


Aufgabe 2a)

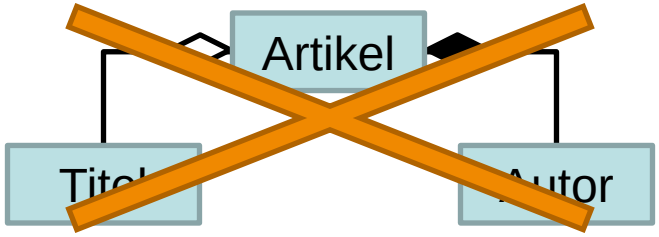
Welche der folgenden Modellierungen ist richtig, wenn man geeignete Multiplizitäten ergänzt? Markieren Sie die richtige Modellierung und ergänzen Sie sinnvolle Multiplizitäten. Begründen Sie ihre Entscheidung. Ohne Begründung gibt es keine Punkte.

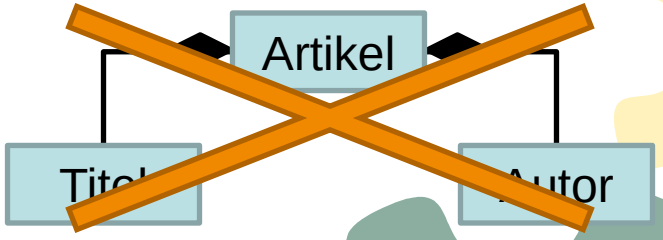
Aufgabe 2a)

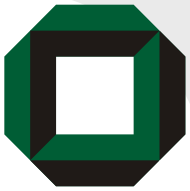




Ein Artikel hat genau einen Titel. Der Titel hat ohne Artikel keine Existenzberechtigung. Ein Artikel wird von einem oder mehreren Autoren geschrieben. Ein Autor kann an mehreren Artikeln beteiligt sein.



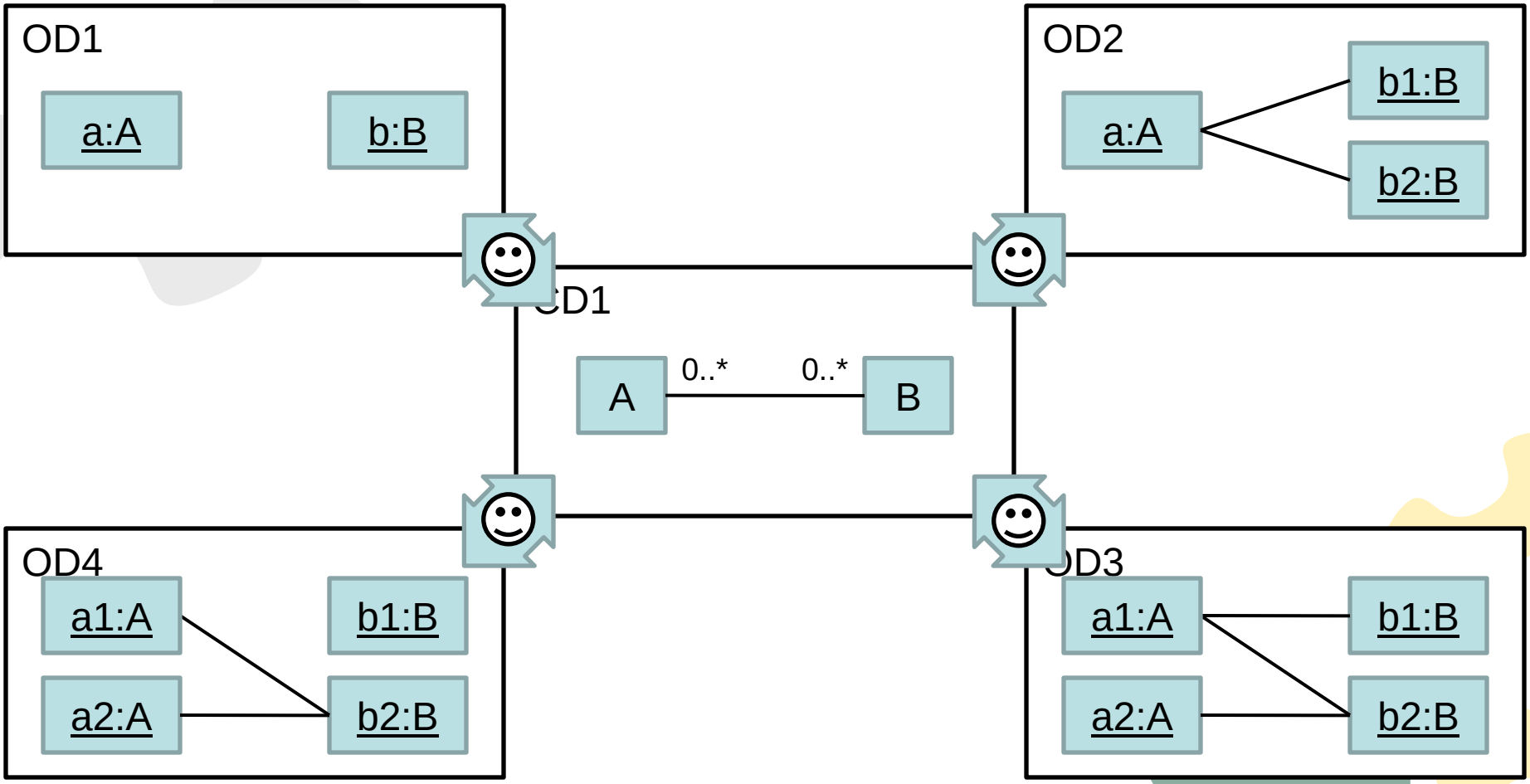




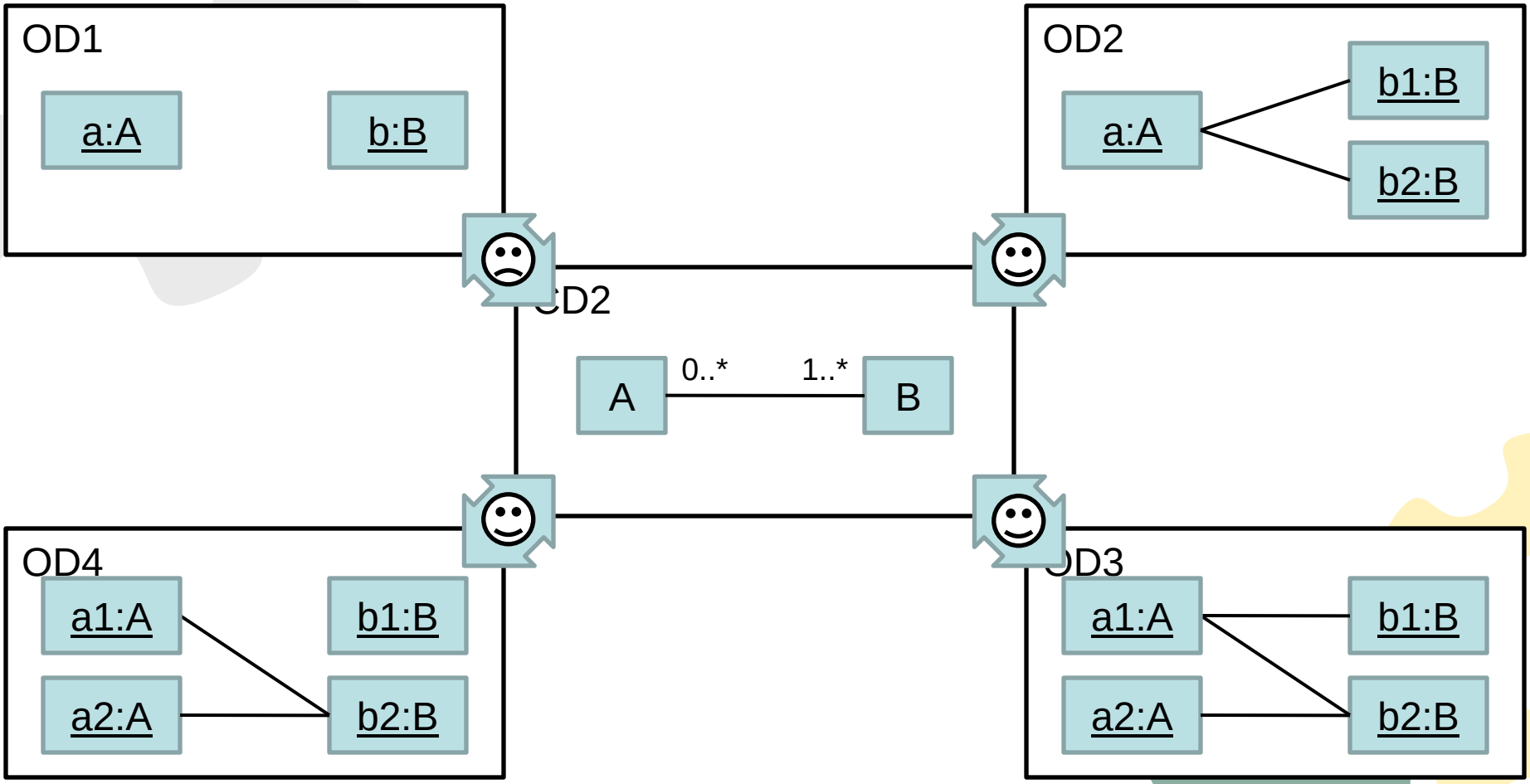
Aufgabe 2b)

Geben Sie für jede Kombination der Objekt- und Klassendiagramme an, ob sie zusammen passen oder nicht. Wählen Sie zur Darstellung der Zugehörigkeit von Objektdiagramm zu Klassendiagramm eine geeignete Darstellung.

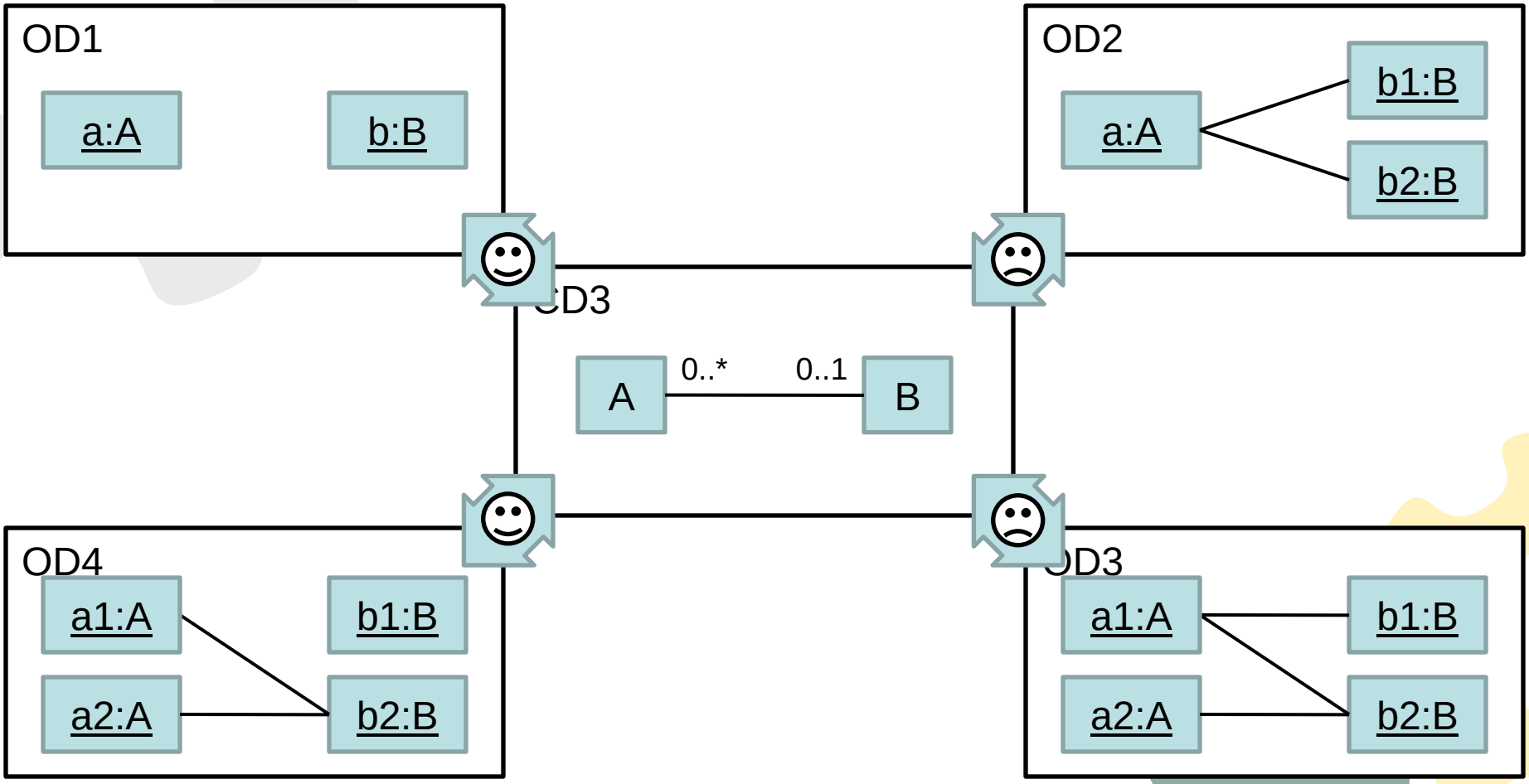
Aufgabe 2b)
Klassendiagramm 1



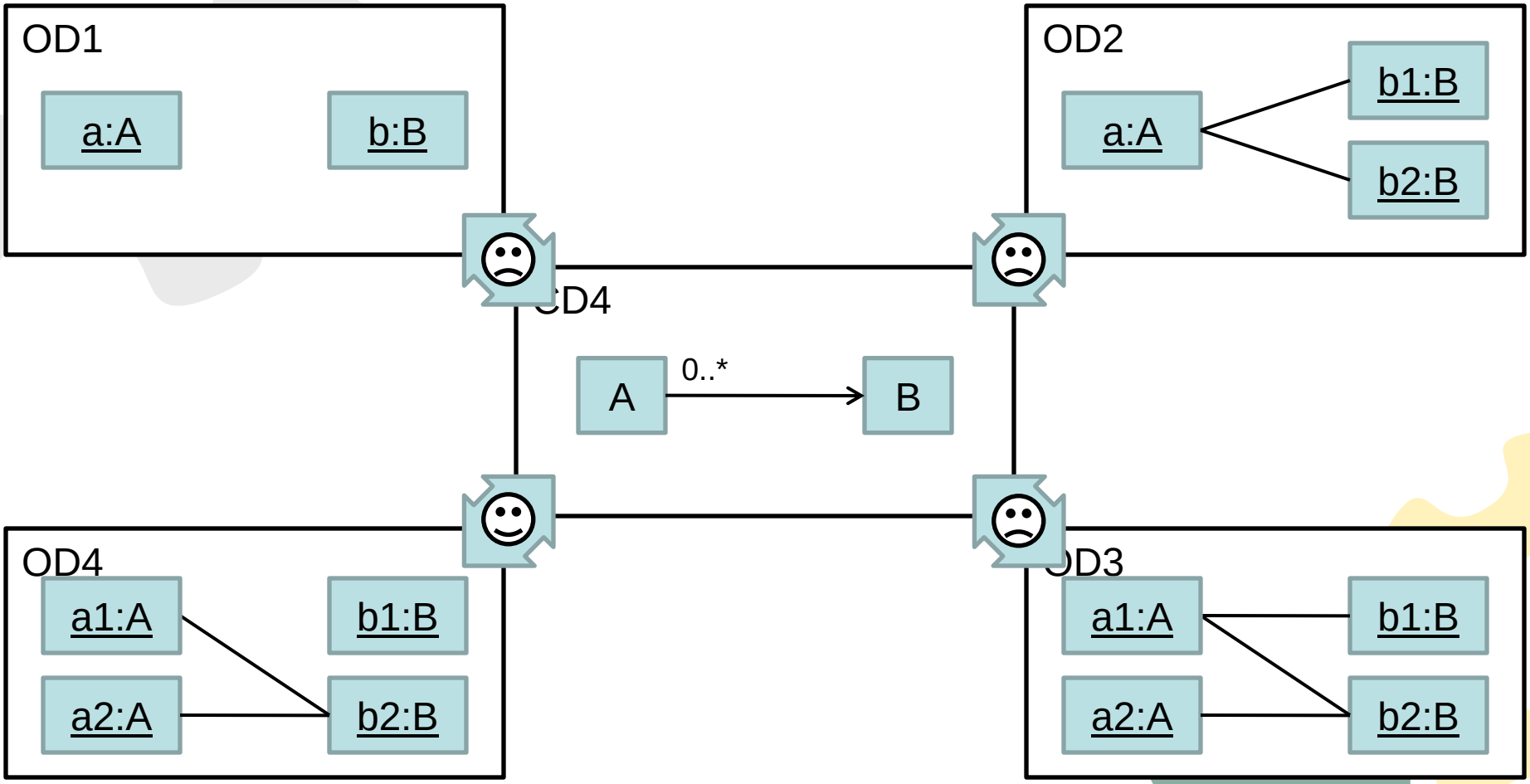
Aufgabe 2b)
Klassendiagramm 2

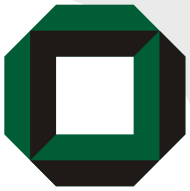


Aufgabe 2b)
Klassendiagramm 3



Aufgabe 2b)
Klassendiagramm 4





Aufgabe 3
Parametervarianz in Java

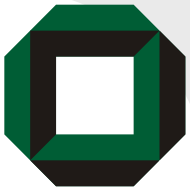
- Welche Varianz wäre nach dem Substitutionsprinzip in Java zulässig?
- Welche Varianz ist in Java tatsächlich (bei vererbten Methoden) zulässig, welche nicht?
- Untersuchen Sie jeweils Ko- und Kontravarianz.
- Hinweise:
  - mehrere Fragen in einer Aufgabe → alle beantworten!
  - Tipp: abhaken

Aufgabe 3
Parametervarianz in Java

- Tabelle aus der Vorlesung

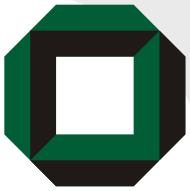
Eingabeparameter	Kontravarianz
Ausgabeparameter (auch Rückgabewert und Ausnahmen)	Kovarianz
Parameter, die gleichzeitig Ein- und Ausgabeparameter sind	Invarianz

- Folgende Fragen sind empirisch zu beantworten:
  Lässt Java bei geerbten Methoden…
  - Kontravarianz der Eingabeparameter zu?
  - Kovarianz der Eingabeparameter zu?
  - Kontravarianz der Rückgabewerte zu?
  - Kovarianz der Rückgabewerte zu?



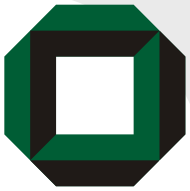
Aufgabe 3
Parametervarianz in Java

- Bauen Sie eine Klasse A mit vier Methoden:
  - Zwei Methoden (void) mit einem Eingabeparameter
  - Zwei Methoden mit einem Rückgabewert (kein Eingabeparameter)
- … und eine Klasse B die von A erbt und alle vier Methoden überschreibt:
  Versuchen Sie, …
  - den Eingabeparameter allgemeiner zu deklarieren
  - den Eingabeparameter spezieller zu deklarieren
  - den Rückgabewert allgemeiner zu deklarieren
  - den Rückgabewert spezieller zu deklarieren



Problematik: Überschreiben oder Überladen?

- Wie stellt man sicher, dass die in B deklarierten Methoden die Methoden aus A überschreiben und nicht überladen?
- Lösungen:
  - Abstrakte Methoden: machen Sie die vier Methoden in A abstrakt, die Klasse B aber nicht-abstrakt
  - Markierung `@Override`: *„Indicates that a method declaration is intended to override a method declaration in a superclass. If a method is annotated with this annotation type but does not override a superclass method, compilers are required to generate an error message.“* (ab Java 1.5)



Aufgabe 3
Parametervarianz in Java

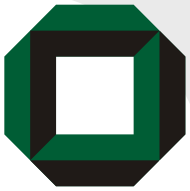
```
class X {}   class Y extends X {}   class Z extends Y {}

abstract class A {
  abstract public void a1(Y y);
  abstract public void a2(Y y);
  abstract public Y    b1(   );
  abstract public Y    b2(   );
}

class B extends A {
  @Override public void a1( X y ) {}
  @Override public void a2( Z y ) {}
  @Override public X b1() { return null; }
  @Override public Z b2() { return null; }
}
```

The method a1(X) of type B must override or implement a supertype method

The method a2(Z) of type B must override or implement a supertype method

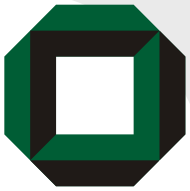
The return type is incompatible with A.b1()

Geht!



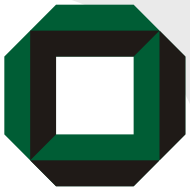
Aufgabe 3
Parametervarianz in Java

- Antwort: Java unterstützt bei vererbten Methoden nur Kovarianz der Rückgabewerte.



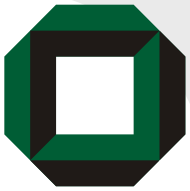
Aufgabe 4

- Anforderungen an die Anwendung:
  - 1 Menüleiste, bestehend aus
    - Mindestens 2 Menüs
      - Mit Namen und mnemonischem Buchstaben
    - Pro Menü mindestens 3 Menüeinträge
      - Mit Namen, Icon, Tastaturkürzel und mnemonischem Buchstaben
  - 1 Werkzeugleiste, welche die selben Funktionen wie die Menüleiste anbietet.
    - Alle Einträge der Werkzeugleiste müssen ein Icon sowie einen Tooltip besitzen
  - Internationalisierung der Anwendung
    - Internationalisierung aller in der Anwendung vorkommenden Texte
    - Unterstützte Sprachen: Mindestens Deutsch und Englisch



Aufgabe 4

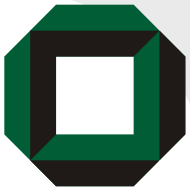
- Sie können Actions verwenden, um sicherzustellen, dass sich zusammengehörende Menü- und Werkzeugleisteneinträge identisch verhalten.
- Erstellen Sie dazu eine eigene Klasse, welche `AbstractAction` erweitert.



Aufgabe 4

```
public class ConsoleAction extends AbstractAction {

  public ConsoleAction(String name, String shortDescription,
    char mnemonic, KeyStroke accelerator,
    Icon smallIcon, Icon largeIcon) {

    super(name, smallIcon);
    putValue(SHORT_DESCRIPTION, shortDescription);
    putValue(MNEMONIC_KEY, (int) mnemonic);
    putValue(ACCELERATOR_KEY, accelerator);
    putValue(LARGE_ICON_KEY, largeIcon);
  }

  @Override
  public void actionPerformed(ActionEvent e) {
    System.out.println((String) getValue(SHORT_DESCRIPTION));
  }
}
```

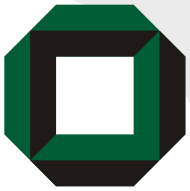
Autoboxing



Aufgabe 4

Verwendung einer Action:

```
Action openAction = new ConsoleAction("Open", "Open file", 'O',
  KeyStroke.getKeyStroke("ctrl O"), new ImageIcon("icons/open_S.png"),
  new ImageIcon("icons/open_L.png"));

JMenuBar menuBar = new JMenuBar();
JMenu file = new JMenu("File");
file.add(openAction);
menuBar.add(file);

JToolBar toolBar = new JToolBar();
toolbar.add(openAction);
```



Aufgabe 4

- Auslagern der sprachspezifischen Texte in eine separate Properties-Datei pro Sprache.
- Zugriff über `java.util.ResourceBundle`

```
ResourceBundle resources =
  ResourceBundle.getBundle("resources", new
  Locale("de"));
```

Nur Sprache. Es gibt noch Konstruktoren mit Land und Variante.

- Auch möglich: Klasse schreiben, die von `ListResourceBundle` erbt und sprachspezifische Texte dort ablegen.

Aufgabe 4

Inhalt von resources_de.properties:

```
action.accl.file.open=ctrl O
action.accl.file.save=ctrl S

action.icon.file.open=/icons/open.png
action.icon.file.save=/icons/save.png

action.mnem.file.open=O
action.mnem.file.save=S

action.name.file.open=Open
action.name.file.save=Save
```

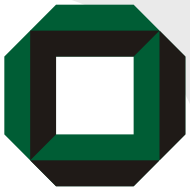
Tastenkürzel definieren. Das für die Klasse **`KeyStroke`** zulässige Format findet sich in der Java-Dokumentation.

Pfad zu den Icon-Dateien.

Mnemonische Buchstaben. Auch möglich: Unicode z.B. \u00FC oder Index des Buchstaben.

Beschriftung der Menüeinträge.

Die Bezeichner auf der linken Seite können Sie frei wählen.



Aufgabe 4

Im Java-Quelltext können Sie auf diese Eigenschaften mit den festgelegten Bezeichnern zugreifen:

```
KeyStroke accelerator = KeyStroke.getKeyStroke(resources
   .getString("action.accl.file.open"));
```

Irgendeine Klasse im JAR

```
URL url = SomeClass.class.getResource(
   resources.getString("action.icon.file.open"));
Icon icon = new ImageIcon(Toolkit.getDefaultToolkit().getImage(url));
```