Is it SOLID?

Ort, Luzern. Juni 2021
Samuel Hopf

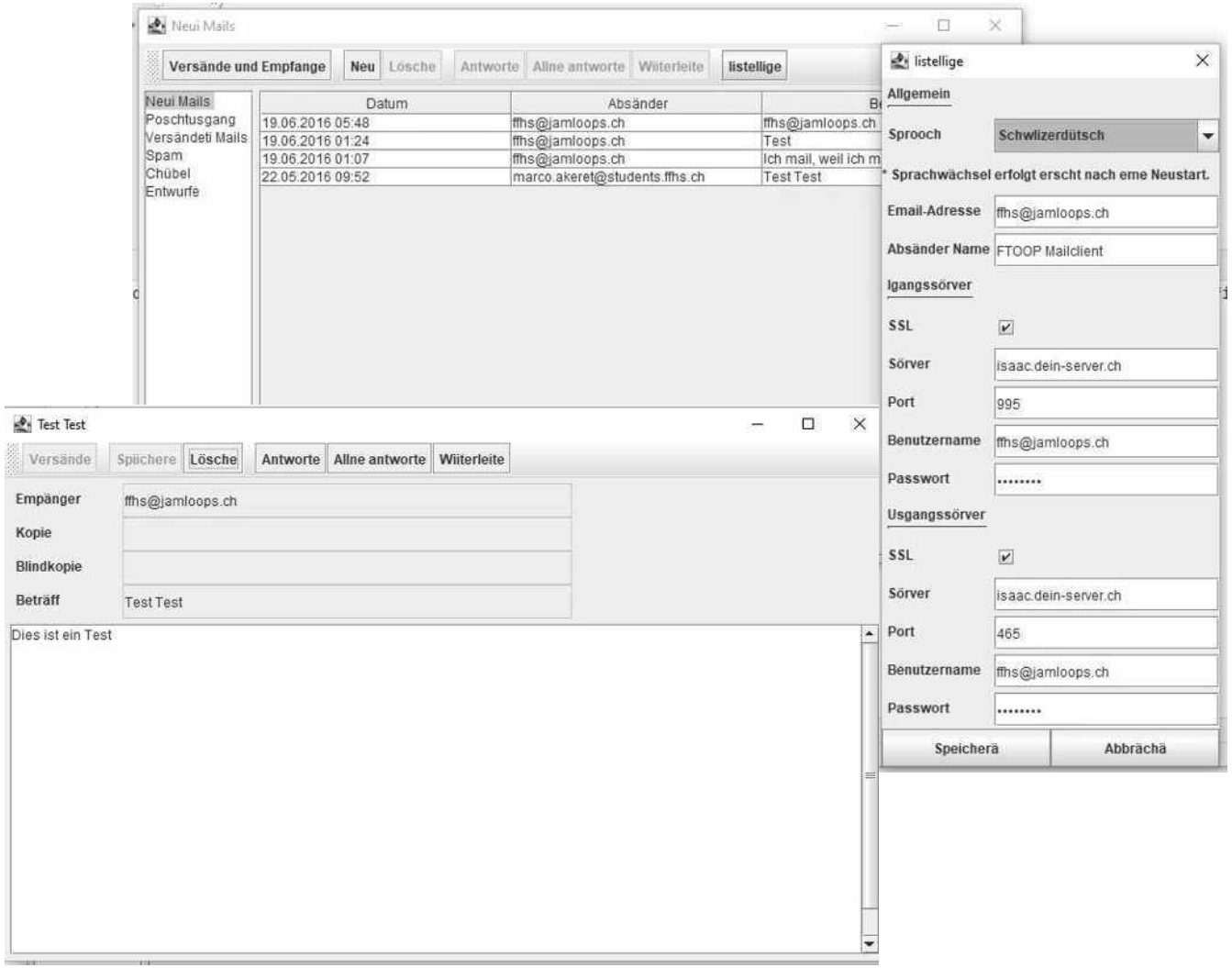
ALCOR Academy Training

Agenda

- The «school» project
- Our approach
- Is it SOLID?
  - Single Responsability
  - Open/Closed
  - Liskov Substitution
  - Interface Segregation
  - Dependency Inversion
- Anything else?

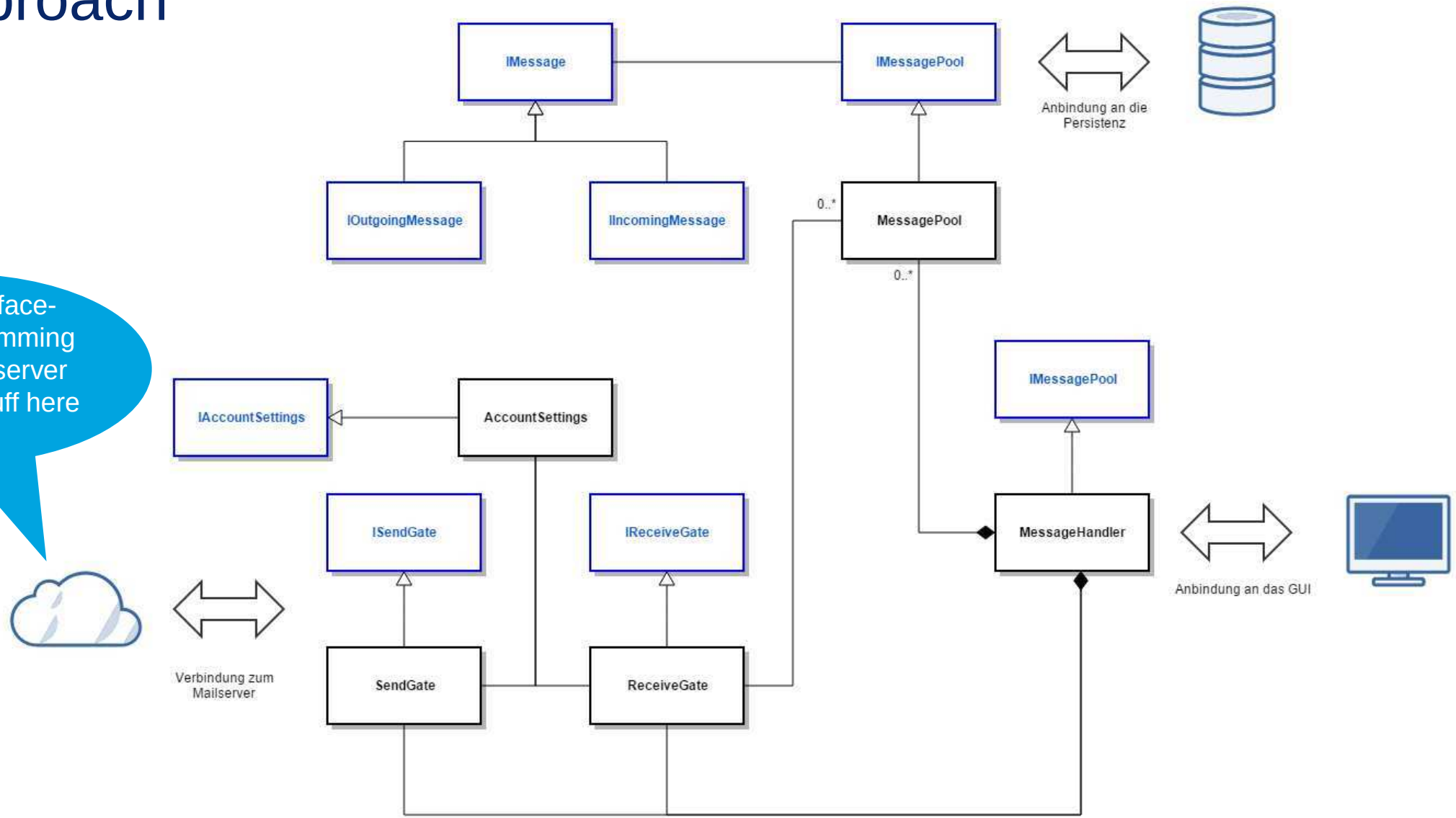
The «school» project 🎓

- from 2016
- A simple email client
  - Basic mailing functionality
  - Configurable
  - No user folders (but it's possible by design)
  - «Very important»: Swiss German 😍
- Develop a test concept 😇
  - Including Unit Testing
  - It's not Test-Driven!
- Checkstyle had to be happy 😋
  - Full Java Doc required



Our approach

All that interface-based programming is in the mailserver connection stuff here too!



Is it SOLID? - Single Responsability

☝ A subsystem, module, class, or even a function, should not have more than one reason to change.

```
public class PortTextField extends TextField {

    private final static String TEXT_FIELD = "txt_port";


    public PortTextField(String fieldId) {
        super(I18n.getInstance().getLangString(TEXT_FIELD));
        setFieldId(fieldId);
        init();
    }

    private void init() {
        addValidator(new NotEmptyValidator(this));
        addValidator(new NumberValidator(this));
        addValidator(new PortValidator(this));
    }
}
```

```
public interface IMessagePool {

    public ArrayList<IMessage> getMessages(MessageStatus status);

    public void addMessageToPool(IMessage message);

    public void removeMessageFromPool(IMessage message);

    public void updateMessage(IMessage message);

    public void loadFromPersistency(); 😱💀

    public ArrayList<IOutgoingMessage> getSendQueue();
}
```

Is it SOLID? – Open/Closed

☝ Open for extension and closed for modification

```
public interface IReceivable {

  List<IIncomingMessage> receiveMails(final ISessionSettings accountSettings) throws MessagingException, IOException, SettingsException;

  boolean deleteMessage(final IMessage message, final ISessionSettings accountSettings) throws MessagingException, IOException, SettingsException;
}
```

```
public class POP3Client implements IReceivable
```

```
public class ReceiveClient {

  private static IReceivable receiveClient;

  private ReceiveClient() {}

  public static synchronized IReceivable getInstance() {
    if(receiveClient == null) {
      receiveClient = new POP3Client();
    }
    return receiveClient;
  }
}
```

This could also be an IMAP Client

Is it SOLID? – Liskov Substitution

☝ Derived classes/interfaces should keep promises made by base classes.

Looks good – do you want to try to find one that does not?

Is it SOLID? – Interface Segregation

☝ Clients should not be forced to depend upon interfaces that they do not use.

```
public class IncomingMessage extends BaseMessage implements IIncomingMessage {
```

```
public abstract class BaseMessage implements IMessage, Serializable {
```

```
public interface IIncomingMessage extends IMessage {
```

Is it SOLID? – Dependeny Inversion

☝ High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.

```
public interface IMessage {
    List<String> getReceivers();
    public MessageStatus getMessageStatus();
    public void setMessageStatus(MessageStatus status);
    List<String> getCC();
    String getSender();
    String getSubject();
    String getContent();
    Date getTimeStamp();
    String getUUID();
    public boolean isMutable();
    public boolean canBeReplied();
    void setCC(final List<String> cc);
    void setContent(final String content);
    void setReceivers(final List<String> receivers);
    void setSender(final String sender);
    void setSubject(final String subject);
    void setTimeStamp(final Date timeStamp);
}
```

```
public interface IIncomingMessage extends IMessage {
    public void setFinishedTransfer();
    void setUUID(final String uuid);
}
```

```
public interface IOutgoingMessage extends IMessage {
    List<String> getBCC();
    void setBCC(final List<String> bcc);
    public void setFinishedSending();
}
```

Anything else?

- Most of that code is really nice to read
- Small classes, short methods
- It may be a little overcommented 😂

BUT:

- It’s a school project: you see where the time was running out
  - …as in production life 😉
- Primitive Obsession
  - SWITCH overuse
- There are a few WTF moments
  - poll() method that sends messages

One more – The Winner of the Identity-Crysis Award 🏆

```java
public class SystemMailFolder implements IMailFolder {

    private final static String NAME_INBOX = "txt_inbox";
    private final static String NAME_OUTBOX = "txt_outbox";
    private final static String NAME_SENT = "txt_sent";
    private final static String NAME_JUNK = "txt_junk";
    private final static String NAME_TRASH = "txt_trash";
    private final static String NAME_DRAFT = "txt_draft";

    private final MessageStatus folderType;

    public SystemMailFolder(MessageStatus status) {
        this.folderType = status;
    }

    @Override
    public IMailFolder getParent() {
        return null;
    }

    @Override
    public String getName() {
        switch (folderType) {
        case INBOX:
            return I18n.getInstance().getLangString(NAME_INBOX);
        case OUTBOX:
            return I18n.getInstance().getLangString(NAME_OUTBOX);
        case SENT:
            return I18n.getInstance().getLangString(NAME_SENT);
        case JUNK:
            return I18n.getInstance().getLangString(NAME_JUNK);
        case TRASH:
            return I18n.getInstance().getLangString(NAME_TRASH);
        case DRAFT:
            return I18n.getInstance().getLangString(NAME_DRAFT);
        default:
            return "";
        }
    }

    @Override
    public ArrayList<IMessage> getMessages() {
        return MailClient.getMessageHandler().getMessages(folderType);
    }

    @Override
    public MessageStatus getType() {
        return folderType;
    }

    @Override
    public boolean isIncomingFolder() {
        if (folderType == MessageStatus.INBOX) {
            return true;
        }
        if (folderType == MessageStatus.JUNK) {
            return true;
        }
        if (folderType == MessageStatus.TRASH) {
            return true;
        }
        if (folderType == MessageStatus.FOLDER) {
            return true;
        }

        return false;
    }

    @Override
    public boolean isOutgoingFolder() {
        if (folderType == MessageStatus.TRASH) {
            return true;
        }
        if (folderType == MessageStatus.FOLDER) {
            return true;
        }

        return !isIncomingFolder();
    }
}
```

Credits

- Source Code: FFHS FTOOP Module Project, Samuel Hopf / Marco Akeret
- Pictures from: https://search.creativecommons.org/ (Public Domain or CC commercial)
  - Apart from the Liskov Picture, which is from Wikipedia
- Solid Principles taken from: Alcor Training Lesson 4 – SOLID++

Thank you for your patience – any questions?