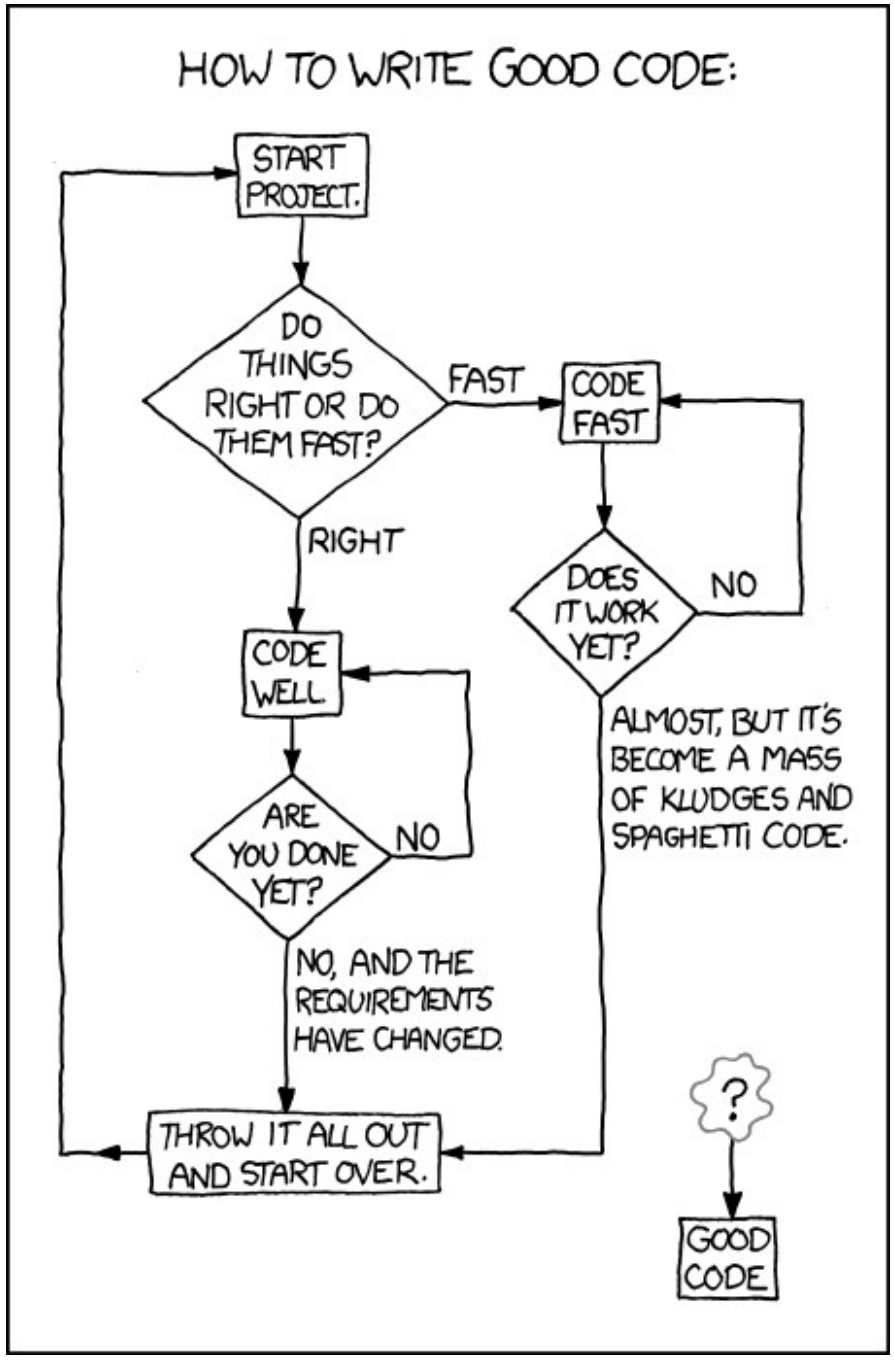
To-Do Server

Miniproject



Project goals

- **Program a Java server that implements a defined API**
  - The server is reachable on a particular IP-Address and Port
  - The API is simple, text-based messaging (RESTful)
  - API requests are messages – a topic we will be discussing

- **We haven't yet covered all topics, but you can already start**
  - Plan, implement and test your data classes
  - Create the GUI for a simple client that can use your server
  - As you learn more, add to your implementation
    - *Especially the topics "Networking Concepts" and "Messaging Technologies"*

- **Option to implement a client**
  - Everyone's client should work with everyone's server!
  - Great chance to help each other test!

Overview: To-Do program

- **Basic concept**
  - Central server that provides API functionality
  - RESTful: Each request is independent, stateless, and self-contained
    - *Synchronous messaging: one client query generates one server answer*
    - *The server never sends a message, except as a reply to a client message*
  - We will use simple text (a real implementation might use XML or JSON)

- **General functionality**
  - Account management
    - *Create account, login, logout, change password*
  - List, Create, Get and Delete “ToDo” entries
    - *Note: there is no way to change an existing ToDo*
  - There are two kinds of ToDo entries: with and without a due-date

Information about the data

- **Account attributes**
  - User-name: An email address
  - Password: 3-20 characters (String)

- **ToDo attributes**
  - A unique ID number, generated by the system (Integer)
  - A title of 3-20 characters (String)
  - A priority (enumeration: Low, Medium, High)
  - A description of 0-255 characters (String, can be 0 characters)
  - Optional: A due date (LocalDate, today or in the future)

Data representation – what you should think about

- **How will you store account information?**
  - Should passwords be encrypted? How do you check passwords?
- **How will you represent the ToDo entries?**
  - You could use a superclass, and a subclass that adds a due-date
  - You could use a single class; if no due-date then store a null
  - Are IDs unique across all ToDo entries, or only for a particular user?
- **How do you know which ToDo belongs to which user?**
  - You could have a separate list for each user
  - You could have a single list, and add a ToDo-attribute to identify the user
- **Design your data classes first**
  - Implement them
  - Test them, either with jUnit or with a simple program

Messaging protocol

- **The messaging protocol is plain-text, similar to the HTTP protocol**
  - Each command is a single line of text / each reply is a single line of text
  - A client should open a Socket connection and use it for multiple commands

- **Individual messages follow the format**
  - `MessageType [Token] Data`
    - ***`MessageType`*** *identifies the kind of message*
    - ***`Token`*** *must be sent with almost all messages from client to server*
      - Everything except Ping, Login and CreateLogin
    - ***`Data`*** *varies by MessageType*
  - Message parts are separated by vertical bars '|'

Messages (client → server)

MessageType	Data	Notes
CreateLogin	Username, Password	Fails if name already taken, or invalid After creating an account, you still have to login
Login	Username, Password	Fails if name/password do not match
ChangePassword	New password	Fails if new password is too short or if token is invalid
Logout	–	Never fails; token becomes invalid
CreateToDo	Title, Priority, Description [, DueDate]	Fails if data is invalid: title too short, date in the past, etc. Server replies with the ID
GetToDo	ID	ID, Title, Priority, Description [, DueDate]
DeleteToDo	ID	Fails if the ID does not exist (for this user)
ListToDos	List of IDs	Returns a list of all ToDo ID numbers for this user
Ping	[Token]	Without a token: always succeeds With token: succeeds only if token is valid

All commands that work with ToDos...
- ...require a valid token (the user must be logged in)
- ...only work for ToDos that belong to the current user

Messages (server → client)

MessageType	Data	Notes
Result	true true Data true Token false	*true* if the command succeeded, no data to return *true* if the command succeeded, and returned data A successful login returns the token as its data *false* if a command fails for any reason

Note the last entry:
- Commands can fail for various reasons.
- If a command fails, the server provides no information.
- The server just returns a result of ***false****.*

Sample exchange of messages

```
Ping
Result|true
CreateLogin|brad@fhnw.ch|mypassword
Result|true
Login|brad@fhnw.ch|mypassword
Result|true|4FA4563A5C2FFD1E703B49190DC348BD
CreateToDo|4FA4563A5C2FFD1E703B49190DC348BD|Shop|high|Buy food
Result|true|0
CreateToDo|4FA4563A5C2FFD1E703B49190DC348BD|Exercise|low|Go jogging
Result|true|1
ListToDos|4FA4563A5C2FFD1E703B49190DC348BD
Result|true|0|1
GetToDo|4FA4563A5C2FFD1E703B49190DC348BD|0
Result|true|0|Shop|High|Buy food
GetToDo|4FA4563A5C2FFD1E703B49190DC348BD|999
Result|false
SomeInvalidCommand
Result|false
Ping|4FA4563A5C2FFD1E703B49190DC348BD
Result|true
Logout
Result|true
```

Client messages are green
Server replies are black

Technical details: tokens

- **A token is a secret identifier that a client receives after logging in**
  - The client sends it with every command, to prove who it is
  - In a commercial system, the messages would be encrypted for security
- **The server sees the token and knows**
  - Who the client is
  - That the client successfully logged in
- **A token is becomes invalid when the user logs out**
  - Optional: a token becomes invalid after a certain amount of time
- **You can generate tokens any way that you want**
  - They don’t have to look like the ones in the example

Technical details: Dates

- **Dates use the ISO format: YYYY-MM-DD**
  - Example, 23 September 2020 is 2020-09-23
- Reading dates:
  - `LocalDate someDate = LocalDate.parse(dateString);`
- Writing dates:
  - `String dateString = someDate.format(DateTimeFormatter.ISO_DATE)`

Technical details: sample server for comparison

- **Sample solution: A ToDo-server is running on**
  - javaprojects.ch (IPv4 147.86.8.31), port: 50002
  - Implements the minimum requirements (see later slides)
    - *Plus due-dates*
    - *No validation*
  - Data deleted after a certain amount of time

- **Why this server?**
  - This shows what your solution ought to do
  - You can use it to test your client

- **If you crash the server, please let me know** ☺

Technical details: test client

- **Online, you will find a simple console-based client**
  - You can use this to test your server
  - It shows the exact text sent and received
- **You can find the client under the "Networking" topic, package "testClient"**
  - Start the client, enter the server and port, then enter commands
  - See example on the right

```
TestClient (3) [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Oct 1, 2020, 4:03:48 PM)
Enter the address of the server
javaprojects.ch
Enter the port number on the server (1024-65535)
50002
Connected
Enter commands to server or ctrl-D to quit
Ping
Sent: Ping
Received: Result|true
Login|brad|brad
Sent: Login|brad|brad
Received: Result|false
CreateLogin|brad|brad
Sent: CreateLogin|brad|brad
Received: Result|true
Login|brad|brad
Sent: Login|brad|brad
Received: Result|true|032F4A1D7C09707F73FB888B733D40BE
ListToDos|032F4A1D7C09707F73FB888B733D40BE
Sent: ListToDos|032F4A1D7C09707F73FB888B733D40BE
Received: Result|true
CreateToDo|032F4A1D7C09707F73FB888B733D40BE|Shop|High|Buy stuff
Sent: CreateToDo|032F4A1D7C09707F73FB888B733D40BE|Shop|High|Buy s
Received: Result|true|0
GetToDo|032F4A1D7C09707F73FB888B733D40BE|0
Sent: GetToDo|032F4A1D7C09707F73FB888B733D40BE|0
Received: Result|true|0|Shop|High|Buy stuff
Logout
Sent: Logout
Received: Result|true
```

Minimum requirements

- **Work as a team of 2, 3 or 4, *using Git to coordinate your work***

- **Your server must**
  - Be able to serve multiple clients in parallel (like a web server)
  - Implement all of the API commands as described
  - Have no GUI – it is just a simple console application

- **Start simple! For the minimum requirements:**
  - The server listens on port 50002 – no input, no choice
  - Passwords are not encrypted; they are stored and compared in plain text
  - Use the client's user-name as the token
  - Only implement ToDos *without* due-dates
    - *If a date is specified by the client, ignore it*
  - Data is not saved; if the server stops, all data is lost

Optional features for more points

- **Validate data *on the server* (½ point)**
  - Username is an email address, minimum string lengths, due-dates today or in the future, etc..
- **Support due-dates (½ point)**
- **Hash the passwords (½ point)**
  - Use a one-way hash function to store and compare passwords
- **Use real tokens for user logins (½ point)**
  - Example: a big random number
- **Save and restore data (1 point)**
  - Save data to a file (whenever it changes? Every X minutes?)
  - When the server starts: if there is saved data, then read it
  - *Note: Only save accounts and ToDo-entries, nothing else*

Optional feature: MVC client (1½ to 3 points)

- **Note: will be tested with servers from other groups – not just yours!**

- **Create an MVC client that supports all API features (1½ to 2 points)**
  - Ask for IP address and port of server
  - All API features except due-date work
    - *If the server sends a due-date, ignore it*
    - *If you support due-dates, that is worth (+½ point)*
  - Simple display of a single ToDo entry
  - Ability to move through the list and see other ToDo entries

- **Nice GUI (1 point)**
  - Display all ToDo entries in a list or table
  - Validate all input data (*on the client*)
    - *Never send bad data to the server*
    - *Enable and disable controls appropriately*

Grading (maximum grade = 6.0)

- **Minimum requirements** **+6**
- **Minus one for each person in project** **-2/-3/-4**
- **Optional features** **+???**
- **MVC client** **+???**
- ***Partial implementations of features may receive partial points***
- **Example: 3 Person project, minimum requirements, plus due-date, plus MVC-client that also supports due-date**
  - +6 - 3 + ½ + 1½ + ½ = 5.5

Conditions of the mini-project

- **Write your own code!**
- **Everyone on a team writes some code!**
- **Mutual help?**
  - With ideas and concepts → yes
  - Giving examples → yes
  - Write code for someone else → no

Handing in your project on Moodle

- **One team member hands in a link to your Git repository**
  - Your repository contains your source code for the project
- **Include a README at the top level that contains**
  - Names of the team members
  - A list of the features you have implemented
    - *Why? To ensure that I don't overlook something that deserves points*
- ***Test your project***
  - Can someone outside your team clone your repository into an Eclipse project, and run it?
- **If you have a private repository, *don't forget to invite the instructor***