NTNU

Norwegian University of Science and Technology

Project Introduction

Magnus Johansen
Nattachart Tamkittikhun
Office A-281
ttm4100@item.ntnu.no

Intro

- Implement a Chat Client & Server (command line)
  - They communicate using a strict protocol (description)
  - The client is "stupid"
  - Server has all the application logic
  - Messages between the client/server are JSON objects

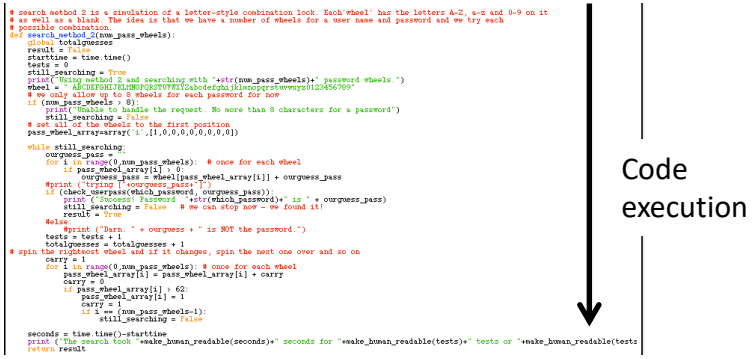
- Skeleton is in Python 2 and 3
  - You can use any other programming language, but:
    - Most likely, you are on your own
      - Translate skeleton, etc.
    - It has to support JSON (shouldn't be a problem)

Threading

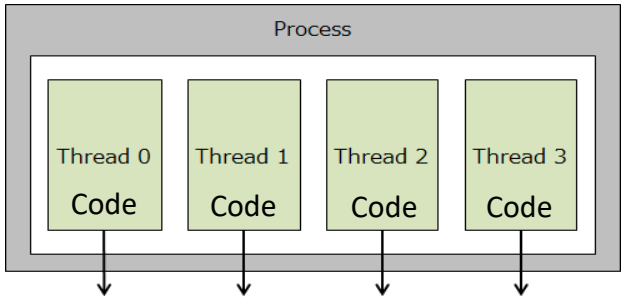
- Skeleton does it (mostly), but you have to understand it conceptually (the basics)

- "A thread is a way for a program (process) to divide itself into two or more simultaneously running tasks"
  - How a program can do several things in parallel
  - Execute code in parallel (same/different)

Single threaded

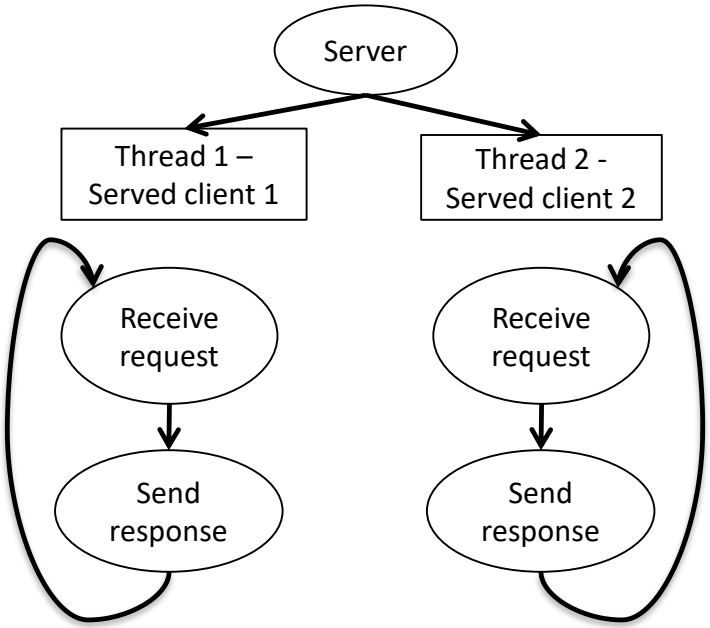


Multithreaded



Threading - Server

- Server parallel tasks:
  - Serving several clients at the same time (receiving requests and sending responses)
  - "served clients" must share some info:
    - Logged in clients
    - History of messages



For the server, individual "served clients" ARE sequential!!

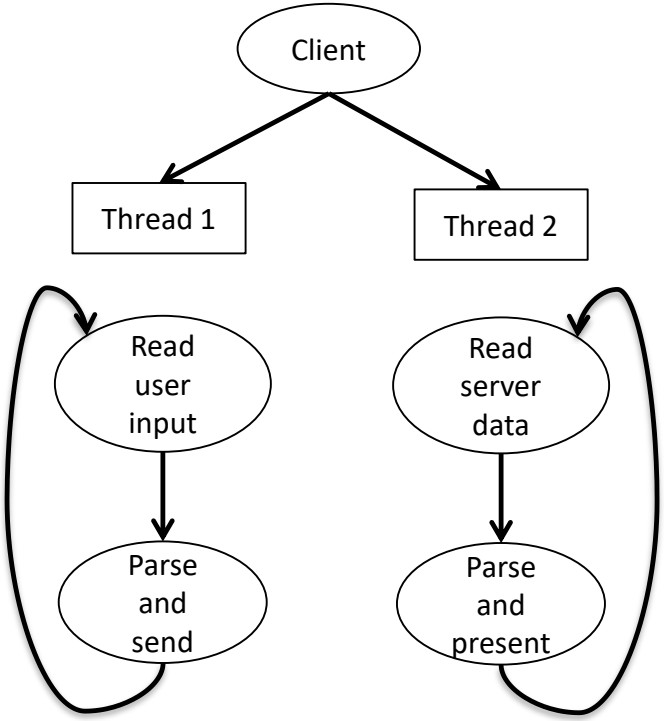
Server (for individual "served client"):
- receive request
- send response back
- repeat

SocketServer - ThreadedTCPServer

- SocketServer has a lot of functionality
  - No need for accept(), bind() -> handled automatically
- ThreadedTCPServer: our server
  - Listens and accepts connections
  - "Automagically" creates a multithreaded environment
    - When a remote socket connects
      - RequestHandler (ClientHandler) is instantiated, one per connection -> represents connected clients
      - handle() method is called -> should contain data wrt. A connected client
  - There is data used by all connected clients: it shouldn't be in the ClientHandler

Threading - Client

- Client parallel tasks:
  - Read input from the user and send request to the server
  - Receive response from the server and present it



Not sequential!!

Unlike e.g. SMTP prog. lab client:
- sends command to server
- reads response and presents it
- repeat

Chat client:
- Can receive responses from server without sending any request!!! (messages from other chat users)

MessageReceiver - Thread

- MessageReceiver inherits Thread
  - As any class, the constructor is called when a new MessageReceiver object is created
    - instantiates variables, etc
  - The run() method contains the code (what the thread does)

- Client class instantiates the MessageReceiver and calls start()
  - start() calls the run() method in a new thread
  - **DO NOT** call run() directly: it will make the caller block till the run() is complete!!

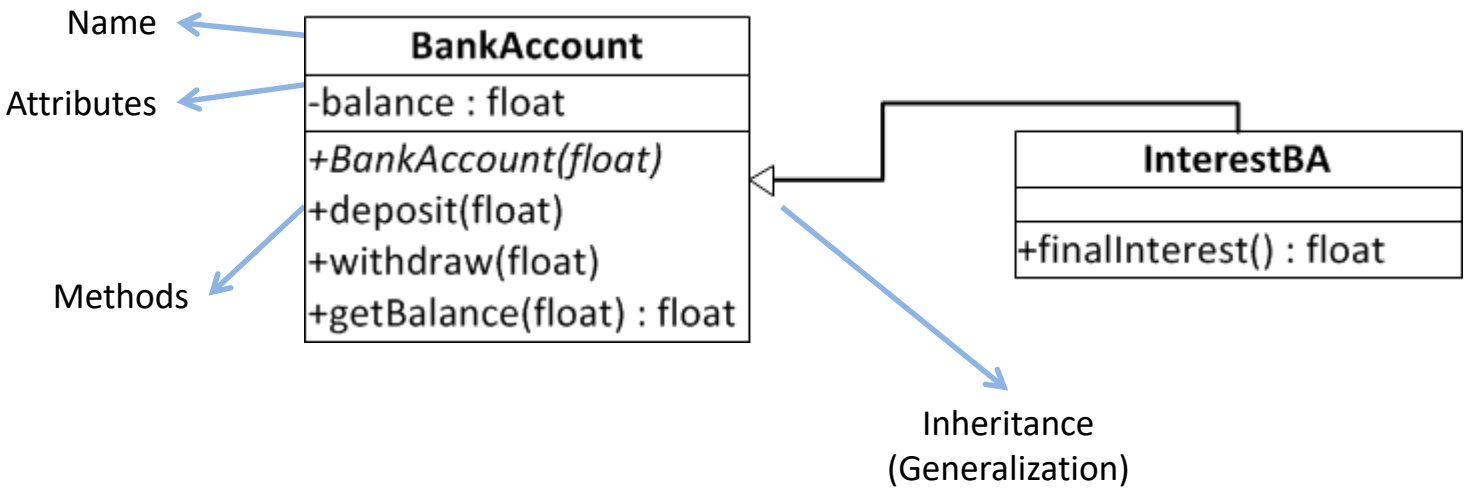
Diagrams (Design - KTN1)

- Objective: you understood the task and have an idea about how to solve it
- Not too concern about formality
  - Python is not pure object oriented, so it's difficult to keep correctness wrt. standards
- KTN1 will only be rejected if it is clear that you have no idea what you're doing

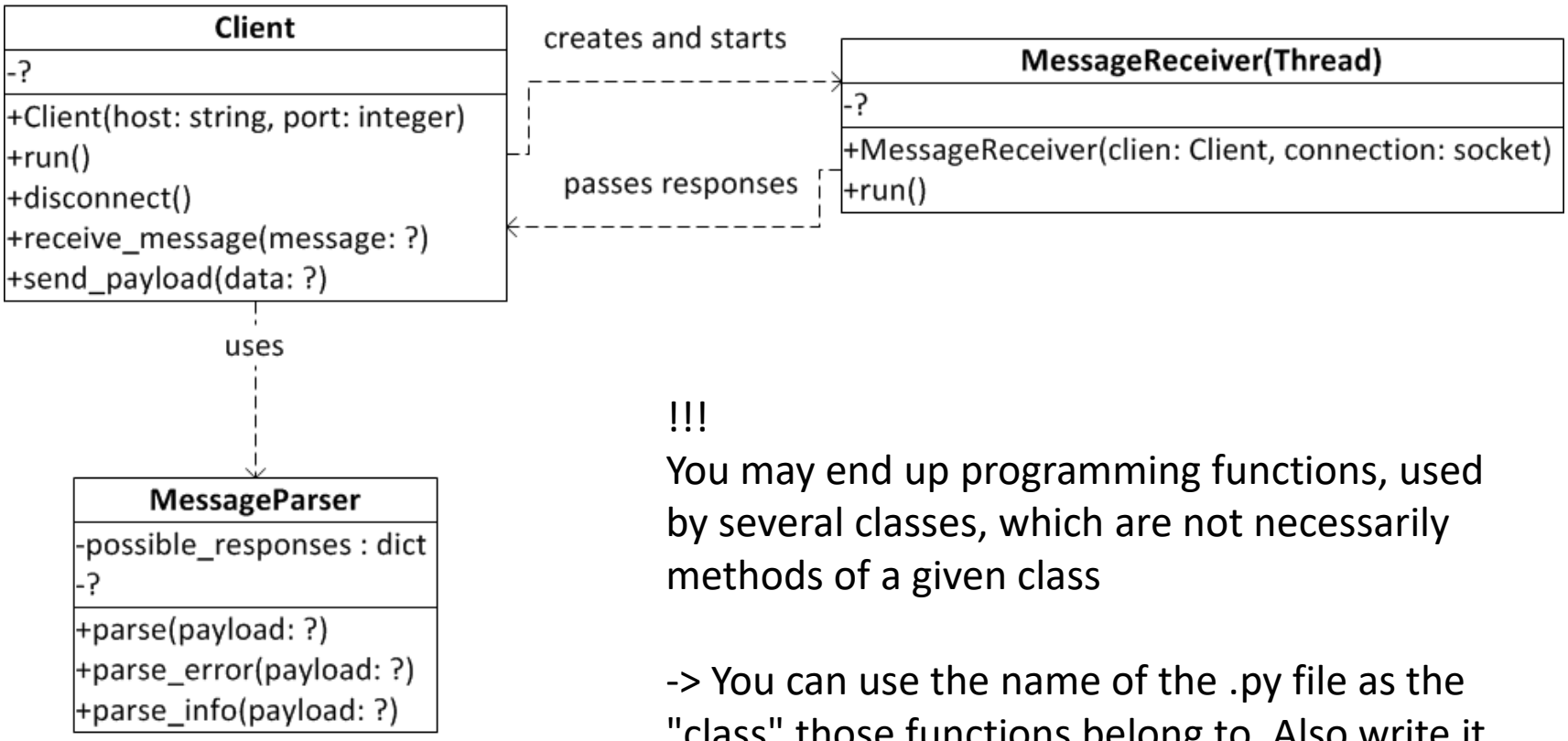
Class Diagrams

- Describe the structure of a system, i.e. its "classes" with
  - Attributes
  - Methods (operations)
  - Relationships

What we really care about (but the more detailed the better)



Example Client Skeleton



!!!
You may end up programming functions, used by several classes, which are not necessarily methods of a given class

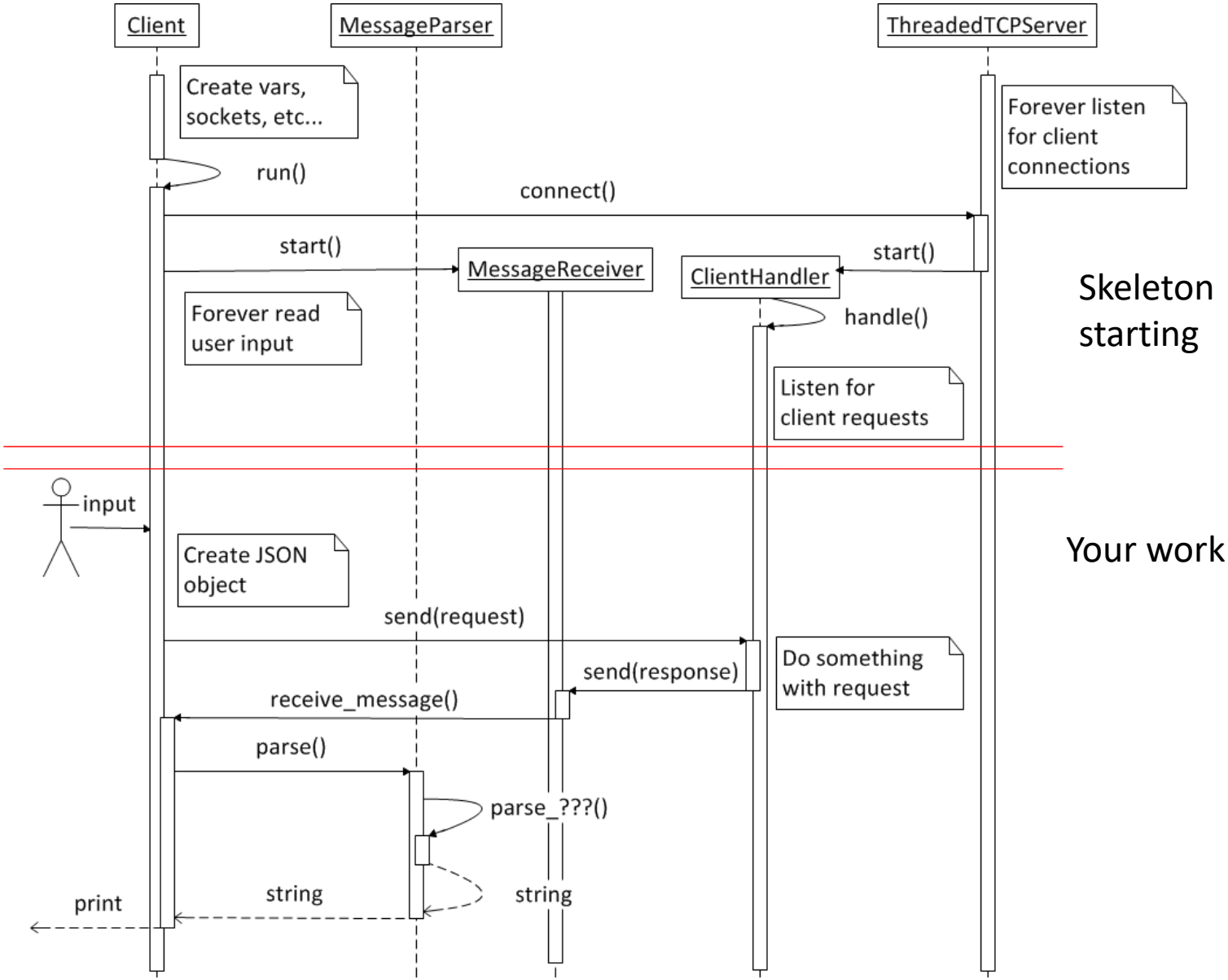
-> You can use the name of the .py file as the "class" those functions belong to. Also write it clearly in the textual description for KTN1

Sequence Diagrams

- Show how objects/processes/threads relate with one another:
  - Sequence of messages
  - In what order

- "Time" goes down
  - Vertical dashed lines: objects lifelines
    - Activation boxes: method/function is executing
  - Horizontal arrows: messages
    - Solid: calling a method/function
    - Dashed: replies/returns

Example Seq. Diagram



JSON

- **J**ava**S**cript **O**bject **N**otation
  - Lightweight data-interchange format
  - Language independent
    - JavaScript syntax, but it's text
  - Self-describing and easy to understand (simpler than XML)
  - Its widely used (e.g. Gmail)

JSON

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

XML

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

Working with JSON

- Import json
- Dictionaries in Python are converted to JSON objects (strings)
  - .dumps(dict)
- JSON objects are converted to dictionaries
  - .loads(JSON_object)

```
import json
'''
Converting a dictionary object to a JSON string:
'''
my_value = 3
my_list = [1, 2, 5]
my_dict = {'key': my_value, 'key2': my_list}
my_dict_as_string = json.dumps(my_dict)
print my_dict_as_string
#print my_dict_as_string['key']          #Error!
'''
Output:
{"key2": [1, 2, 5], "key1": 3}
'''
```

```
'''
Converting a JSON string to a dictionary object:
'''
my_value = 3
my_list = [1, 2, 5]
my_dict = {'key': my_value, 'key2': my_list}
my_dict_as_string = json.dumps(my_dict)
my_dict_from_string = json.loads(my_dict_as_string)
```

Deadlines

- KTN1 – Design: 10.03.2016
  - Class / sequence diagrams
  - Textual description
  - Submit early
    - More feedback
    - More time for KTN2
- KTN2 – Implementation: 24.03.2016
  - Code
  - Update of KTN1 if needed
  - **Demonstrate to a TA by 24.03.2016**

Layering (opinion)

- Individual, but may work in groups of 2
- Try to divide the work
  - Natural division
    - Client – Server
  - Layering
    - Networking layer: sending/receiving data
    - Parsing layer: coding/decoding into JSON
    - Application layer: app. logic (handling requests/responses)
- Layering: make class(es) that handle work at a given layer
  - App. layer calls methods in parsing layer
  - Parsing layer calls methods in netw. layer