

It's a kind of magic

How do application servers and frameworks work behind the scenes

AMT 2020

Olivier Liechti

**[https://github.com/SoftEng-HEIGVD/
Teaching-HEIGVD-AMT-WHATELSE](https://github.com/SoftEng-HEIGVD/Teaching-HEIGVD-AMT-WHATELSE)**

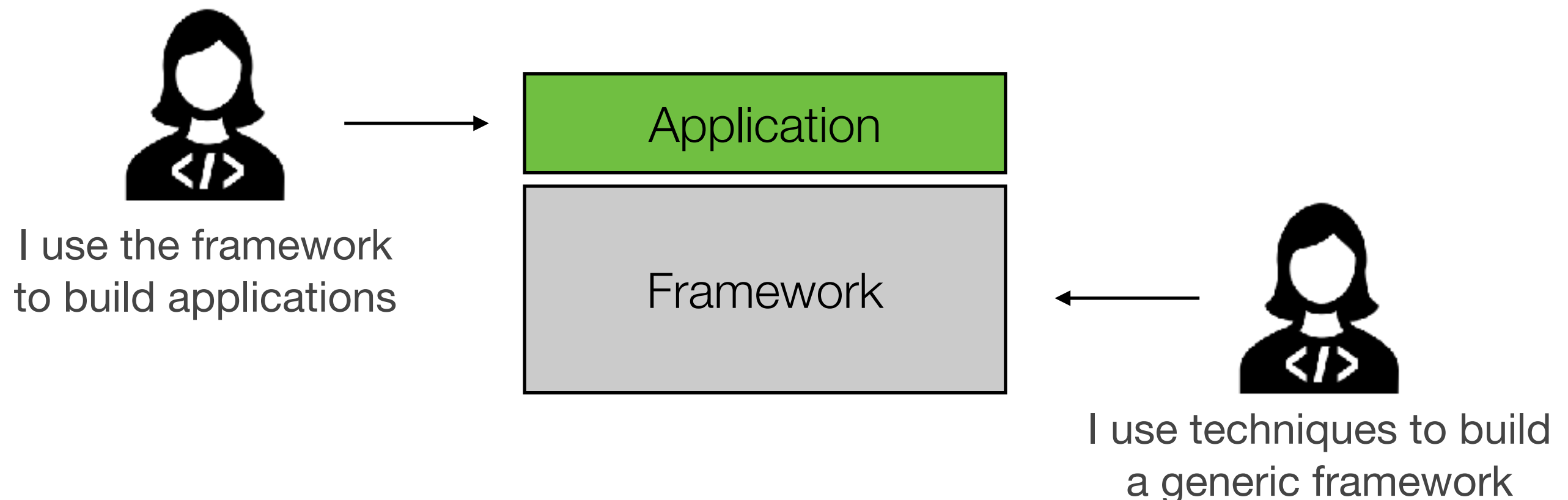
[https://github.com/SoftEng-HEIGVD/
Teaching-HEIGVD-AMT-Framework](https://github.com/SoftEng-HEIGVD/Teaching-HEIGVD-AMT-Framework)

[https://github.com/SoftEng-HEIGVD/
Teaching-HEIGVD-AMT-WHATELSE](https://github.com/SoftEng-HEIGVD/Teaching-HEIGVD-AMT-WHATELSE)

**[https://github.com/SoftEng-HEIGVD/
Teaching-HEIGVD-AMT-Framework](https://github.com/SoftEng-HEIGVD/Teaching-HEIGVD-AMT-Framework)**

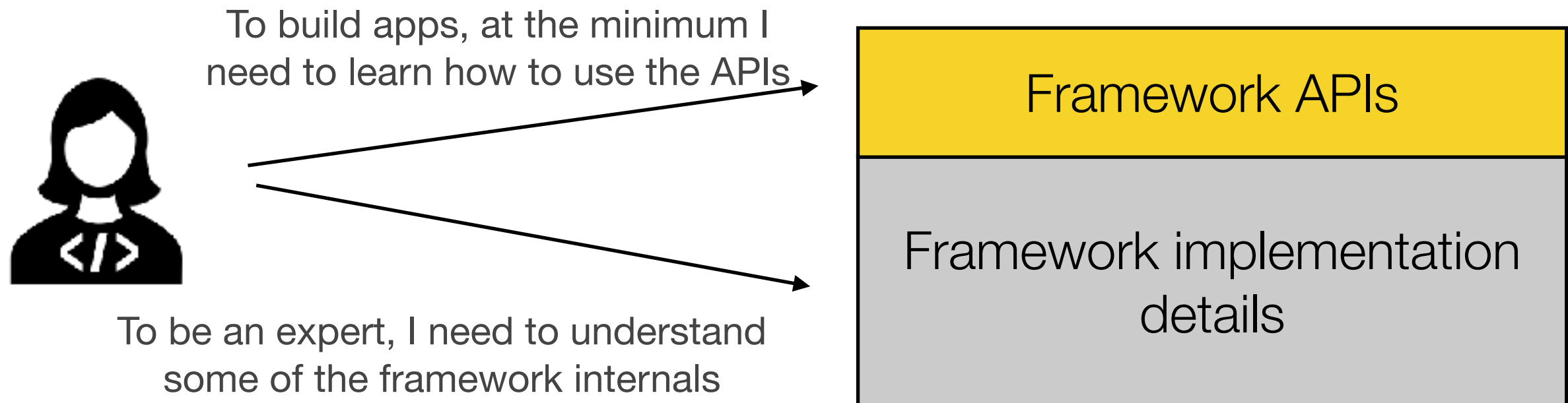
Applications vs frameworks

- During the AMT lectures, we present a number of **design patterns** and **implementation techniques**.
 - Some of these are **applied by application developers**.
 - Others are **used to build the frameworks** provided to application developers (the implementation of Java EE APIs is one such framework).



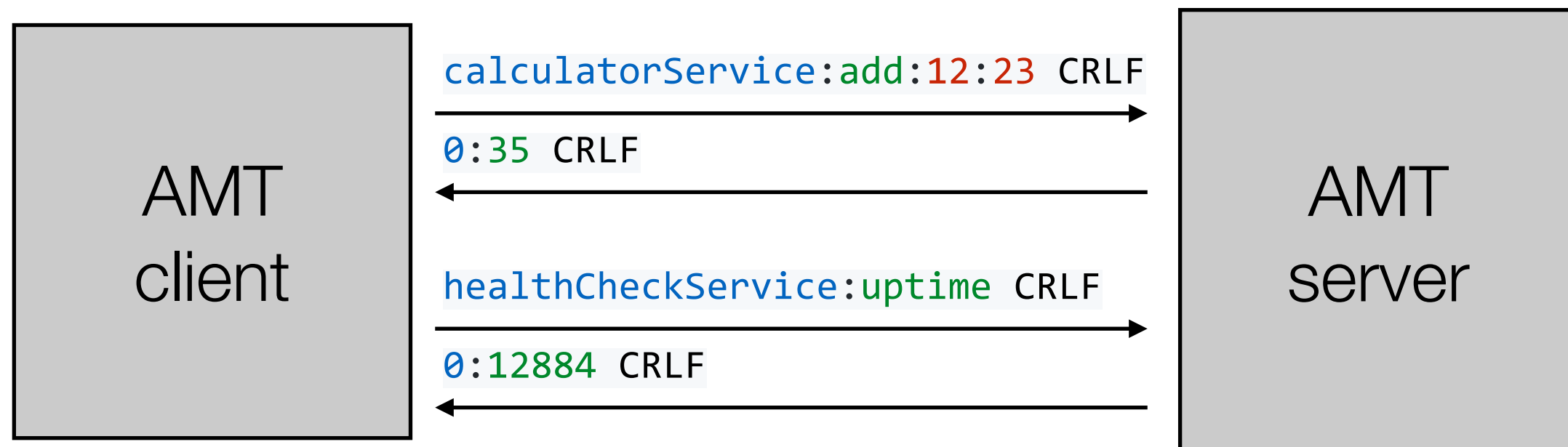
Framework internals

- Frameworks and application servers provide a rich, **high-level** development environment.
 - On the bright side, they **make application developers more productive**, because they do not have to take care of all details.
 - However, it is often hard to understand what is happening **behind the scenes**. This becomes a problem when app developers encounter bugs or need to optimize their code.



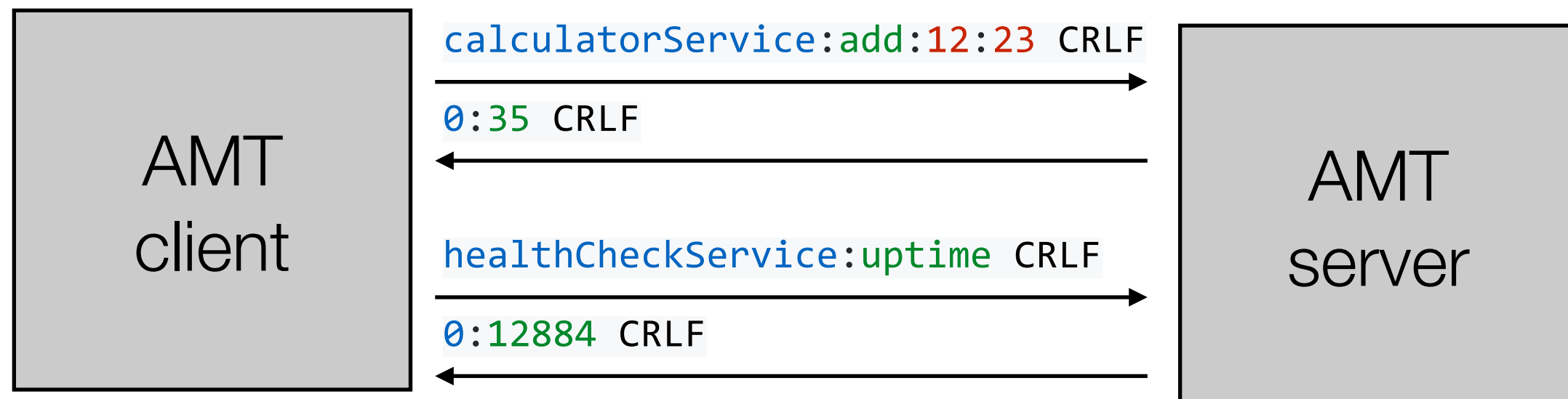
The AMT application level protocol

- **Instead of using HTTP**, we have decided to define a **simpler application-level protocol**.
- A client establish a **TCP** connection with a server. It can then a series of commands. The server sends a response for every command.
- A **command** contains the name of a **service**, the name of an **operation** and a list of 0, 1 or more **parameters**. They are separated by colons (:).
- A **response** contains a **status code** (0 = success) and a **return value**.



The AMT application level protocol

- Given this protocol specification, we can implement a server using **any programming language**.
- RES students would do it with the socket API in Java. They would **handle everything**: the **low-level aspects** (networking, IOs, concurrency) and the **high-level aspects** (the logic of a calculator service).
- Can we do better? Can we provide a **framework** to “**business**” **developers**, so that they can **easily extend** the server with cool services and operations? In other words, can we realize **separation of concerns**?

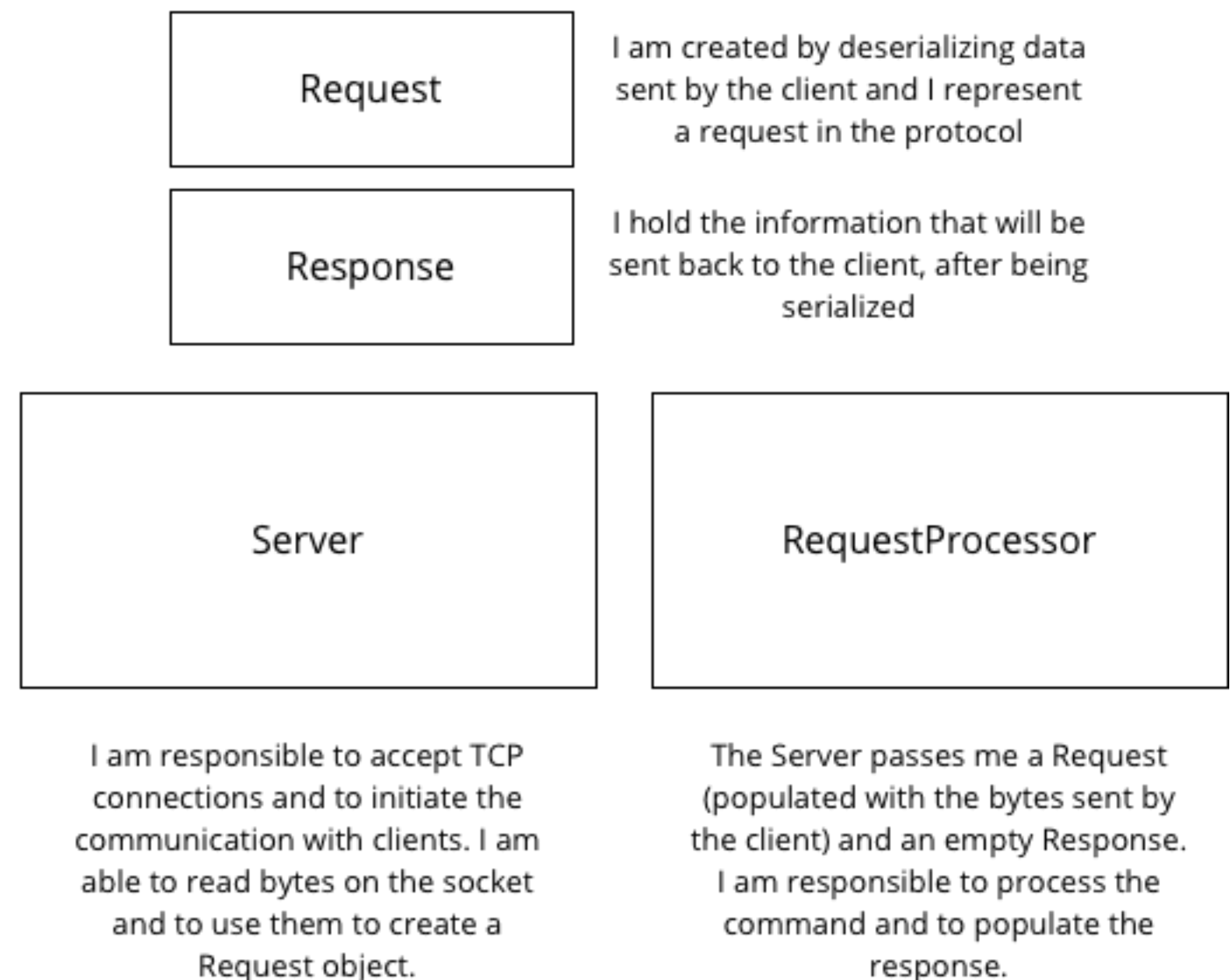


step-01-only-a-framework

Let's **get started** and implement an AMT server. We'll deal with separation of concerns later...

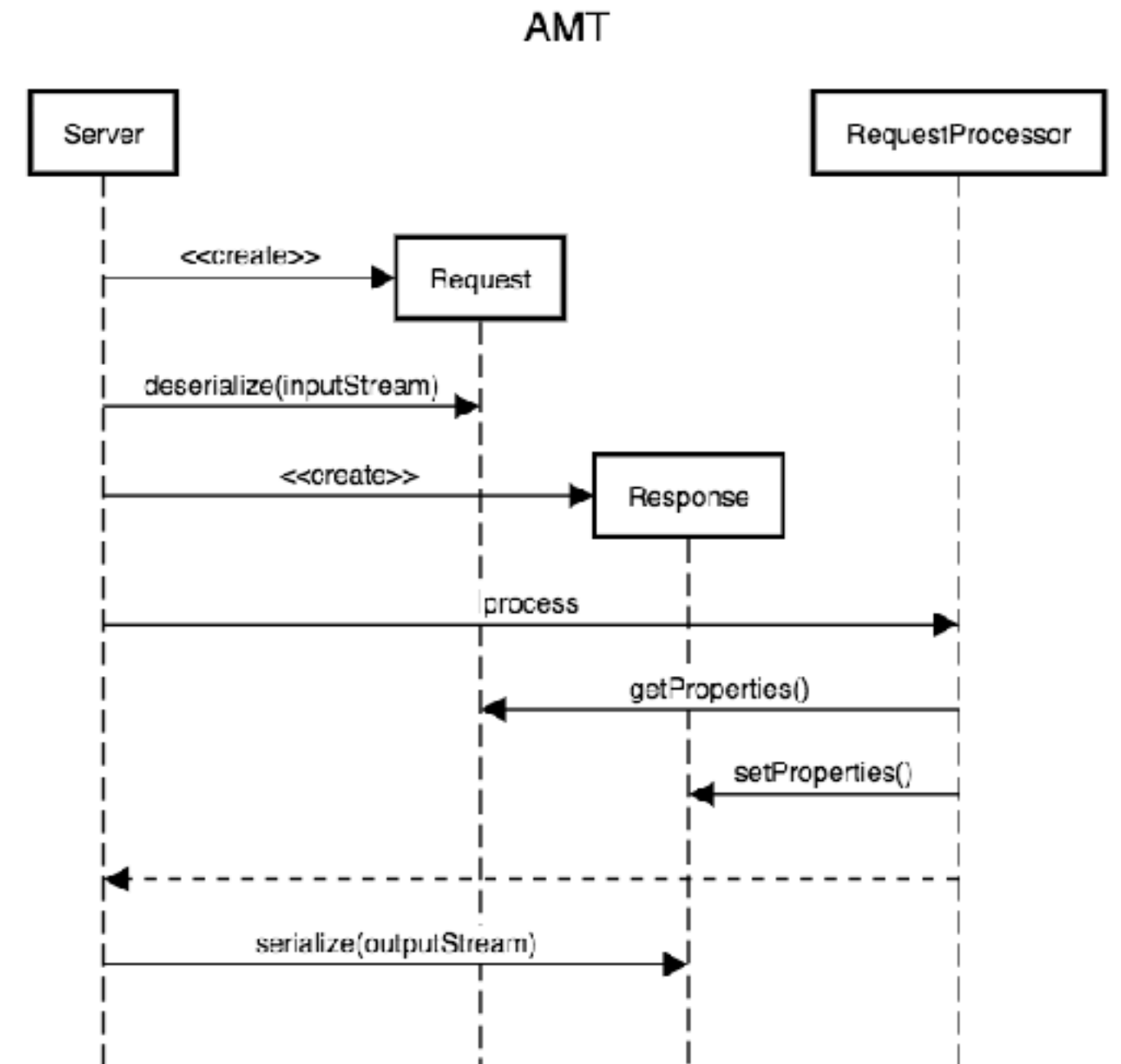
Step 1: tightly coupled server (1)

- In our initial solution, we did not seek to separate the “framework” from the “application” code.
- Our goal was to get a TCP server working, with an OO design that captures abstractions of the AMT protocol.
- We have a classes that deal with threading, networking and IOs. They create Request and Response objects that remind us of the Servlet API.



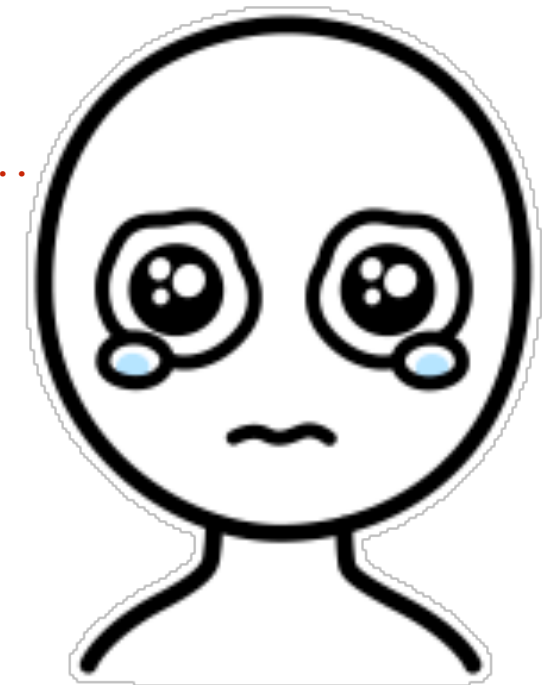
Step 1: tightly coupled server (2)

- The **RequestProcessor** does not need to know anything about the low-level protocol syntax (bytes read on a socket)
- It receives a **Request** object, with a **serviceName**, an **operationName** and a **parameterValues** properties
- It also receives an empty **Response** object, which it populates with a **statusCode** and a return **value**



Step 1: tightly coupled server (3)

```
package ch.heigvd.amt.framework.engine;
import ch.heigvd.amt.framework.exceptions.InvalidOperationException;
class RequestProcessor {
    void processRequest(Request request, Response response) throws InvalidOperationException {
        // TODO: Not a big fan of using a switch statement, which is going to grow large as we add commands...
        switch (request.getServiceName()) {
            case "healthCheckService": // this is a "framework" service, so kind of OK
                response.setStatusCode(0);
                switch (request.getOperationName()) {
                    case "ping":
                        response.setValue("I am alive");
                        break;
                    case "uptime":
                        response.setValue(Long.toString(Server.getServer().getUptime()));
                        break;
                }
                break;
            case "clockService": // is this a "framework" or "application" service...
                response.setStatusCode(0);
                response.setValue("2019-01-01");
                break;
            case "calculatorService": // TODO: this is definitely an "application" service and needs to be extracted!!
                response.setStatusCode(0);
                switch (request.getOperationName()) {
                    case "add":
                        Integer p1 = Integer.parseInt(request.getParameterValues().get(0));
                        Integer p2 = Integer.parseInt(request.getParameterValues().get(1));
                        response.setValue(Integer.toString(p1 + p2));
                        break;
                    case "mult":
                        p1 = Integer.parseInt(request.getParameterValues().get(0));
                        p2 = Integer.parseInt(request.getParameterValues().get(1));
                        response.setValue(Integer.toString(p1 * p2));
                        break;
                }
                break;
        }
    }
}
```



step-02-service-registry

Every framework **manages** a set of some sort of “**component**” (aka service, bean, resource, etc.). It gives them **names** and stores them in a **registry**.

Step 2: introduce a service registry

Step 2: As a **consumer**, I can **lookup** a service, based on its **name**.

```
registry.lookup("calculator").add(3, 5);
```



name	object
"calculator"	CalculatorService@6239967b
"quotes"	QuotesService@2881763b
"jokes"	JokesService@2214762b
...	...

Step 1: As a **provider**, I can **register** a service, giving it a **name**.

```
registry.register("calculator", new CalculatorService());
```

Step 2: define a service interface



```
package ch.heigvd.amt.framework.api;

import ch.heigvd.amt.framework.exceptions.InvalidOperationException;
import java.util.List;

public interface IService {

    String execute(String operationName, List<String> parameterValues) throws
        InvalidOperationException;

}
```

Step 2: implement a service registry

```
package ch.heigvd.amt.framework.engine;

public class ServiceRegistry {

    private Map<String, IService> registry = new HashMap<>();
    private static ServiceRegistry singleton = new ServiceRegistry();

    protected ServiceRegistry() {}
    public static ServiceRegistry getServiceRegistry() {
        return singleton;
    }

    public void register(String serviceName, IService service) {
        registry.put(serviceName, service);
    }

    public IService lookup(String serviceName) throws LookupException {
        IService candidate = registry.get(serviceName);
        if (candidate == null) {
            throw new LookupException("Service " + serviceName + " could not be looked up.");
        }
        return candidate;
    }
}
```

Step 2: register services at startup

```
public static void main(String[] args) {  
  
    ServiceRegistry registry = ServiceRegistry.getServiceRegistry();  
  
    registry.register(HealthCheckService.SERVICE_NAME, new HealthCheckService());  
    registry.register(ClockService.SERVICE_NAME, new ClockService());  
    registry.register(CalculatorService.SERVICE_NAME, new CalculatorService());  
  
    Server server = new Server();  
    server.start();  
}
```



This does not smell very good... we'll have to get back to that in the future (not today)

Step 2: cleaner RequestProcessor

```
package ch.heigvd.amt.framework.engine;

import ...

class RequestProcessor {

    void processRequest(Request request, Response response) throws
        InvalidOperationException {

        ServiceRegistry registry = ServiceRegistry.getServiceRegistry();

        try {
            IService service = registry.lookup(request.getServiceName());
            String returnValue = service.execute(request.getOperationName(),
                request.getParameterValues());
            response.setStatusCode(0);
            response.setValue(returnValue);
        } catch (LookupException e) {
            throw new InvalidOperationException(e.getMessage());
        }
    }
}
```



Step 3: split the codebase

What do we do if we want to add a “Translation” service to our server?

We need to work in the server codebase.

If many people start to build AMT services, we will problems.

Do we ask them to submit pull requests with their services? Or do they fork our server code and regularly integrate our changes?



step-03-split-codebase

We want to **separate** the **generic** server functionality from the **specific** “business” functionality. We want to provide a “SDK” to **service developers**.

Reflection might come in handy.

Step 3: break the code in modules



HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

How do we split these?

```
ch/heigvd/amt/framework/exceptions/InvalidOperationException.java
ch/heigvd/amt/framework/exceptions/InvalidRequestException.java
ch/heigvd/amt/framework/exceptions/LookupException.java
ch/heigvd/amt/framework/api/IService.java
ch/heigvd/amt/framework/engine/RequestProcessor.java
ch/heigvd/amt/framework/engine/Response.java
ch/heigvd/amt/framework/engine/Protocol.java
ch/heigvd/amt/framework/engine/ServiceRegistry.java
ch/heigvd/amt/framework/engine/Server.java
ch/heigvd/amt/framework/engine/Request.java
ch/heigvd/amt/framework/services/ClockService.java
ch/heigvd/amt/framework/services/CalculatorService.java
ch/heigvd/amt/framework/services/HealthCheckService.java
```

Step 3: break the code in modules

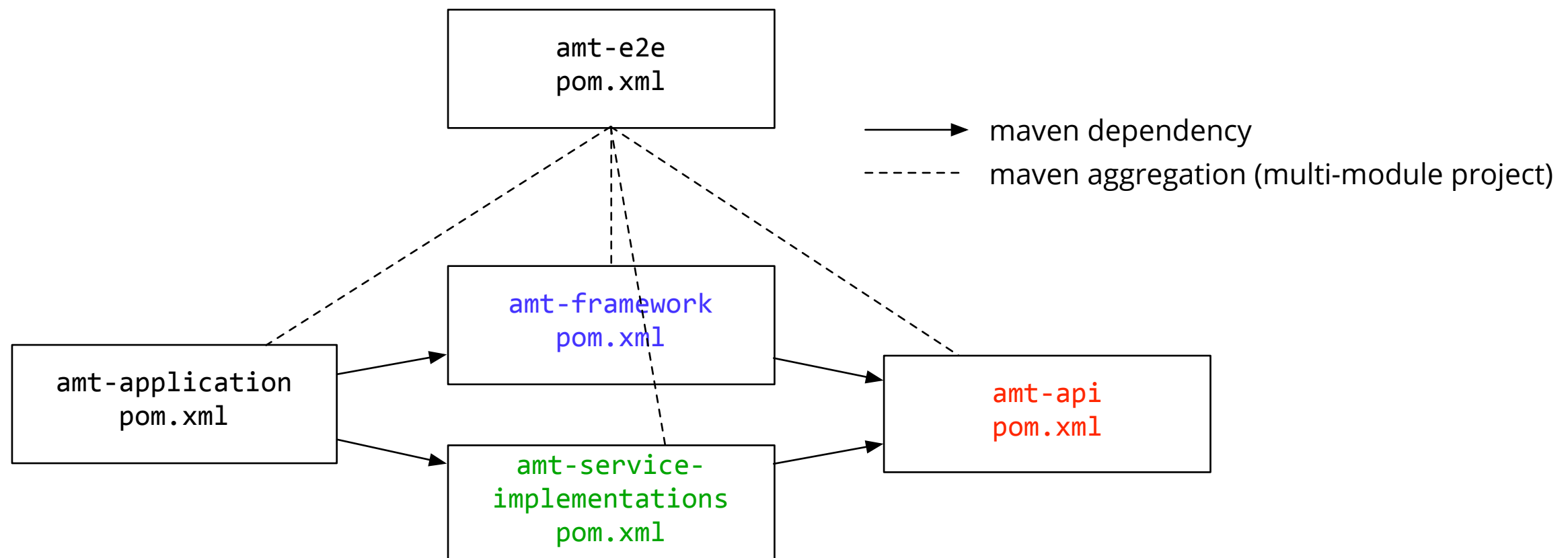
How do we split these?

```
ch/heigvd/amt/framework/exceptions/InvalidOperationException.java
ch/heigvd/amt/framework/exceptions/InvalidRequestException.java
ch/heigvd/amt/framework/exceptions/LookupException.java
ch/heigvd/amt/framework/api/IService.java
ch/heigvd/amt/framework/engine/RequestProcessor.java
ch/heigvd/amt/framework/engine/Response.java
ch/heigvd/amt/framework/engine/Protocol.java
ch/heigvd/amt/framework/engine/ServiceRegistry.java
ch/heigvd/amt/framework/engine/Server.java
ch/heigvd/amt/framework/engine/Request.java
ch/heigvd/amt/framework/services/ClockService.java
ch/heigvd/amt/framework/services/CalculatorService.java
ch/heigvd/amt/framework/services/HealthCheckService.java
```

Step 3: break the code in modules

```
ch/heigvd/amt/framework/exceptions/InvalidOperationException.java
ch/heigvd/amt/framework/exceptions/InvalidRequestException.java
ch/heigvd/amt/framework/exceptions/LookupException.java
ch/heigvd/amt/framework/api/IService.java
ch/heigvd/amt/framework/engine/RequestProcessor.java
ch/heigvd/amt/framework/engine/Response.java
ch/heigvd/amt/framework/engine/Protocol.java
ch/heigvd/amt/framework/engine/ServiceRegistry.java
ch/heigvd/amt/framework/engine/Server.java
ch/heigvd/amt/framework/engine/Request.java
ch/heigvd/amt/framework/services/ClockService.java
ch/heigvd/amt/framework/services/CalculatorService.java
ch/heigvd/amt/framework/services/HealthCheckService.java
```

How do we split these?



Step 3: break the code in modules

maven multi-modules

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>ch.heigvd.amt</groupId>
  <artifactId>amt-e2e</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>amt-api</module>
    <module>amt-framework</module>
    <module>amt-service-implementations</module>
    <module>amt-application</module>
  </modules>

</project>
```

Here, we state that in order to build our “end to end” project, we need to build these 4 modules. Maven will resolve dependencies between them, to figure out in which order they should be built.

<https://maven.apache.org/guides/mini/guide-multiple-modules.html>

<https://www.baeldung.com/maven-multi-module>

Step 3: break the code in modules



HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Running mvn clean install...

```
</project>Oliviers-MacBook-Pro:codebase olivierliechti$ mvn clean install
[INFO] Scanning for projects...
[INFO] -----
[INFO] Reactor Build Order:
[INFO]
[INFO] amt-api [jar]
[INFO] amt-service-implementations [jar]
[INFO] amt-framework [jar]
[INFO] amt-application [jar]
[INFO] amt-e2e [pom]
[INFO]
[INFO] -----< ch.heigvd.amt:amt-api >-----
[INFO] Building amt-api 1.0-SNAPSHOT [1/5]
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ amt-api ---
[INFO] Deleting /Users/olivierliechti/Documents/heig-vd/teaching/AMT/repos/Teaching-HEIGVD-AMT-
Framework/framework/codebase/amt-api/target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ amt-api ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
```


Step 3: break the code in modules



HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

... runs all build steps on every sub-module

```
[INFO] -----< ch.heigvd.amt:amt-e2e >-----
[INFO] Building amt-e2e 1.0-SNAPSHOT [5/5]
[INFO] -----[ pom ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ amt-e2e ---
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ amt-e2e ---
[INFO] Installing /Users/olivierliechti/Documents/heig-vd/teaching/AMT/repos/Teaching-HEIGVD-AMT-
Framework/framework/codebase/pom.xml to /Users/olivierliechti/.m2/repository/ch/heigvd/amt/amt-e2e/
1.0-SNAPSHOT/amt-e2e-1.0-SNAPSHOT.pom
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] amt-api ..... SUCCESS [ 3.363 s]
[INFO] amt-service-implementations ..... SUCCESS [ 1.919 s]
[INFO] amt-framework ..... SUCCESS [ 3.274 s]
[INFO] amt-application ..... SUCCESS [ 2.122 s]
[INFO] amt-e2e 1.0-SNAPSHOT ..... SUCCESS [ 0.005 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.770 s
[INFO] Finished at: 2019-10-13T12:34:39+02:00
```



<https://github.com/SoftEng-HEIGVD/Teaching-HEIGVD-AMT-Framework-SDK>

Specify the behaviour of your service...



HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

```
package com.wasabi.amt.services;

import ch.heigvd.amt.framework.api.ServiceTest;
import ch.heigvd.amt.framework.exceptions.InvalidOperationException;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.Arrays;

import static org.junit.jupiter.api.Assertions.*;

class JokesServiceTest extends ServiceTest {

    @BeforeEach
    void setupService() {
        service = new JokesService();
    }

    @Test
    void itShouldProvideATellJokeOperation() throws InvalidOperationException {
        String value = service.execute(JokesService.OPERATION_TELL_JOKE, null);
        assertNotNull(value);
    }
}
```

... and implement it

```
package com.wasabi.amt.services;

import ch.heigvd.amt.framework.api.IService;
import ch.heigvd.amt.framework.exceptions.InvalidOperationException;

import java.util.List;

public class JokesService implements IService {
    public static final String SERVICE_NAME = "jokesService";
    public static final String OPERATION_TELL_JOKE = "tellJoke";

    @Override
    public String execute(String operationName, List<String> parameterValues) throws InvalidOperationException {
        switch (operationName) {
            case OPERATION_TELL_JOKE:
                return "It's the story of paf the dog.";
            default:
                throw new InvalidOperationException("Operation " + operationName + " is not valid.");
        }
    }

    @Override
    public String getHelpMessage() {
        return "service: " + this.getClass().getCanonicalName() + "\r\n"
            + " operation: " + OPERATION_TELL_JOKE + " (no arguments)";
    }
}
```

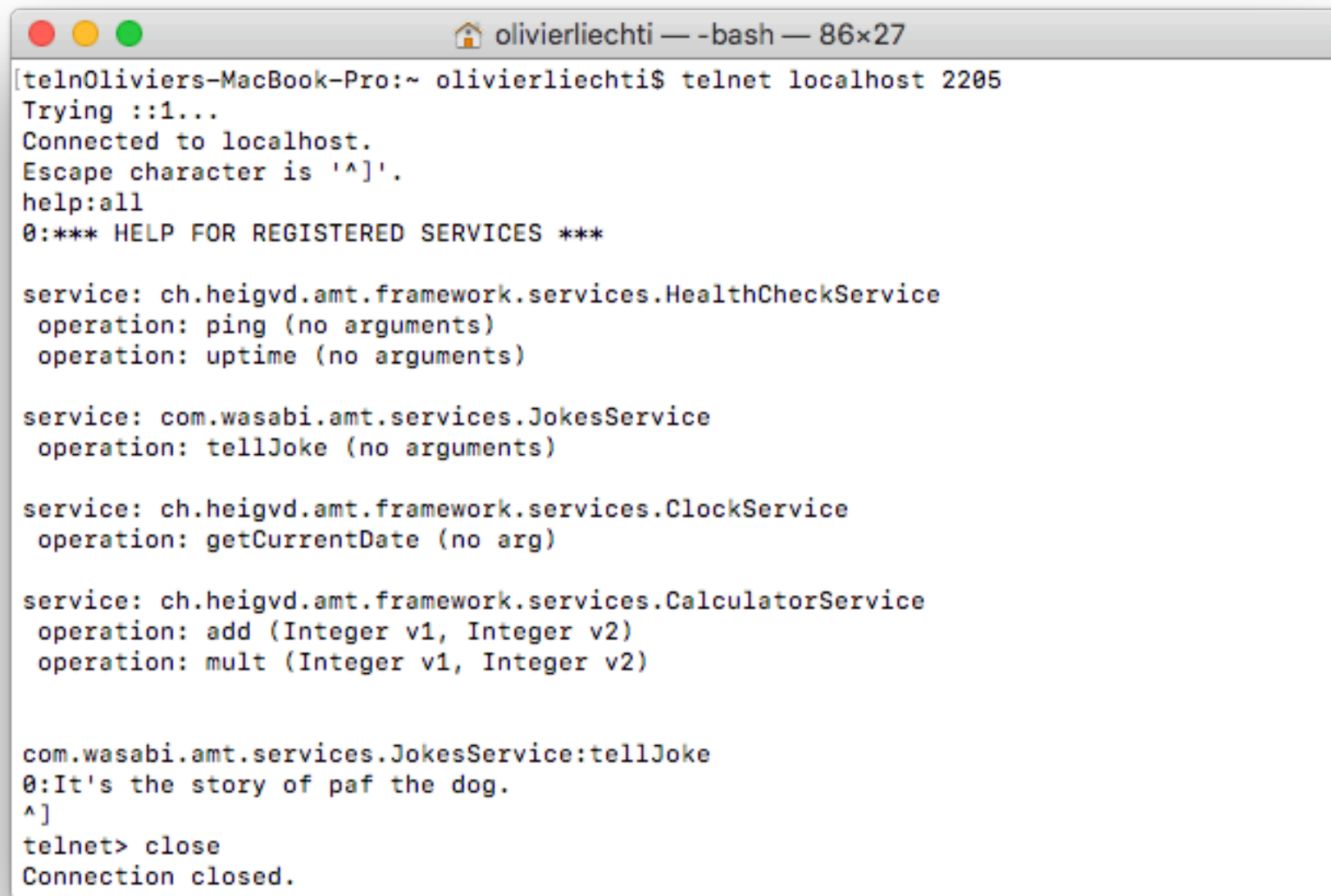
Update the server code

```
public static void main(String[] args) throws ClassNotFoundException, NoSuchMethodException,  
IllegalAccessException, InvocationTargetException, InstantiationException, IOException {  
    ServiceRegistry registry = ServiceRegistry.getServiceRegistry();  
    registry.register(HealthCheckService.class.getCanonicalName(), new HealthCheckService());  
  
    IService service = ((IService)  
Class.forName("ch.heigvd.amt.framework.services.ClockService").getDeclaredConstructor().newInstance());  
    registry.register(service.getClass().getCanonicalName(), service);  
  
    service = ((IService)  
Class.forName("ch.heigvd.amt.framework.services.CalculatorService").getDeclaredConstructor().newInstance());  
    registry.register(service.getClass().getCanonicalName(), service);  
  
    service = ((IService)  
Class.forName("com.wasabi.amt.services.JokesService").getDeclaredConstructor().newInstance());  
    registry.register(service.getClass().getCanonicalName(), service);  
  
    Server server = new Server();  
    server.start();  
}
```



Run the server with full classpath

```
java -cp amt-application/target/amt-application-1.0-SNAPSHOT-shaded.jar:./third-party-libs/* ch.heigvd.amt.framework.engine.Server
```



```
olivierliechti — -bash — 86x27
[telnOliviers-MacBook-Pro:~ olivierliechti$ telnet localhost 2205
Trying ::1...
Connected to localhost.
Escape character is '^]'.
help:all
0:*** HELP FOR REGISTERED SERVICES ***

service: ch.heigvd.amt.framework.services.HealthCheckService
  operation: ping (no arguments)
  operation: uptime (no arguments)

service: com.wasabi.amt.services.JokesService
  operation: tellJoke (no arguments)

service: ch.heigvd.amt.framework.services.ClockService
  operation: getCurrentDate (no arg)

service: ch.heigvd.amt.framework.services.CalculatorService
  operation: add (Integer v1, Integer v2)
  operation: mult (Integer v1, Integer v2)

com.wasabi.amt.services.JokesService:tellJoke
0:It's the story of paf the dog.
^]
telnet> close
Connection closed.
```