Welcome to ACS TA Session 1

Julian, Leonardo, Li, Nikolaus, Rodrigo, Tilman, Yijian

Department of Computer Science, University of Copenhagen

Academic year 2021 - 2022, Block 2

25 Nov 2021

Overview of TA sessions

Schedule

*Thursday from 13:15 to 15:00

*Universitetsparken 5, HCØ, øv -

A103 (Julian), A106 (Nikolaus), A110 (Tilman), A112(Yijian/Rodrigo)

Agenda

*Exercises on the ACS topics covered in the week

*Q & A for clarifications on the topics

*Work on the assignment and get unstuck if needed

(Feel free to open a discussion on Absalon if you have any questions)

Agenda for today

- Java practices
- Walk through the code of a sample project
  - Remote procedure calls using the Jetty framework
  - Test using the JUnit framework
  - Discuss local calls vs RPC
- Get started with your first programming assignment
- Q & A

Java

- Object-oriented programming
- Everything in Java is (ultimately) a reference
  - No pointer handling or (direct) memory allocation
  - Primitive types: `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` and `double`
  - Reference types: all other types

Primitive type	Reference type

`int a = 1;`
`int b = a;`
`b++;`
`System.out.println("a = " + a);`
`System.out.println("b = " + b);`
 	
`a = 1`
`b = 2`
	`int[] a = {1, 1, 1};`
	`int[] b = a;`
	`b[0]++;`
	`System.out.println("a[0] = " + a[0]);`
	`System.out.println("b[0] = " + b[0]);`
	
	`a[0] = 2`
	`b[0] = 2`

Java - data encapsulation

- Encapsulation needs proper thought
  - What data should be encapsulated together
  - What (object) references are exposed through your abstractions

  (public, private, protected)

Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Reference: https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Java - data encapsulation

- Immutability
  - encapsulated mutable data are not shared

    eg. acertainbookstore/business/BookRating.java

```java
public class BookRating {

    /** The ISBN. */
    private int isbn;

    /**
     * Sets the ISBN of the book.
     *
     * @param iSBN
     *            the new ISBN
     */
    public void setISBN(int isbn) {
        this.isbn = isbn;
    }
```

Modify data through the specific interface

Java - data encapsulation

- Immutability
  - make a deep copy before manipulating the object

copy constructor

```java
public class Employee {
    private int id;
    private String name;
    private Date startDate;

    public Employee(Employee employee) {
        this.id = employee.id;
        this.name = employee.name;
        this.startDate = new Date(employee.startDate.getTime());
    }
}
```

The constructor takes an object of the same type as a parameter

Make sure every field makes a deep copy

Reference: https://www.baeldung.com/java-copy-constructor

Java - data encapsulation

- Immutability
  - make a deep copy before manipulating the object

    cloneable interface

```
class A implements Cloneable {

    @Override
    protected Object clone()
        throws CloneNotSupportedException
```

```
A b = (A)a.clone();
```

Java - thread safety

- Thread safety
  - Multiple threads may get access to the same data at the same time
  - Concurrent threads harm data integrity
  - How to achieve thread safety?

    synchronized keyword: only one thread gets access to the block of code

    ```
    public synchronized void addBooks(
    ```

    locking (eg. ReentrantReadWriteLock)

    etc…
  - In ACS, you'll learn more about concurrency control

Test-driven development

- Recommended team organization
  - Consider “pairwise” programming
  - One person writes the tests, another person writes the codes
  - Alternate roles for different systems under test
- Recommended order of steps
  - Both design and implement the API for the component
  - A writes failing test cases for this API
  - B implements the component step by step
  - In effect of the implementation, all tests gradually pass

Test-driven development

- Isolate the test cases from each other
- Isolate the component under test from dependencies (for unit tests)
- Do not isolate the component under test from dependencies (for integration tests)
- Test sunshine, dark and unexpected scenarios
- Test corner cases and values e.g.,
  - For numerics, test the maximum, minimum, zero, other thresholds as appropriate
  - For strings, test the empty string and null
  - For IDs, test both existing and missing IDs
  - For time or space complexities of functions, clamp f(n) for a sufficiently large, but tractable n
- Consider handling expected exceptions, e.g., using
  - @Test(expected=SomeException.class)
- Consider using a test coverage tool, e.g., www.eclemma.org

Sample project — BankAccount

- Download the sample project from Absalon

  The sample project has the same architecture as acertainbookstore.

  It is important that you understand the sample project before doing programming assignment 1.
- Install an IDE you like
- Install JDK (version 10 – 15 work well)
- Import the project to your IDE (we use Eclipse as an example)
  - Import… → General → Projects from Folder or Archive → choose `bankaccount` folder / archive
- **Make sure you are using the proper Java Compiler**

  Project → Properties → Java Compiler → Enable project specific settings → Compiler compliance level: 10/11/12/13/14/15 → Apply and Close
- **Make sure you are using the proper JRE**

  Preferences → Java → Installed JREs → choose Java SE 10/11/12/13/14/15 → Apply and Close

  If your installed JDK does not appear in this list, you need to Add it manually (ask TA if you need help)
- Let's walk through the code together!

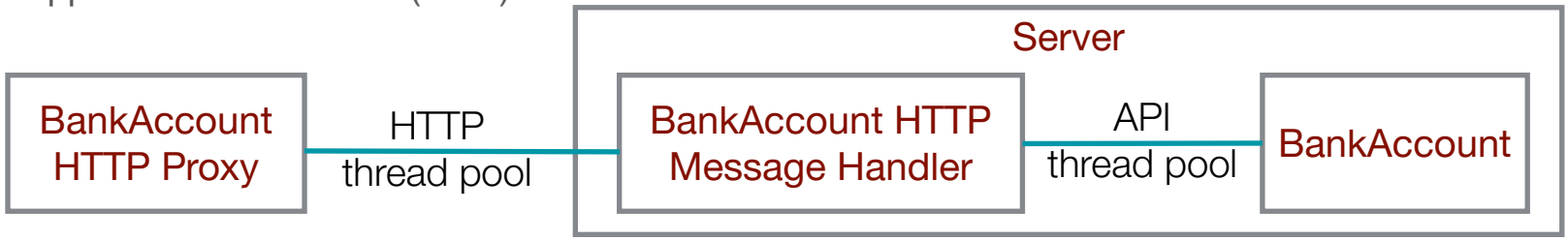
Sample project — BankAccount

- Application building blocks
  - Server (app logic): `BankAccount.java`
  - Local calls vs RPC (two modes can be chosen)
  - RPC (a mechanism to interact with the server)

    RPC uses HTTP for communication

    Jetty: provide HTTP server and HTTP client libraries

    Server is hosted in the `BankAccountHTTPServer.java`
  - Tests: JUnit testing framework
- Application architecture (RPC)



JUnit testing framework

- Setup
  - Use @Test to annotate each of your test case
  - Notice: test cases are not running in the programming order
  - Remember to setup the system state before running tests
  - @BeforeClass, @AfterClass (run only once before/after all tests)
  - @Before, @After (run before/after each test)

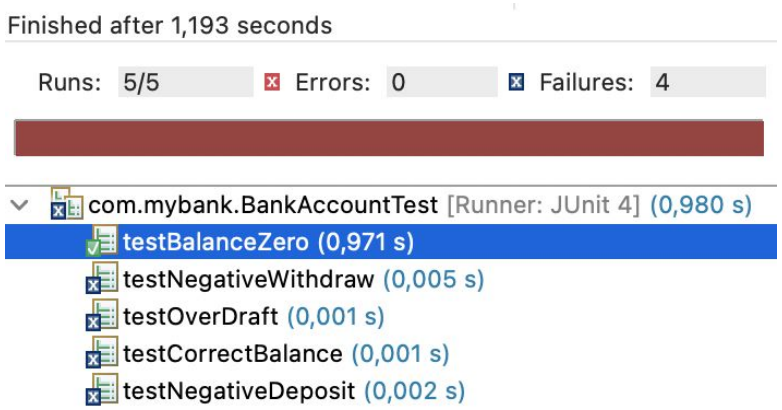
eg. initialize the `client` before running any tests

```
@BeforeClass
public static void setUpBeforeClass() throws Exception {
    if (localTest) {
        client = new BankAccount(TEST_CPR);
    } else {
        client = new BankAccountHTTPProxy("http://localhost:8081");
    }
}
```

JUnit testing framework

- Run tests
  - Set local / non-local (RPC) mode
  - In non-local mode, always remember to start the server first!

    run `BankAccountHTTPServer.java` as `Java Application`
  - run `BankcAccountTest.java` as `JUnit Test`
  - JUnit will report the test results
  - Now we have 1 test passed and 4 failed



JUnit testing framework

- Assertions
  - Use assertions to test if conditions are met
  - A test case will pass if the assertion turns out to be true
  - Different assertions supported by JUnit:

```
assertTrue(String message, boolean condition)
assertEquals(String message, TYPE expected, TYPE actual)
assertNotNull(String message, Object obj)
assertNull(String message, Object obj)
assertSame(String message, Object expected, Object actual)
fail(String message)
……
```

Local calls vs RPC

- Local calls
  - Are server and client running in one process?
  - Are server and client running on the same JVM?
  - What's the impact on the client if the server fails?
  - What if the host machine crashes?
- RPC
  - Are server and client running in one process?
  - Are server and client running on the same JVM?
  - What's the impact on the client if the server fails?
  - What if the host machine crashes?
  - What if client and server are deployed on two different machines?

BankAccountTest.java

```
if (localTest) {
    client = new BankAccount(TEST_CPR);
} else {
    client = new BankAccountHTTPProxy("http://localhost:8081");
}
```

Local calls vs RPC

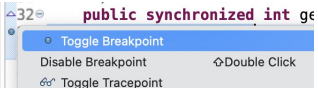
- Local calls
  - Are server and client running in one process? — yes
  - Are server and client running on the same JVM? — yes
  - What's the impact on the client if the server fails? — client also fails
  - What if the host machine crashes? — both client and server crash
- RPC
  - Are server and client running in one process? — no
  - Are server and client running on the same JVM? — no
  - What's the impact on the client if the server fails?

    — client can work, but the server becomes unavailable
  - What if the host machine crashes? — both client and server crash
  - What if client and server are deployed on two different machines?

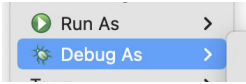
    — the crash of one machine will not affect the other machine

Debugger

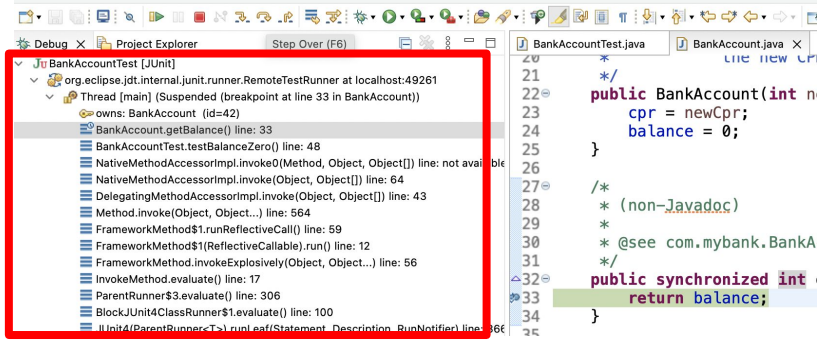
- Use debugger (built in your IDE) to understand the call hierarchy
  - STEP 1: set line breakpoints



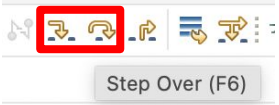
  - STEP 2: run the code under debug mode



  - STEP 3: the process will stop at the breakpoint



  - STEP 4: move forward step by step to trace the call chain



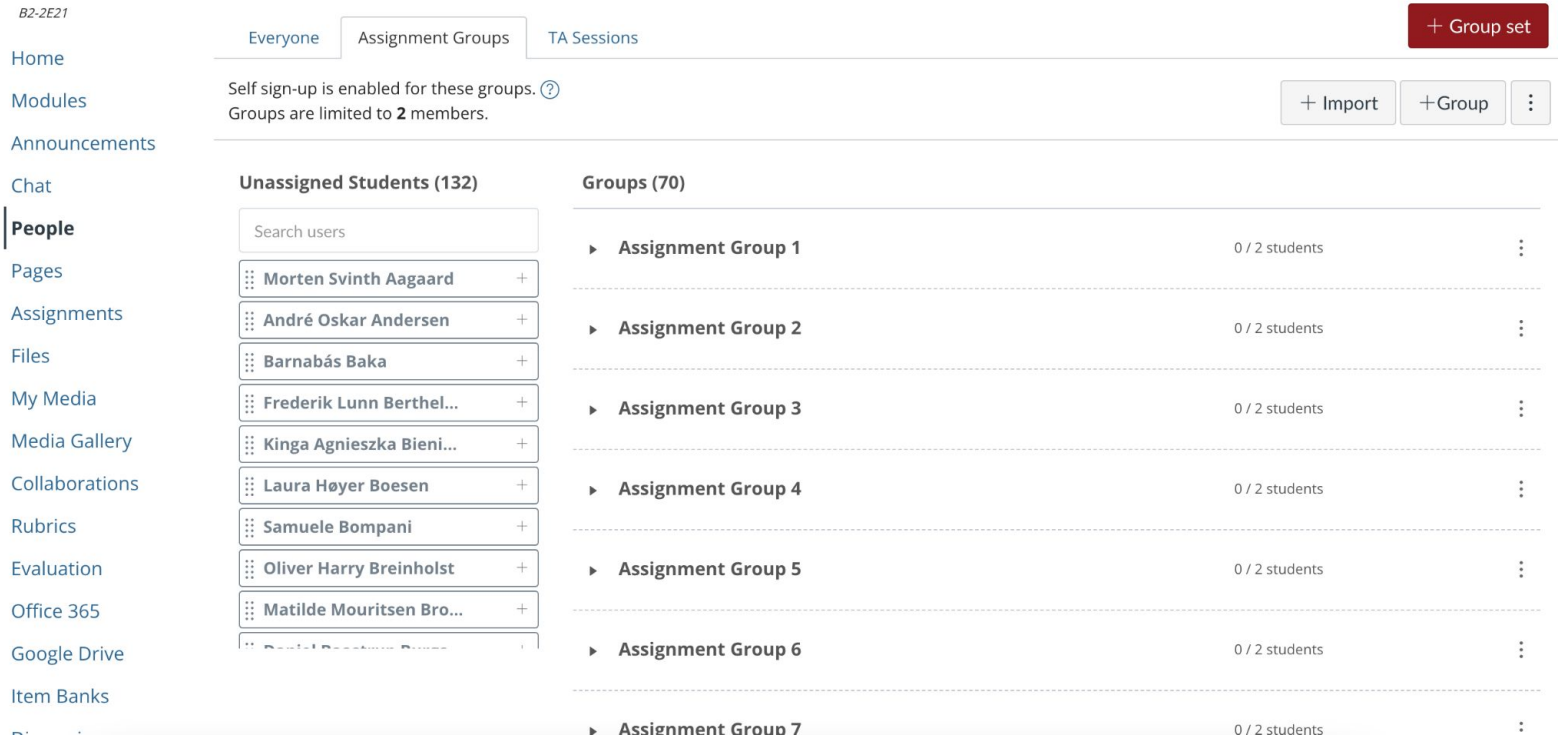
Tools to monitor JVM

- monitor CPU, threads and memory usage of Java applications

- **JConsole**
  - It can monitor local / remote JVMs
  - Start a terminal
  - Enter into the folder where you installed your JDK
    eg. `:/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk/Contents/Home/bin`
  - Run `JConsole` on your terminal

- **JVM Monitor**
  - It is integrated with Eclipse
  - It can automatically find the running JVMs on local host
  - Help → Eclipse Marketplace → Search “JVM Monitor” → Install
  - Window → Perspective → Open Perspective → Other… → Java Monitor —> Open

Register your assignment group on Absalon!



Thank you

Julian, Leonardo, Li, Nikolaus, Rodrigo, Tilman, Yijian