A Terms of cooperation

Obtaining a credit

Each task has a **first deadline** specified in its title. A student may obtain maximum score for solution to an assignment in case he/she presents it during the laboratory class on the first deadline at the latest. Assignments are, in general, evaluated for maximum 10 points. Solutions to some specific assignments may be assessed for more points, which is discussed in the text of the particular tasks.

The **second and very last opportunity to present a solution to an assignment** is the second deadline – i.e. laboratory class next to the first deadline class. Defending a solution to an assignment during a **second deadline** – i.e. laboratory class next to the first deadline class – entitles a student to obtain 50% of the maximum original score at the most.

Evaluation of student solutions

While presenting a solution to an assignment a student should **demonstrate that he/she is aware of its internal architecture** (the applied technologies and combining them or used algorithms).

During the presentation a student should be prepared to answer some control question asked by the person who evaluates the solution. In case the person assessing the solution is not convinced of student's authorship, the **student gets 0 points for the solution and loses any further opportunity to defend the solution to the given assignment**.

The above statement implies that a student **may fail in getting a credit** in case he/she attempts to defend a solution to a mandatory task (see below) and the person evaluating the solution comes to conclusion that the student has not authored the presented solution.

Robustes of student solutions

The presented solutions should validate the input data provided by the user. **The undesired behaviours which entail reducing of awarded numer of points during assessment include i.a.:**

- no verification whether a user passed a valid file path or whether file exists;
- missing validation of the input format – in case the format is known in advance (e.g. format of postal code);
- missing verification, whether value of a reference variable/field was set (i.e. avoiding of NullPointerException);
- missing verification whether value of array index does not exceed the actual size of the array/list (ArrayIndexOutOfBoundsException).

Evaluation

- **3.0 – 25÷35 points**
- **3.5 – 36÷55 points**
- **4.0 – 56÷85 points**
- **4.5 – 86÷120 points**
- **5.0 – 121+ points**

Minimum requirements for getting credit (mandatory tasks)

To get a credit you need to receive at least 5 points for each of the assignments concerning the following topics:

1. Non-blocking I/O **(assignment 1)**
2. Client/Server based on NIO channels **(assignment 2)**
3. Web Applications – introduction **(assignment 3)**
4. either (1) Remote Method Invocation **(assignment 8)** – or alternatively (2) Java Messaging Service **(assignment 9)**
5. Web Services – introduction **(assignment 10)**

1 Non-blocking I/O – shared mapped file

Deadlines

Tuesday 15th March

Write simple two applications (it may be one application which operates in two modes) which communicate with each other by saving and reading data from a shared file mapped with into memory.

One of the application should save some data – e.g. two integer values. The other application should retrieve the data and perform some operation – e.g. adding the values read – and then print out the result to the console.

Reader and writer should run in a continuous mode – i.e. writer should attempt writing data to the shared file for specific number of iteration. On the other hand the reader should await for the date originated from the writer until it makes sense – writer is still working.

Writer must not overwrite data put previously by itself into the file until reader fetches previous data set (or a message) from the file.

Analogously reader must not re-read data which it has already read from the file. In other words reader should read particular data set (or a message) only once and wait for the next one in case it has not been overwritten by the origin (i.e. writer).

NOTE: Your solution should be based on non-blocking I/O implementation.

2 Client/Server – custom protocol for echoing messages and adding values

Deadlines

Tuesday 22nd March

Write a client/server solution which implements a simple (human-readable) protocol which supports two basic operations:

- echoing messages – the server echoes the message sent by the client;
- adding values of two operands and returning the result to the requestor (client).

NOTE: Your solution should be based on non-blocking I/O implementation.

3 Web Applications – Introduction

Deadlines

Friday 29th March

Develop a simple Java™ servlet which adds two integer values passed by a user as parameters to a web form and prints the result below the mentioned form. The user of your application should be capable to pass parameters to the servlet either with GET or POST method – i.e. you could for instance put two different submit buttons for each of the HTTP method.

NOTE: make your servlet robust to invalid input – a user may provide parameter values which may not be parsed into integer.

4 Web Applications – Model-View-Controller

Deadlines

Friday 5rd April

Modify the architecture of the solution for assignment 3 so that it complies with the Model-View-Controller concept implementation for web application as discussed during lecture.

5 Web Applications – database access (15 pts)

Deadlines

Friday 12th April

Develop a simple web application which allows particular users to display the resources available to them.

The application would hold user directory in a database table ‘Users’. A ‘Users’ entry contains: technical (so called surrogate) identifier, login, first name, surname and the password.

The resources are stored in ‘Resources’ database table. Each resource has: surrogate identifier, name and content – i.e. a 2048-byte long string.

A resource may be accessible to multiple users, as well as each user may access multiple resources, which implies that in the relational model there should be a junction table between the two above entity types.

Your web application should allow a user to establish a session after successful authentication. Within a session a user can browse the resources he/she has access to. This means that your application should provide 3 different user screens:

(1) entry login screen for user authentication;
(2) a screen listing resources available to the user – each resource entry presents only name of a resource – the resource details page (see below) is displayed when a user follows a corresponding link or presses a ‘Details’ button besides the name of the resource;
(3) a screen presenting details of particular resource displaying both the name and the content.

NOTE: Screens (2) and (3) should contain a ‘logout’ button which invalidates the session. Screen (3) should have a button/link to navigate back to the list of the available resources.

NOTE: Your solution should utilize DataSource mechanism which supports database connection pooling.

6 Web Applications – AJAX

Deadlines

Friday 19th April

Implement solution enabling verification whether string matches entered Java regular expression. Create a web page with two input text boxes.

The upper textbox will allow user to enter a pattern complying with Java regular expression syntax.

The bottom input textbox will enable passing a string to match against the pattern entered in the first text box. The bottom input text box should be disabled in case the upper is empty.

The result whether the input string matches the regular expression should be presented below. The information presented to the user should enumerate the parts of the input string matching the regex in the following format:

"<matched-input-part>" (start index: <start-index>, end index: <end-index>)

In case there are no matches found the following message should be displayed below both input textboxes:

No matches have been found.

In case the entered pattern does not meet the syntax of Java regular expression an adequate message should be displayed instead of the result:

The entered pattern is not compatible with Java regular expression syntax.

User interface should not contain any buttons – the evaluation of the result should occur after any of the textboxes loses the focus. The application should be based on single-page application design concept. The evaluation should be based on AJAX – i.e. request for evaluation should be sent in the background by JavaScript code handling events raised by input textboxes.

7 Web Applications – Filters and Events

Deadlines

Friday 26th April

Modify your web application developed as the solution to assignments 3, 4, 5 or 6 so that it complements the generated web page with (1) a header, and (2) a footer added by filters.

The footer will print out the current date which will be updated each second.

Add handlers to the following events:

(1) servlet context initialization and destruction;
(2) request construction and destruction.

Your handler will print out some notification to web-container log that the particular event has been raised.

Request construction event will print out the request content.

8 RMI – echoing messages and adding values

Deadlines

Friday 10th May

Write a client/server solution based on RMI technology which implements the operations discussed in task 2:

- echoing messages;
- adding values of two number operands.

For both features create separate classes representing requests and responses (e.g. EchoRequest/EchoResponse and AddRequest/AddResponse).

NOTE: Passing or getting results as values of 'simple' types (i.e. primitive types or strings) is not satisfactory.

9 Java Messaging Service – asynchronous message processing (15 pts)

Deadlines

Friday 17th May

Write a solution based on JMS technology composed of two types of parties:

- 10 requestor threads run in a pool sending requests and
- 5 service threads run in a pool responsible for processing the received requests.

Your solution should support the following scenario:

1. Requestor prepares the body of request message.
2. Requestor puts a request message into a JMS destination and awaits result of processing.
3. An idle service retrieves the request message from the JMS destination.
4. Service processes the request – processing should take a random amount of time between 3 and 5 seconds.
5. Service puts the response message into a JMS destination.
6. Requestor retrieves the message from JMS destination and consumes the result. Please note that requestor should consume only the response of the request it has put in step (2).

All the above steps should be logged with standard Java™ logger – each log entry should be a single line of the following format: '**hh:mm:ss.SSS <participant>: <log-message>**'.

A sample log message could look as follows:

12:31:34.678 REQUESTOR-1: request put into destination

Your application should support two different types of requests:

1. Generating random number;
2. Performing a simple two-argument arithmetic operation: addition, subtraction, multiplication or division.

Both types of requests/responses should be your custom types:

- RandomRequest/RandomResponse;
- ArithmeticRequest/ArithmeticResponse.

Please note that ArithmeticRequest should specify the type of operation to be performed.

The requestors and services should be started from one entry point (i.e. method *main*).

During a session each requestor should put 3 different requests and consume the processing results.

```
# SAMPLE Apache ActiveMQ configuration


java.naming.factory.initial                                                    =
org.apache.activemq.jndi.ActiveMQInitialContextFactory


# use the following property to configure the default connector

#java.naming.provider.url = vm://localhost

java.naming.provider.url = tcp://localhost:61616


# use the following property to specify the JNDI name the connection factory

# should appear as.

#connectionFactoryNames     =     connectionFactory,     queueConnectionFactory,
topicConnectionFactry


# register some queues in JNDI using the form

# queue.[jndiName] = [physicalName]

queue.Queue = pl.edu.pjwstk.tpr.jms.Queue



# register some topics in JNDI using the form

# topic.[jndiName] = [physicalName]

topic.Topic = pl.edu.pjwstk.tpr.jms.Topic
```

```
# SAMPLE log4j configuration required by Apache ActiveMQ

#log4j.rootLogger=INFO, A1, A2
log4j.rootLogger=INFO, A2
#log4j.rootLogger=DEBUG, A1, A2

#log4j.rootLogger=INFO, A2

#log4j.appender.A1=org.apache.log4j.ConsoleAppender
#log4j.appender.A1.layout=org.apache.log4j.SimpleLayout
#log4j.appender.A1.layout=org.apache.log4j.PatternLayout
#log4j.appender.A1.layout.ConversionPattern=%d %5p [%t] (%F:%L) - %m%n
#log4j.appender.A1.layout.ConversionPattern=%d{HH:mm:ss,SSS} %5p [%t] %C{1}.%M
- %m%n

log4j.appender.A2=org.apache.log4j.RollingFileAppender
log4j.appender.A2.MaxFileSize=1MB
log4j.appender.A2.File=C:/Temp/edek-jms.html
log4j.appender.A2.Append=false
log4j.appender.A2.layout=org.apache.log4j.HTMLLayout
```

10 XML Web Services – Introduction

Deadlines

Friday 24st May

Based on the samples for JAX-WS implementation provided for the lecture create your own SOAP web service which will hold information about people (e.g. in a map). Each person has the following attributes:

- First name;
- Surname;
- Birth date.

Your web service will respond to a query which will filter the stored entries based on:

- Surname;
- Birth date.

Based on the provided samples create a JUnit test for verifying whether your SOAP web service works as expected.

11 XML Web Services – MTOM transport (15 pts)

Deadlines

Friday 28th May

Based on samples provided for JAX-WS create a SOAP web service enabling uploading and downloading files. Each file uploaded to your web service should be stored along with its metadata:

- File name;
- File size;
- Keywords describing file content.

Your web service should support three types of requests:

- File upload – used for file uploading. Some of the aforementioned properties, such as file name or file size, could be retrieved from the file itself. Keywords should be provided as separate set of values in the request.
- File query – used for filtering files of the given keywords.
- File download – for downloading file based on file name.

Based on the provided samples create a JUnit test for verifying whether your SOAP web service works as expected.

12 Web Services – MTOM transport and JPA data access (25 pts)

Deadlines

Friday 11th June

Based on the samples on JPA discussed during the UTP course extend your web service developed as the solution to assignment 10 so that information about the people is stored in a relational database.

Access to the database should be based on Java™ Persistence API.

Extend the data model of your application so that information about each person is enriched with his/her picture.

Create a simple GUI application which enables browsing information about the people – i.e. presents his/her personal data along with the picture of him/hers.

HINT: Store person picture as a BLOB object in the database.

13 WebAPI – Introduction

Deadlines

Friday 17th June

Based on the samples for JAX-RS implementation provided for the lecture create your own WebAPI aka. REST web service which will hold information about people (e.g. in a map). Each person has the following attributes:

- First name;
- Surname;
- Birth date.

Your web service will respond to a query which will filter the stored entries based on:

- Surname;
- Birth date.

Based on the provided samples create a JUnit test for verifying whether your REST web service works as expected.