## QUEUE TEST TASK

Original task 2

Concepts 3

Formula – extended 3

Input and output messages – exchanging messages with AMQP server 4

JAVA Packages and code examples 4

Service configuration file 6

WORKPLAN 6

NB! GOOD CODE CONTAINS 6

# Original task

Task

Create a small service which does the following:

1. Connects to AMQP (RabbitMQ) server (details below)
2. Listens on **'interest-queue'** queue in default exchange for messages
3. For each message it calculates the "interest" and total sum by formula given below
4. Broadcast the new messages to **'solved-interest-queue'** in the same exchange

If everything is done correctly you should be able to see the messages popping up in the display table above.

Interest formula

1. Interest is calculated based on *sum* and *days* fields
2. Interest is calculated per day as a percentage from the original amount
3. If day is...
   1. divisible by three, the interest is: **1%**
   2. divisible by five, the interest is: **2%**
   3. divisible by both three and five, the interest is: **3%**
   4. not divisible by either three or five, interest is: **4%**
4. Each day interest amount is rounded to two digits
5. Final interest is a sum of all days interests
6. Total sum is the sum of original amount and total interest

Message Format

Messages are transmitted as JSON.

Incoming messages will look like following:{ sum: 123, days: 5 }

Outgoing messages should look like following:{ sum: 123, days: 5, interest: 18.45, totalSum: 141.45, token: "myIdentifier" }

Token will be displayed on the monitor above for clarity when several services are running at the same time. Use your name, nick, or something else clever.

AMQP server (RabbitMQ) details

**Server:** impact.ccat.eu  
**User:** myjar  
**Password:** myjar

References

* <https://www.rabbitmq.com/>
* <https://github.com/videlalvaro/php-amqplib>
* <http://json.org/>

# Concepts

* Server – provides service with input and output messages
* Service – listens to queue messages from the server, handles it with specific task and sends an output message to the server queue
* Input message – message that contains the sum and number of days in **JSON format**
* Output message – message that contains the data from input message and the calculated interest and totalSum, also in JSON format
* Calculator – handles the data from input message to calculate the interest and totalSum using the specified formula

# Formula – extended

* Input:
  + Sum
  + Days
* Output
  + Sum
  + Days
  + Interest
  + Totalsum
* Interest is a total summation of each day
* Each day has its own interest percentage and a corresponding sum which is a percentage from the **original amount**
* Totalsum is the sum of original amount and total interest
* Interest for each day is calculated as follows
  + Day is divisible by **3**, the interest is **1%**
  + Day is divisible by **5**, the interest is **2%**
  + Day is divisible by **both**, the interest is **3%**
  + Day is not divisible by either, the interest is **4%**
  + **E.g.** if the interest needs to be calculated for 5 days, the interest percentages would be> 4% for day 1, 4% for day 2, 1% for day 3, 4% for day 4, 2% for day 5
* First draft of code >

getInterest( sum, days ) {

interest = 0;

foreach (days) {

interest += getPercentage(day) of sum;

}

Return interest; // round to two digits

}

getPercentage ( index of day) {

if ( day !%3 && day !%5)

return 4;

else

percentage = 0

if (day %3)

percentage += 1

if (day % 5)

percentage += 2

return percentage;

}

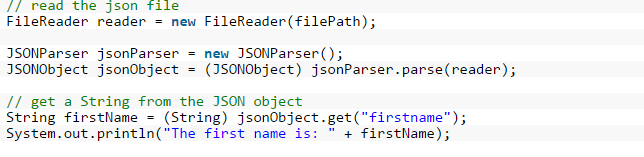
# Input and output messages – exchanging messages with AMQP server

Message formats

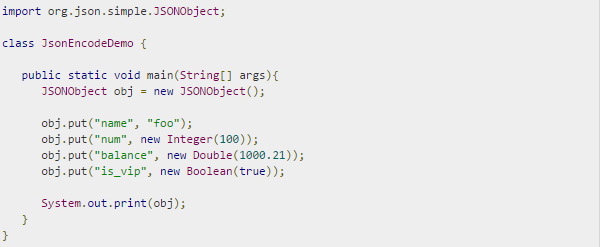
* Input - {sum: <sum>, days: <days>}
* Output – {sum: <sum>, days: <days>, interest: <getInterest(sum, days), totalSum: <sum+interest>, token: <themerru©>}

# JAVA Packages and code examples

* **Json**

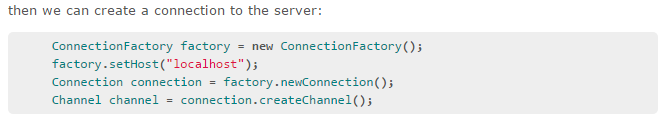


Code example - json deconding {http://examples.javacodegeeks.com/core-java/json/java-json-parser-example/ [3.10.2015]}

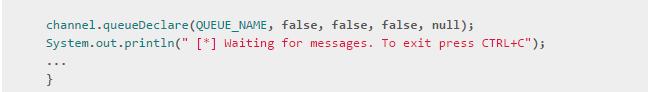


Code example 2 - json encoding {http://www.tutorialspoint.com/json/json\_java\_example.htm [3.10.2015]}

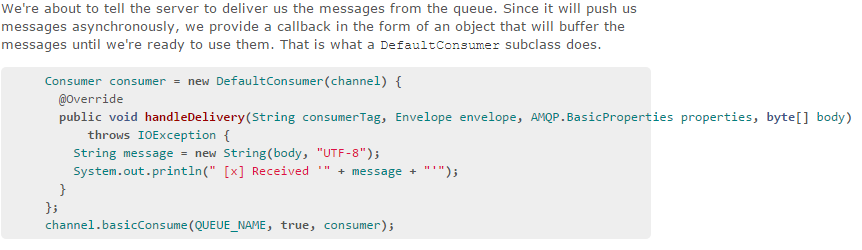
* RabbitMQ Java Client
  + the groupIdcom.rabbitmq and the artifactId amqp-client



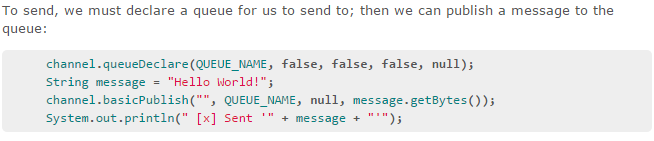
Code example 3 – Connect to server {https://www.rabbitmq.com/tutorials/tutorial-one-java.html [3.10.2015] }



Code example 4 - waiting for messages { https://www.rabbitmq.com/tutorials/tutorial-one-java.html [3.10.2015] }



Code example 5 - handling messages { https://www.rabbitmq.com/tutorials/tutorial-one-java.html [3.10.2015] }



Code example 6 - sending messages { https://www.rabbitmq.com/tutorials/tutorial-one-java.html [3.10.2015] }

# Service configuration file

* server connection information
* input queue
* output queue
* token
* nrOfMessages [ -1 > until stopped]

# WORKPLAN

* Tested calculator
* Receive input messages
* Compile tested output messages
* Publish output messages
* Define configuration outside of service
* Starting and stopping the service

# NB! GOOD CODE CONTAINS

* exception handling
* validations of input
* tests
* documentation
* no hardcoding if possible