Advanced Java with Java 8 Labs

# Lab 1

Create 4 interfaces:

1) public void printSquareOfA(int a);

2) public int getSquareOfA(int a);

3) public int getAxB(int a, int b);

4) public double getPi();

1) Implement a lambda that squares itself and prints it (lambda block)

2) Implement a lambda that returns the square of itself (lambda expression)

3) Implement a lambda that multiplies the two numbers (lambda expression)

4) Implement a lambda that returns 3.14 (lambda expression)

# Lab 2

Refactor 4 interfaces from lab 1 to @FunctionalInterfaces (1, 2, 3, 4)

Refactor to use static method references where possible (2, 3)

Refactor to use constructor references (not 4, but 5)

# Lab 3

Refactor 4 interfaces and use a default implementation for each

Call the default implementation

# Lab 4

From lab 1, refactor the four into the standard functional interface

Consumer, IntConsumer

Function, IntFunction

BiFunction, BinaryOperator,

Supplier, DoubleSupplier

# Lab 5

A) Use functional composition to implement this rule:

1. Must pass all exams (> 60%)
2. Must have a B average
3. Or last exam was perfect
4. Must have taken all exams

B) Use Functions to create a series of functions that:

* Double, square, cube then negate a number using andThen
* Double, square, cube then negate a number using compose

# Lab 6

Write a simple chat server:

* ChatReceiver sets up a server socket on port 8080
* Use telnet to connect to it
* Send your name to establish connection on the telnet side
* Receiver wait for telnet to send name, then responds with its own name
* Telnet side then starts conversation
* Only one side at a time can speak while the other is waiting

# Lab 7

Create a CRUD application that stores a movie database

* Movies can be added, found by name, returned by category and deleted
* Movies have categories: comedy, horror, drama, romance

# Lab 7.1

Quick lab for executor service

- Write an executior service with submit, call, future

- Determines the prime number within range

# Lab 8

Refactor lab 7 and make it thread safe

* Use Read/write locks to make adding/finding/removing thread safe
* Keep a log of every access (start time, end time) This can be in a synchronized list
* Make sure you get no concurrent modifications error
* Make sure you get no overlap of read and write

# Lab 9

Threaded chat server:

* Write a chat server and use telnet as client
* Chat server has chat rooms (sports, politics, movies)
* Chat client connects and enters chat room,line feed, name line feed
* Now it can send and receive messages to all other participants
* Chat server is shutdown when someone connects and sends "SHUTDOWN"

# Lab 10

Do lab 7 by using functions instead

# Lab 11

Use streams to implement all of these

1) Print all even numbers from 0 to 100

- Then, modify your algorithm to add only odd numbers 0, 100

- Then, modify your algorithm to add only odd numbers 0, 100 but remove prime numbers

2) Go back to lab5 and change the implementation of the predicate composition using streams

- Keep the compositional portion intact - just change the imperative code to streams

3) Grep from BufferedReader

- Count occurrences of a given search word

- Return line for each occurrence of word (regular grep)

4) Given a list of strings, return each that is a palindrome.

- Modify your algorithm to return the original word (not inverted)

5) Implement the Fizz Buzz algorithm

- Iterate from 1 to 100, print "Fizz" for every number divisible by 3 and "Buzz" for every number divisible by 5

6) Implement the game of life

# Lab 12

Use currying to create a grading system that uses average, best or worse test scores as final grade