1. Homework 9

Posted: October/22/2018
Due: October/28/2018 24.00

All homework solutions are due October/28/2018 24.00. I recommend to submit at least one version of all homework solutions long before due date.

1.1. Homework 9.1 (10 Points)

Objective: Getting familiar Java's File/IO

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

You have to implement a parts of the find utility which recursively descends the directory tree for each path listed.

Explanation:

Write a java program which can be used like a simplified version of find. The manual page of the *find* program can be found

The syntax of the your find is:

```
find  starting_directory [-name name|-type (f|d)]|-date|-length]
```

The semantic is of the above syntax is:

```
$ find . -name thisName   # print all files with name:
                          # thisName starting in current dir
$ find . -type f          # print all files starting in current dir
$ find . -type d          # print all files starting in current dir
$ find . -type d -date    # print all directories starting in
# current dir and the last modification time
$ find . -type d -length  # print all directories starting in
                          # current dir and length of the file
```

Your Work:

find is using a recursive algorithm. There is no reason for you to use a recursive algorithm.

These are examples of my implementation:

```
% java Find.java
% java Find --printFile --directory A --printDate --printLength
% java Find --directory A --printDate --printLength
% java Find --directory A  --printLength=true
```

Requirements:

You have to name your file *Find.java*. Your program has to throw an exception if error conditions arise. You have to create some of the required exception classes.

Example:

An example of a solution execution:

Assuming the directory *A* has the following structure:

```
% find . -name A -exec ls -l {} ;
total 0
drwx------  4 hpb  staff  136 Oct 22 09:23 B
drwx------  3 hpb  staff  102 Oct 22 09:23 C
drwx------  2 hpb  staff   68 Oct 22 09:23 D
-rw-------  1 hpb  staff    0 Oct 22 09:23 aa
-rw-------  1 hpb  staff    0 Oct 22 09:23 aaa
```

My implementation would execute like:

```
% java Find --printFile --directory A --printDate --printLength
    Mon Oct 22 09:23:12 EDT 2018 238
:aa Mon Oct 22 09:23:12 EDT 2018 0
:aaaMon Oct 22 09:23:12 EDT 2018 0
    Mon Oct 22 09:23:12 EDT 2018 238
    Mon Oct 22 09:23:12 EDT 2018 136
:B:bb   Mon Oct 22 09:23:12 EDT 2018 0
:B:bbb  Mon Oct 22 09:23:12 EDT 2018 0
    Mon Oct 22 09:23:12 EDT 2018 238
    Mon Oct 22 09:23:12 EDT 2018 102
:C:cc   Mon Oct 22 09:23:12 EDT 2018 0
    Mon Oct 22 09:23:12 EDT 2018 238
    Mon Oct 22 09:23:12 EDT 2018 68
% java Find --directory A  --printLength=true
dirV:         A
    238
    0
    0
    238
    136
    0
    0
    238
    102
    0
    238
    68
```

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab9-1 ’All files required’
```

1.2. Homework 9.2 (10 Points)

Objective: Getting familiar with threads

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

Given is a number *n*. You have to design a program which finds all prime numbers *p*, $1 < p \leq n$. You must analyze your implementation based on speed.

Explanation:

thread 3 would strike out all multiples of 5. *p*, $1 < p \leq n$ are { 2, 3, 5, 7 }

Your Work:

n is given as command line argument.

Requirements:

You have to name your file *PrimeAsFastAsPossible.java*. Your program has to throw an exception if error conditions arise; like an negative number as a command line argument, or if the number is not an integer. You have to create these exception classes. Your program has to determine the optimal number of threads for your implementation.

Example:

An example of a solution execution:

```
% java PrimeAsFastAsPossible 3168885720
16 threads are optimal.
```

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab9-2 'All files required'
```

1.3. Homework 9.3 (10 Points)

Objective: Getting familiar with competing threads

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

Given is a fixed amount of storage space. Many producer is producing items which will be stored in the storage space, and many consumer removes items from the storage space and consumes the items. The producer can only produce items if there is space in the storage space to store it. The consumer can only take items if they're storage space has enough items. The producer produces *n* items each time the production starts. The consumer produces *k* items each time the production starts.

Explanation:

The producers, like the consumers can not be in busy loops. Each of them as to be notified by the other one. Your solution can not use *sleep()* to solve a problem. You can only use *wait(), notify()* and/or *notifyAll()*. You cannot use any existing storage solution. **Your Work:**

n, *k*, how many consumer, how many producer, and the length of the storage is give s command line arguments.

I suggest to start out and test your program with one consumer, one producer, then one consumer, two producer, then two consumer, two producer, then 1000 consumer, 1200 producer.

Requirements:

You have to name your file *ConsumerProducer.java*. Your program has to throw an exception if error conditions arise; like a negative number as a command line argument, or if the number is not an integer. You have to create these exception classes. Your program has to determine the optimal number of threads for your implementation.

Example:

An example of a solution execution:

```
% java ConsumerProducer 1 3 2 3 100   # 1 consumer consuming 3 items
                                        # 3 producer producing 2 items
                                        # 100 storage spaces
1: produce  2
2: produce  2
1: consumer 3
3: produce  2
 ....
```

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab9-3 'All files required'
```