# DD1339 Introduktion till datalogi 2013/2014

# Uppgift nummer: 6

# Namn: Marcus Larsson

# Grupp nummer: 5

# Övningsledare: Marcus Dicander

# Betyg: ..... Datum: .............. Rättad av: .......................................

# Exercise 4.73-4.78

/\*\*

\* Read web server data and analyse

\* hourly access patterns.

\*

\* @author David J. Barnes and Michael Kölling. Modified by Marcus Larson.

\* @version 2013.10.18

\*/

public class LogAnalyzer

{

// Where to calculate the hourly access counts.

private int[] hourCounts;

// Use a LogfileReader to access the data.

private LogfileReader reader;

/\*\*

\* Create an object to analyze hourly web accesses.

\*/

public LogAnalyzer()

{

// Create the array object to hold the hourly

// access counts.

hourCounts = new int[24];

// Create the reader to obtain the data.

reader = new LogfileReader();

}

/\*\*

\* Analyze the hourly access data from the log file.

\*/

public void analyzeHourlyData()

{

while(reader.hasNext()) {

LogEntry entry = reader.next();

int hour = entry.getHour();

hourCounts[hour]++;

}

}

/\*\*

\* Print the hourly counts.

\* These should have been set with a prior

\* call to analyzeHourlyData.

\*/

public void printHourlyCounts()

{

System.out.println("Hr: Count");

for(int hour = 0; hour < hourCounts.length; hour++) {

System.out.println(hour + ": " + hourCounts[hour]);

}

}

/\*\*

\* Print the lines of data read by the LogfileReader

\*/

public void printData()

{

reader.printData();

}

/\*\*

\* Return the number of accesses recorded in the log

\* file.

\*/

public int numberOfAccesses()

{

int total = 0;

for(int hour = 0; hour < hourCounts.length; hour++) {

total+= hourCounts[hour];

}

return total;

}

/\*\*

\* This method will find and return the busiest hour in the log.

\*

\*/

public int busiestHour(){

int max=0;

int busiestHour = 0;

for (int i = 0; i<hourCounts.length;i++){

if(hourCounts[i]>max){

max=hourCounts[i];

busiestHour = i;

}

}

return busiestHour;

}

/\*\*

\* This method will find and return the busiest 2-hour period in the log.

\* @return Returns the value of the first hour in the peroid.

\*/

public int busiestTwoHours(){

int max=0;

int busiestHour = 0;

for (int i = 0; i<hourCounts.length-1;i++){

int sum = hourCounts[i] + hourCounts[i+1];

if(sum>max){

max=sum;

busiestHour = i;

}

}

return busiestHour;

}

/\*\*

\* This method will find and return the quietest hour in the log.

\*

\*/

public int quietestHour(){

int min=0;

int quietestHour = 0;

if(hourCounts.length>0){

min=hourCounts[0];

}

for (int i = 0; i<hourCounts.length;i++){

if(hourCounts[i]<min){

min=hourCounts[i];

quietestHour = i;

}

}

return quietestHour;

}

}

# Exercise 4.75

Se kod för busiestHour ovan.

Då listan inte är sorterad så måste jag kolla varje element i listan. Jag valde att använda mig av en for loop eftersom jag ändå behövde ett index som indikerade vilken timme jag kollade på. Vid användning av en for-each loop hade jag ändå behövt en till variabel som ökade för varje iteration för att hålla koll på vilken timme jag var på, då listan bara innehöll antalet och inte vilken timme.

# Exercise 4.77

Se kod ovan.

Om 2 element skulle ha samma värde och det värdet skulle vara max så skulle min metod returnera den första timmen av dem 2 som detta uppstod. Eftersom jag kollar om nästkommande timmar är strikt större än den senast största jag stött på.

# Exercise (beräkna fakultet)

Beräkning av fakultet utan rekursion.

/\*\*

\* Computes n!.

\* Precondition: 0 <= n <= 20.

\* (20! < 2^63 - 1, the maximum value of a long.)

\*/

public long factorialFor(int n) {

long result=1;

for(int i=1; i<=n;i++){

result\*=i;

}

return result;

}

# Exercise (beräkna summa)

Beräkning av summan av alla tal i en array utan rekursion.

/\*\*

\* Computes the sum of the elements from v[first] to v[last].

\* Precondition: 0 <= first <= last < v.length.

\*/

public long sum(int[] v, int first, int last) {

long sum = 0;

for(int i =first;i<=last;i++){

sum+=v[i];

}

return sum;

}

# Exercise (maxvalue)

Gör två implementationer, en iterativ och en rekursiv, av en metod som returnerar det maximala värdet i en vektor. Metoden ska ha följande signatur:

public int max(int[] v)

## Iterativ:

/\*\*

\* @param v Enter an array of integers on which you would like to find the biggest value in.

\* @return Returns the maximum value in the array that is passed in to the method.

\*/

public int max(int[] v){

int result=0;

if(v.length>0){

result = v[0];

}

for (int i: v){

if(i>result){

result=i;

}

}

return result;

}

## Rekursiv

/\*\*

\* @param v Enter an array of integers on which you would like to find the biggest value in.

\* @return Returns the maximum value in the array that is passed in to the method.

\*/

public int max(int[] v){

int first = v[0];

int next = 0;

int[] u = new int[v.length-1];

if(v.length==1){

return first;

} else{

System.arraycopy(v, 1, u, 0, u.length);

next = max(u);

}

return Math.max(first, next);

}