CS553 Cloud Computing

Programming Assignment #3

Source Code

Pranitha Nagavelli(A20345406)

Client.java:

```
import java.awt.List;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.ArrayList;

import javax.swing.JOptionPane;

import Schedular.localQueue;
import Schedular.SQS;
import Worker.localWorker;


/**
 * Trivial client for the date server.
 */
public class client {
	static localQueue queue;
	static SQS sqs;
	static localWorker lwk;

	  public static void main(String[] args) throws Exception {
				String version=args[2];
				String wor=args[3];
				 queue=new localQueue();
		       String num=args[0];
				String filename=args[1];
			   int threads=0;
			   int w=0;

		    try{
		       threads=Integer.parseInt(num);
		       w=Integer.parseInt(wor);
		    }
		    catch(Exception e)
		    {
		       System.out.println("Task is runnig on Remote Worker....");
```

```
          }
          ObjectInputStream in;

          BufferedReader br = null;
                    try {
                              br = new BufferedReader(new FileReader(filename));
                    } catch (FileNotFoundException e) {
                              // TODO Auto-generated catch block
                              e.printStackTrace();
                    }
         String sl=" ";
         String task="";
         int i=0;
         int s=0;
         sl=br.readLine();

             try {
                    while(sl!=null)
                {
                     int t=Integer.parseInt(sl.split(" ")[1]);
if(version.equals("lw")){



          if(t==0){
             for(int a=0;a<100000;a++){
                    queue.insert(sl,i);
              i++;
              }
             System.out.println("client is done");
            task(t,"lw",threads);

       }
               else if(t==10){
                    for(int a=0;a<1000;a++){
                            queue.insert(sl, i);
                           i++;
                            }

                    System.out.println("client is done");
                    task(t,"lw",threads);




               }
       else if(t==100){
             for(int a=0;a<100;a++){
                     queue.insert(sl, i);
```

```
                                i++;
                                }
                        System.out.println("client is done");
                       task(t,"lw",threads);

                }
                else if(t==1000){
                        for(int a=0;a<100;a++){
                                queue.insert(sl, i);
                                i++;
                                }
                        System.out.println("client is done");
                       task(t,"lw",threads);

                }
                else if(t==10000){
                        for(int a=0;a<10;a++){
                                queue.insert(sl, i);
                                i++;
                                }
                        System.out.println("client is done");
                        task(t,"lw",threads);
                }
                else{

                       i=0;
                        queue.insert(sl, i);
                        i++;
                       //System.out.println(tasks.get(s));
                       task(t,"lw",threads);

                }



                  }

else if(version.equals("rw"))
{
sqs=new SimpleQueueServiceSample("RequestQueue");
System.out.println("add to sqs");
if(t==0){
        for(int a=0;a<(w*100000);a++){
            sqs.insert_sqs(sl);}}
        else if(t==10){
        for(int a=0;a<(w*1000);a++){
            sqs.insert_sqs(sl);}
        }
```

```
else if(t==100){
            for(int a=0;a<(w*100);a++){
                  sqs.insert_sqs(sl);}
            }
else if(t==1000){
            for(int a=0;a<(w*100);a++){
                        sqs.insert_sqs(sl);}
            }
else if(t==10000){
            for(int a=0;a<(w*10);a++){
                        sqs.insert_sqs(sl);}
            }
else{
            sqs.insert_sqs(sl);
            ;
}

            }
             else if(version.equals("animoto")){
                                    //sqs.insert(sl, "Queue");
                        }
          sl=br.readLine();
                        }

                     }
                                    catch (IOException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                    }}



             public static void task(int t,String v,int th) throws java.lang.NullPointerException{
                         if(v.equals("lw"))
        {int s;
        int s1=0;
        long start,end=0;
           s=queue.getsize();



             start=System.currentTimeMillis();
             lwk= new localWorker(queue, th);
                   s1=lwk.work();
                  end=System.currentTimeMillis();
                 //System.out.println("result "+s1);
                  System.out.println("Time taken "+(end-start));
        }
        else
        {
             System.out.println("Client has sent the messages to the SQS");
        }
             }

   }
```

LocalWorker.java:

```
package Worker;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import Scheduler.localQueue;

public class localWorker {

              thread f=new thread();
              localQueue input;
              int thread;
               int j,s;
              localQueue result;
              public localWorker(localQueue input,int th){
                           this.input=input;
                           thread=th;
                           result= new localQueue();

              }
              public int work(){


              ExecutorService executor = Executors.newFixedThreadPool(thread);
              j=input.getsize();
              System.out.println("Task is running on Local Worker:");
              int t=0;
              System.out.println(j);
                            for (int i = 0; i < j; i++) {

                                         String s=input.delete(i);

                                         int time = Integer.parseInt(s.split(" ")[1]);


                                         {
                                                      if(time==t){
```

```
                                        Runnable worker = new thread(s,i);
                                                executor.execute(worker);
                                                //System.out.println("Task sleep"+time+"is done");
                                //System.out.println(i);
                                 }
                                 else{

                                        t=time;
                                        Runnable worker = new thread(s,i);
                                                executor.execute(worker);
                                                //System.out.println(i);
                                 }
                        }

                 }

                executor.shutdown();
                while (!executor.isTerminated()) {
                }
                System.out.println("Finished all threads");

                result=f.getresult();
                result.showqueue();
                //System.out.println(result.getsize());
                return result.getsize();
              }


}
```

Thread.java:

```
package Worker;

import Scheduler.localQueue;

public class thread implements Runnable {
        public static  localQueue result ;
   private String command;
        int k;
        static int m=0;
   public thread()
   {}
   public thread(String s,int k){
      this.command=s;
      this.k=k;
      result = new localQueue();
   }
```

```
    @Override
    public void run() {
            m++;
        int time = Integer.parseInt(command.split(" ")[1]);
        processCommand(time);
        result.insert("Task#:"+time+"->END", k);

       //System.out.println(k);
        //System.out.println("Task #:"+key+" End.");
    }

    private void processCommand(int i) {
        try {
            Thread.sleep(i);
            System.out.println("Sleep"+i+",Thread:"+Thread.currentThread());
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public localQueue getresult()
    {
            return this.result;
    }


    @Override
    public String toString(){
        return this.command;
    }

}
```

LocalQueue.java:

```
package Scheduler;
import java.util.HashMap;
public class localQueue {
        public HashMap<Integer,String> Tasks=new HashMap<Integer,String>();

         public void insert(String s,int i)
         {
                 Tasks.put(i, s);

         }

         public int getsize()
```

```
              {
                return Tasks.size();
              }
   public void showqueue()
   {for (int name: this.Tasks.keySet()){

      System.out.print("sleep"+name+":");
      System.out.println(Tasks.get(name));
   }

   }
   public String delete(int key)
   {      String s=this.Tasks.get(key);

      this.Tasks.remove(key);
      return s;

   }


}
```

RemoteWorker.java:

```
package Worker;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import Queue.SQS;
public class remoteWorker {
              public static void main(String arg[]) {
                          SQS tasks=new SQS();
                           runningsqs a=new runningsqs();
                           String th=arg[0];
                           long start,end=0;
         int threads=0;
       try{
              threads=Integer.parseInt(th);
       }
       catch(Exception e)
       {
              System.out.println("Task is running on Remote Worker:");
       }

                           remoteThread f=new remoteThread();
                           boolean flag=true;
```

```
                        int j=tasks.sizeofsqs("Queue");

                        //a=tasks.get_first("Queue");

                        ExecutorService executor = Executors.newFixedThreadPool(threads);

                        start=System.currentTimeMillis();
                         while(j>0) {

                                Runnable worker = new remoteThread();
                      executor.execute(worker);
                      j--;

                       }
                         end=System.currentTimeMillis();

                    executor.shutdown();
                    while (!executor.isTerminated()) {
                    }
                    System.out.println("Finished all threads");
                    System.out.println("Time:"+(end-start));

                }

}
```

RemoteThread.java:

```
package Worker;

import Queue.SQS;

public class remoteThread implements Runnable{
        AmazonDynamoDB ad=new AmazonDynamoDB();
        runningsqs a=new runningsqs();
        SQS tasks=new SQS();
        SQS results=new SQS();

        private String command;
        int d;
       int key;
     static int m=0;

   @Override
   public void run() {
   try {
                  ad.init();
```

```
         } catch (Exception e2) {
                     // TODO Auto-generated catch block
                     e2.printStackTrace();
         }
  a=tasks.get("Queue");

  try {
          d=ad.duplicate(a.taskid);
         } catch (Exception e1) {
                     // TODO Auto-generated catch block
                     e1.printStackTrace();
         }
  if(d==1)
  { if(a.task.equals("0"))
    {
                       m++;

          System.out.println("stop.");
     }
      else{
      int time = Integer.parseInt(a.task.split(" ")[1]);
        processCommand(time);
      try {
                        results.insert(a.taskid+"is Finished", "result");
               }   catch (Exception e) {
                       // TODO Auto-generated catch block
                       e.printStackTrace();
              }
      m++;
    System.out.println("Task #:"+a.taskid+" End.");

    }
  }
else
  {if(a.taskid.equals("0"))
  {
            m++;

          System.out.println("stop.");
    }
   else{
          System.out.println("duplicate::"+a.taskid);}

  }


  }
```

```
    private void processCommand(int i) {
        try {
            Thread.sleep(i);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    @Override
    public String toString(){
        return this.command;
    }
}
```

SQS.java:

```
package Queue;

import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClient;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.DeleteMessageRequest;
import com.amazonaws.services.sqs.model.DeleteQueueRequest;
import com.amazonaws.services.sqs.model.GetQueueAttributesRequest;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.SendMessageRequest;

import Worker.runningsqs;

public class SQS {
```

```
            String myQueueUrl;
            AmazonSQS sqs;
            BasicAWSCredentials credentials = new BasicAWSCredentials("AKIAJ6AHA7V4SB5RXHBA",
"q6+aogB3NL22KgXGn7afIo1PapDSAmbetIpdUV6b");
            CreateQueueRequest createQueueRequest;
            public SQS()
            {

                        sqs = new AmazonSQSClient(credentials);
                createQueueRequest = new CreateQueueRequest("MyQueue");
       myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();
       Region usWest2 = Region.getRegion(Regions.US_WEST_2);
       sqs.setRegion(usWest2);

            }

    public void insert(String tasks,String name) throws Exception {

   try {
          CreateQueueRequest createQueueRequest = new CreateQueueRequest(name);
          myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();
          sqs.sendMessage(new SendMessageRequest(myQueueUrl, tasks));

       } catch (AmazonServiceException ase) {
          System.out.println("Caught an AmazonServiceException, which means your request made it " +
                "to Amazon SQS, but was rejected with an error response for some reason.");
            } catch (AmazonClientException ace) {
          System.out.println("Caught an AmazonClientException, which means the client encountered " +
                "a serious internal problem while trying to communicate with SQS, such as not " +
                "being able to access the network.");
          System.out.println("Error Message: " + ace.getMessage());
       }
    }

public runningsqs get(String name)
{
            runningsqs a=new runningsqs();
            System.out.print("Receiving:");

try {

} catch (Exception e) {}
              sqs = new AmazonSQSClient(credentials);
  Region usWest2 = Region.getRegion(Regions.US_WEST_2);
  sqs.setRegion(usWest2);
try {
    CreateQueueRequest createQueueRequest = new CreateQueueRequest(name);
    myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();
```

```
} catch (AmazonServiceException ase) {
   System.out.println("Caught an AmazonServiceException, which means your request made it " +
          "to Amazon SQS, but was rejected with an error response for some reason.");
     } catch (AmazonClientException ace) {
   System.out.println("Caught an AmazonClientException, which means the client encountered " +
          "a serious internal problem while trying to communicate with SQS, such as not " +
          "being able to access the network.");
   System.out.println("Error Message: " + ace.getMessage());
}
ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest(myQueueUrl);
List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();
for (Message message : messages) {
   System.out.println("  Message");

}

            if(messages.size()>0)
            {
                System.out.println("Deleting a message from SQS\n");
        CreateQueueRequest createQueueRequest = new CreateQueueRequest("MyQueue1");
        String messageRecieptHandle = messages.get(0).getReceiptHandle();
        sqs.deleteMessage(new DeleteMessageRequest(myQueueUrl, messageRecieptHandle));
            }
            else{
                        a.task="0";
                        a.taskid="0";
            }
            return a;
}

public void delete(String name)
{
              System.out.println("Receiving messages from Queue.\n");

              try {
             //
              } catch (Exception e) {}
                          sqs = new AmazonSQSClient(credentials);
                Region usWest2 = Region.getRegion(Regions.US_WEST_2);
                sqs.setRegion(usWest2);
              try {
                 CreateQueueRequest createQueueRequest = new CreateQueueRequest(name);
                 myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();


              } catch (AmazonServiceException ase) {
                System.out.println("Caught an AmazonServiceException, which means your request made it
" +
                        "to Amazon SQS, but was rejected with an error response for some reason.");
                  } catch (AmazonClientException ace) {
                System.out.println("Caught an AmazonClientException, which means the client encountered
" +
                        "a serious internal problem while trying to communicate with SQS, such as not " +
                        "being able to access the network.");
                System.out.println("Error Message: " + ace.getMessage());
            }

                    sqs.deleteQueue(new DeleteQueueRequest(myQueueUrl));

}
public int sizeofsqs(String name)
{
            try {

            } catch (Exception e) {}
                     sqs = new AmazonSQSClient(credentials);
              Region usWest2 = Region.getRegion(Regions.US_WEST_2);
              sqs.setRegion(usWest2);
            try {
                CreateQueueRequest createQueueRequest = new CreateQueueRequest(name);
                myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();


            } catch (AmazonServiceException ase) {
                System.out.println("Caught an AmazonServiceException, which means your request made it
" +
                        "to Amazon SQS, but was rejected with an error response for some reason.");
                  } catch (AmazonClientException ace) {
                System.out.println("Caught an AmazonClientException, which means the client encountered
" +
                        "a serious internal problem while trying to communicate with SQS, such as not " +
                        "being able to access the network.");
                System.out.println("Error Message: " + ace.getMessage());
            }
            GetQueueAttributesRequest request = new GetQueueAttributesRequest();
    request = request.withAttributeNames("ApproximateNoOfMessages");

     request = request.withQueueUrl(myQueueUrl);

       Map<String, String> map = sqs.getQueueAttributes(request).getAttributes();


       int size = Integer.parseInt(map.get("ApproximateNoOfMessages"));
```

```
        System.out.println("sizeOfMessages "+size);

            return size;
                        }
}
```

AmazonDynamoDB.java:

```
package Worker;
import java.util.HashMap;
import java.util.Map;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.ComparisonOperator;
import com.amazonaws.services.dynamodbv2.model.Condition;
import com.amazonaws.services.dynamodbv2.model.ConditionalCheckFailedException;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableRequest;
import com.amazonaws.services.dynamodbv2.model.ExpectedAttributeValue;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.PutItemRequest;
import com.amazonaws.services.dynamodbv2.model.PutItemResult;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.services.dynamodbv2.model.ScanRequest;
import com.amazonaws.services.dynamodbv2.model.ScanResult;
import com.amazonaws.services.dynamodbv2.model.TableDescription;
//import com.amazonaws.services.dynamodbv2.util.Tables;
import com.amazonaws.services.dynamodbv2.util.TableUtils;
import com.amazonaws.services.dynamodbv2.util.Tables;


@SuppressWarnings("deprecation")
public class AmazonDynamoDB {




            /*
    * Before running the code:
    *      Fill in your AWS access credentials in the provided credentials
```

```
     *      file template, and be sure to move the file to the default location
     *      (C:\\Users\\Prani\\.aws\\credentials) where the sample code will load the
     *      credentials from.
     *      https://console.aws.amazon.com/iam/home?#security_credential
     *
     * WARNING:
     *      To avoid accidental leakage of your credentials, DO NOT keep
     *      the credentials file in your source directory.
     */
    static AmazonDynamoDBClient dynamoDB;
    static String tableName = "Duplicate";



    /**
     * The only information needed to create a client are security credentials
     * consisting of the AWS Access Key ID and Secret Access Key. All other
     * configuration, such as the service endpoints, are performed
     * automatically. Client parameters, such as proxies, can be specified in an
     * optional ClientConfiguration object when constructing a client.
     *
     * @see com.amazonaws.auth.BasicAWSCredentials
     * @see com.amazonaws.auth.ProfilesConfigFile
     * @see com.amazonaws.ClientConfiguration
     */
    public static void init() throws Exception {
        /*
         * The ProfileCredentialsProvider will return your [default]
         * credential profile by reading from the credentials file located at
         * (C:\\Users\\Prani\\.aws\\credentials).
         */
       // AWSCredentials credentials = null;
            BasicAWSCredentials credentials = new BasicAWSCredentials("AKIAJ6AHA7V4SB5RXHBA",
"q6+aogB3NL22KgXGn7afIo1PapDSAmbetIpdUV6b");
        try {
            //credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\Prani\\.aws\\credentials), and is in valid format.",
                    e);
        }
        dynamoDB = new AmazonDynamoDBClient(credentials);
        Region usWest2 = Region.getRegion(Regions.US_WEST_2);
        dynamoDB.setRegion(usWest2);
        // Create table if it does not exist yet
        try{
            if (Tables.doesTableExist(dynamoDB, tableName)) {
          System.out.println("Table: " + tableName + " is already ACTIVE");
        } else {
          // Create a table with a primary hash key named 'name', which holds a string
          CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
              .withKeySchema(new
KeySchemaElement().withAttributeName("name").withKeyType(KeyType.HASH))
              .withAttributeDefinitions(new
AttributeDefinition().withAttributeName("name").withAttributeType(ScalarAttributeType.S))
              .withProvisionedThroughput(new
ProvisionedThroughput().withReadCapacityUnits(2L).withWriteCapacityUnits(2L));
              TableDescription createdTableDescription =
dynamoDB.createTable(createTableRequest).getTableDescription();
          System.out.println("Created Table: " + createdTableDescription);

          // Wait for it to become active
          System.out.println("Waiting for " + tableName + " to become ACTIVE...");
          Tables.waitForTableToBecomeActive(dynamoDB, tableName);

        }}catch(Exception e){
          System.out.println("Waiting for " + tableName + " to become ACTIVE...");
          Tables.waitForTableToBecomeActive(dynamoDB, tableName);
        }

       // Describe our new table
       DescribeTableRequest describeTableRequest = new
DescribeTableRequest().withTableName(tableName);
       TableDescription tableDescription = dynamoDB.describeTable(describeTableRequest).getTable();
       //System.out.println("Table Description: " + tableDescription);
    }


            public int duplicate(String taskid) throws Exception {



       try {

            System.out.println(" amazondb "+ taskid);
          // Add an item
         /* Map<String, AttributeValue> item = newItem("Bill & Ted's Excellent Adventure", 1989, "****",
"James", "Sara");
          PutItemRequest putItemRequest = new PutItemRequest(tableName, item);
          PutItemResult putItemResult = dynamoDB.putItem(putItemRequest);
          System.out.println("Result: " + putItemResult);*/

          // Add another item
          Map<String, AttributeValue> item = newItem(taskid);
```

```
            ExpectedAttributeValue notExpected = new ExpectedAttributeValue(false);
            Map<String, ExpectedAttributeValue> expected = new HashMap<String,
ExpectedAttributeValue>();
            expected.put("name", notExpected);
            PutItemRequest putItemRequest = new
PutItemRequest().withTableName(tableName).withItem(item).withExpected(expected);
            try{ PutItemResult putItemResult = dynamoDB.putItem(putItemRequest);
            }catch( ConditionalCheckFailedException e)
            {


               return 0;
            }

        } catch (AmazonServiceException ase) {

        } catch (AmazonClientException ace) {

        }
        return 1;
    }

    private static Map<String, AttributeValue> newItem(String name) {
              Map<String, AttributeValue> item = new HashMap<String, AttributeValue>();
         item.put("name", new AttributeValue(name));
 return item;
    }

}
```

Animoto.java:

```
package Worker;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import com.amazonaws.HttpMethod;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.ClasspathPropertiesFileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.S3Object;
public class animoto {
        public static void main(String args[])
        {

                int thread = Integer.parseInt(args[0]);

                boolean flag=true;

                                ExecutorService executor = Executors.newFixedThreadPool(thread);
                                int j=0;
                                while(flag==true)
                                {
                                Runnable worker = new animotoThread();
                                 j++;
                      executor.execute(worker);
                      if(j>=1)
                                        flag=false;
                                }
                      executor.shutdown();


        }

}
```

AnimotoThread.java

```
package Worker;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import com.amazonaws.HttpMethod;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.ClasspathPropertiesFileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.S3Object;

import Scheduler.SQS;
public class animotoThread implements Runnable{
        static int threads=0;
          String cmd;
        int d=0;
        SQS tasks=new SQS();;
        SQS  results=new SQS();
        AmazonDynamoDB dp=new AmazonDynamoDB();
        runningsqs a=new runningsqs();
static BasicAWSCredentials credentials = new BasicAWSCredentials("AKIAJ6AHA7V4SB5RXHBA",
"q6+aogB3NL22KgXGn7afIo1PapDSAmbetIpdUV6b");
static AmazonS3 s3client = new AmazonS3Client(credentials);
private static String bucket;
 int img=0;
        @Override
   public void run() {
        try {
                a=tasks.get("Queue");
      bucket="PA3";
      s3client.createBucket(bucket);
                 d=dp.duplicate(a.taskid);
           if(d==1)
           {
       String url[]=a.task.split("@");
           System.out.println("url:"+url[1]);
           for(int i=1;i<url.length;i++)
                 {
             URL urlno = new URL(url[i]);
             String commandString="wget "+url[i];
             Process process = Runtime.getRuntime().exec(commandString);
                                 }
           img++;}
```

```
                      cmd = "C:\\Users\\Prani\\Downloads\\ffmpeg-20160425-git-9ac154d-win32-
static\\ffmpeg-20160425-git-9ac154d-win32-static\\bin\\ffmpeg -framerate 1/5 -i "
                              + "C:\\Users\\Prani\\workspace\\PA#3\\src\\image" +img+ "\\img%2d.jpg -
pix_fmt yuv420p -r 25 "  + "F:\\workspacejava\\animoto\\src\\img" + "\\video.mpeg";



             Process p =Runtime.getRuntime().exec(cmd);

                        File f=new File("C:\\Users\\Prani\\workspace\\PA#3\\src\\image\\video.mpeg");
                                          S3Object s3Object = new S3Object();

                    InputStream in = new FileInputStream(f);

                    ObjectMetadata omd = new ObjectMetadata();

                    omd.setContentLength(f.length());
            s3client.putObject(new PutObjectRequest(bucket, "1", in,omd));

                   S3Object obj = s3client.getObject(new GetObjectRequest(bucket, "1"));
                   GeneratePresignedUrlRequest generatePresignedUrlRequest =
                                                       new GeneratePresignedUrlRequest(bucket, "1");
                                         generatePresignedUrlRequest.setMethod(HttpMethod.GET);

                                         URL url = s3client.generatePresignedUrl(generatePresignedUrlRequest);


                                  try {

                                         System.out.println("Url is created"+url.toString());
                               } catch (Exception e) {
                                         // TODO Auto-generated catch block
                                         e.printStackTrace();
                               }

              }
           } catch (Exception e) {
                     e.printStackTrace();
           }

   }
```