CS 102: Data Structures

Project Five - The Bank of Nyan

Student Name	Mohammad El-Abid
Student ID	8905652
Professor	Subrina Thompson
Date	Tuesday, April 17, 2012

Table of Contents

Abstract

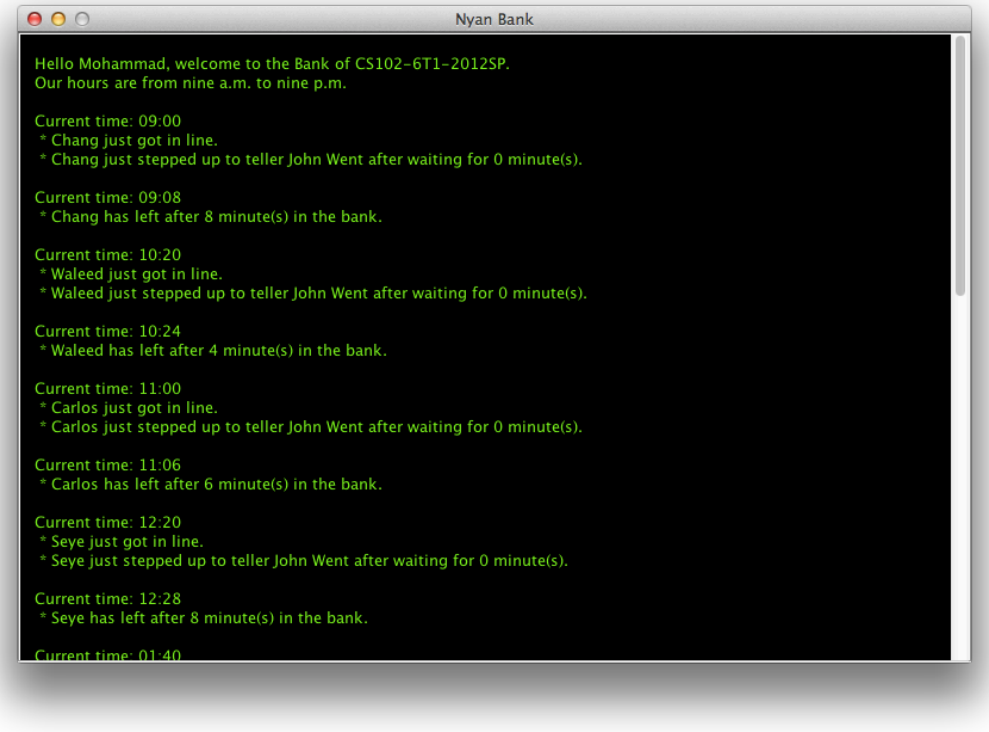
A Java application using custom implementations of a `Queue`. This queue is then used to simulate a bank with a teller processing the customers as they enter the bank and are processed by the teller.

Introduction

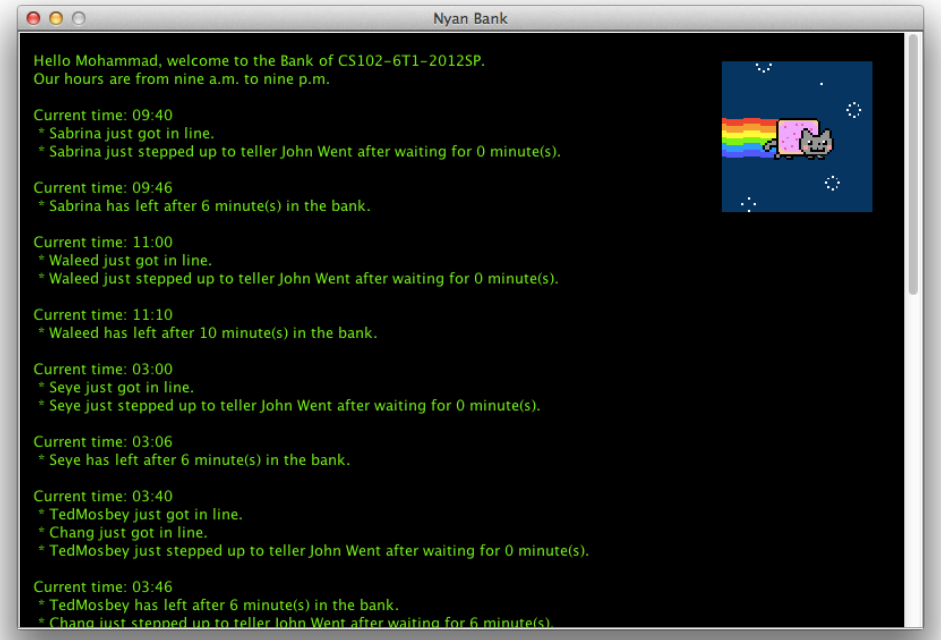
The application loads names, arrival time, and an estimated length of time to serve the customer. The customers are pushed to a queue to wait until their arrival time. Once they “arrive” they are added to a queue/lineup inside the bank while waiting for a teller. After completing the task rather quickly, I made it only load the names from the text file and create random arrival and service lengths. I also added some Nyan Cat to it.

Screenshots

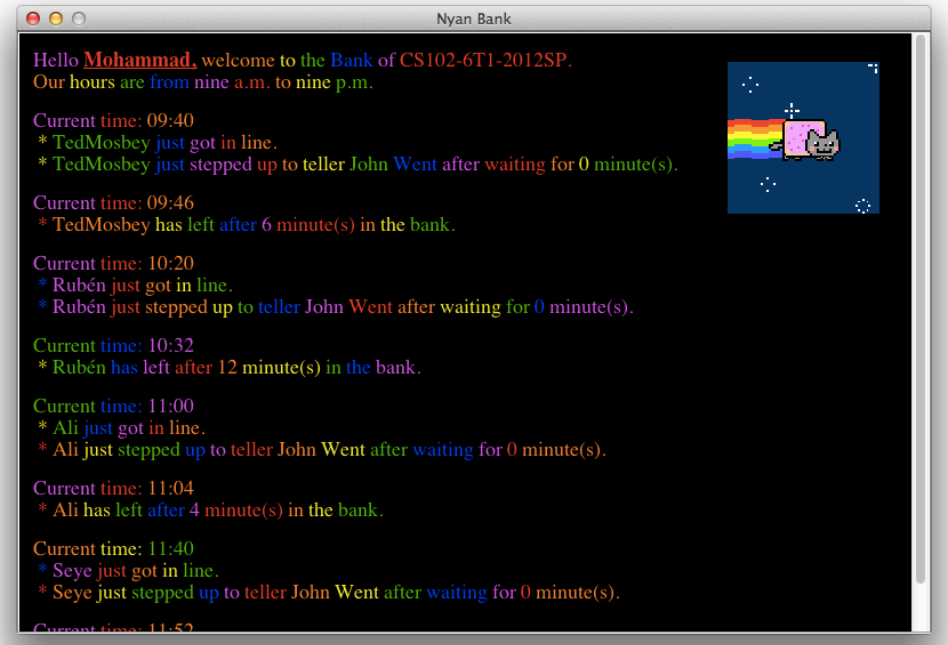
(proper version)



(fun version)



(more fun version)



Code

File: Application.java

```
package edu.bridgeport.mohammad.bank;

import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

import edu.bridgeport.mohammad.Queue;

public class Application extends javax.swing.JFrame {
  private javax.swing.JTextArea display;

  public Application() {
    setTitle("Bank of CS102-6T1-2012SP");
    setSize(800, 550);
    setDefaultCloseOperation(EXIT_ON_CLOSE);

    display = new javax.swing.JTextArea();
    display.setBackground(java.awt.Color.black);
        display.setForeground(new java.awt.Color(40, 254, 20));
        display.setEditable(false);
        add(new javax.swing.JScrollPane(display));

        setVisible(true);
        requestFocus(true);
  }

  public javax.swing.JTextArea getDisplay() {
    return display;
  }

  public void display(String text) {
    display.append("   " + text + "\n");
  }

  public static void main(String[] args) {
    Application gui = new Application();
    gui.display("");

    Queue<Customer> outSide = new Queue<Customer>();
    Queue<Customer> inSide = new Queue<Customer>();
    Teller[] tellers = {new Teller("John Went")}; // http://imgur.com/
gallery/iWaY1
    int time = 0; // minutes past nine
```

```
    // Be creepy
    String uname = System.getProperty("user.name");
    char cap = Character.toUpperCase(uname.charAt(0));
    uname = String.valueOf(cap) + uname.substring(1);
    gui.display("Hello " + uname +", welcome to the Bank of
CS102-6T1-2012SP.");
    gui.display("Our hours are from nine a.m. to nine p.m.");
    gui.display("");

    // Metric variables
    int customersServed = 0;
    int averageWaitTime = 0;
    int currentLineLength = 0;
    int maxLineLength = 0;
    int maxLineLengthTime = 0;

    // load file to seed outSide
    Scanner input = new
Scanner(Application.class.getResourceAsStream("/customers.txt"));
    ArrayList<Customer> customers = new ArrayList<Customer>();
    while(input.hasNextLine()) {
      /*
      Customer temp = new Customer(input.next(), input.nextInt(),
input.nextInt());
      if(input.hasNextLine()) input.nextLine(); // move to end of line
      // works because the text file list customers in order,
otherwise we would need to sort them
      outSide.enqueue(temp);
      */
      int arrivalTime = (int) (System.nanoTime() % (12 * 60)); //
twelve hours max 9am-9pm
      int serviceTime = (int) (System.nanoTime() % 14); // max
thirteen minutes
      Customer temp = new Customer(input.next(), arrivalTime,
serviceTime);
      customers.add(temp);
      if(input.hasNextLine()) input.nextLine(); // move to end of line
    }
    Collections.sort(customers); // order in time of arrival since it
was randomly generated
    for(Customer customer : customers) outSide.enqueue(customer);

    // tick
    ArrayList<String> actions = new ArrayList<String>();
    boolean done = false;

    while(!done) {
      actions.clear();
```

```
      while(outSide.look() != null && outSide.look().getArrivalTime()
<= time) {
        Customer walkingIn = outSide.dequeue();
        actions.add(walkingIn.getName() + " just got in line.");
        inSide.enqueue(walkingIn);
        currentLineLength++;
      }

      for(Teller teller : tellers) {
        if(teller.getHelping() != null) {
          int doneAt = teller.getStartedHelping() +
teller.getHelping().getServiceLength();
          if(time >= doneAt) {
            int minutes = (time -
teller.getHelping().getArrivalTime());
            actions.add(teller.getHelping().getName() + " has left
after " + minutes + " minute(s) in the bank.");
            customersServed++;
            teller.setHelping(null, 0);
          }
        }

        if(teller.isAvailable() && !inSide.empty()) {
          Customer next = inSide.dequeue();
          int wait = (time - next.getArrivalTime());
          averageWaitTime += wait; // gets divided later
          actions.add(next.getName() + " just stepped up to teller " +
teller.getName() + " after waiting for " + wait + " minute(s).");
          currentLineLength--;
          teller.setHelping(next, time);
        }
      }

      // max line length
      if(currentLineLength > maxLineLength) {
        maxLineLength = currentLineLength;
        maxLineLengthTime = time;
      }
      // e/o max line length

      // check done
      done = true;
      if(outSide.empty() && inSide.empty()) {
        for(Teller teller : tellers) {
          if(!teller.isAvailable()) {
            done = false;
          }
        }
```

```
    } else {
      done = false;
    }
    // e/o check done

    if(actions.size() > 0) {
      int hour = 9;
      int minutes = time;

      if(minutes >= 60) {
        hour += minutes/60;
        minutes = minutes % 60;
      }
      if(hour > 12) hour = hour % 12;

      gui.display("Current time: " + timeStamp(time));

      for(int i = 0; actions.size() > i; i++) {
        gui.display(" * " + actions.get(i));
      }
      gui.display("");

      try {
        Thread.sleep(3000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }

    time++;
  }
  // e/o tick


  // finish metrics
  try {
    averageWaitTime = averageWaitTime / customersServed;
  } catch(java.lang.ArithmeticException ex) {
    // division by zero, zero customers
    averageWaitTime = 0;
  }

  int hour = 9;
  int minutes = maxLineLengthTime;

  if(minutes >= 60) {
    hour += minutes/60;
    minutes = minutes % 60;
  }
```

```
    if(hour > 12) hour = hour % 12;

    // display metrics
    gui.display("Customer(s) served: " + customersServed);
    gui.display("Average wait time was about " + averageWaitTime + "
minute(s).");
    gui.display("Busiest time was " + timeStamp(maxLineLengthTime) + "
when there was " + maxLineLength + " customers in the line.");

    if(maxLineLength > 0) {
      gui.display("This queue could have been served faster with a
second teller.");
    } else {
      gui.display("This queue only needed one teller.");
    }

    gui.display("There were zero robberies during this time
interval.");
  }

  public static String timeStamp(int time) {
    NumberFormat formatter = new DecimalFormat("00");

    StringBuilder build = new StringBuilder();
    int hour = 9;
    int minutes = time;

    if(minutes >= 60) {
      hour += minutes/60;
      minutes = minutes % 60;
    }
    if(hour > 12) hour = hour % 12;

    build.append(formatter.format(hour) + ":" +
formatter.format(minutes));
    return build.toString();
  }

}
```

File: Customer.java

```
package edu.bridgeport.mohammad.bank;

public class Customer implements Comparable<Customer> {
  private String name;
  private int arrivalTime, serviceLength;

  public Customer(String name, int arrivalTime, int serviceLength) {
    this.setName(name);
```

```
    this.setArrivalTime(arrivalTime);
    this.setServiceLength(serviceLength);
  }

  public int getArrivalTime() {
    return arrivalTime;
  }

  public void setArrivalTime(int arrivalTime) {
    this.arrivalTime = arrivalTime;
  }

  public int getServiceLength() {
    return serviceLength;
  }

  public void setServiceLength(int serviceLength) {
    this.serviceLength = serviceLength;
  }

  @Override
  public String toString() { return getName(); }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int compareTo(Customer c2) {
    if(arrivalTime > c2.getArrivalTime()) {
      return 1;
    } else if(arrivalTime == c2.getArrivalTime()) {
      return 0;
    } else {
      return -1;
    }
  }
}
```

File: Teller.java

```
package edu.bridgeport.mohammad.bank;

public class Teller {
  Customer helping = null;
  int startedHelping = 0;
  String name;
```

```
  public Teller(String name) {
    this.name = name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getName() {
    return this.name;
  }

  public void setHelping(Customer helping, int time) {
    this.helping = helping;
    this.startedHelping = time;
  }

  public Customer getHelping() {
    return this.helping;
  }

  public int getStartedHelping() {
    return this.startedHelping;
  }

  public boolean isAvailable() {
    return helping == null;
  }
}
```

File: Queue.java

```
package edu.bridgeport.mohammad;

public class Queue<T> {
  private Node<T> front;
  private Node<T> rear;

  public boolean enqueue(T item){
    Node<T> node = new Node<T>(item);

    if(front == null) {
      front = rear = node;
    } else {
      rear.setNext(node);
      rear = node;
    }
```

```
    return true;
  }

  public T look() {
    if(front != null) {
      return front.getData();
    } else {
      return null;
    }
  }

  public T dequeue() {
    if(front == null) return null;
    Node<T> current = front;
    this.front = current.getNext();
    return current.getData();
  }

  public boolean empty() {
    return front == null;
  }

  public String toString() {
  if(empty()) { return "Queue: [empty]"; }

    StringBuilder output = new StringBuilder();
    output.append("Queue: ");

    Node<T> current = front;
    while(current != null) {
      output.append(current.getData() + ", ");
      current = current.getNext();
    }

    output.delete(output.length() - 2, output.length());
    return output.toString();
  }
}
```

File: Node.java

```
package edu.bridgeport.mohammad;

public class Node<T> {
    private T data;
    private Node<T> next;

    public Node() {}

    public Node(T data){
```

```
        this.data = data;
    }

    public T getData() {
        return data;
    }

    public void setNext(Node<T> next){
        this.next = next;
    }

    public Node<T> getNext() {
      return next;
    }
}
```

Conclusion

Once I was done with the base requirements, I was challenged by a peer to add Nyan Cat to the application. It did not take too much time to hack a thread to draw the Nyan Cat, so I added the sound as well. I did this by using an external dependency, JavaZoom. I also compiled the application into a jar and converted the jar into a Mac application.