Queues II:

Applications
Palindrome recognition

• Palindrome: collection of characters that reads the same backwards and forwards

• Examples:
  – “otto” is a palindrome
  – “34543” is a palindrome
  – “snub no man; nice cinnamon buns!” is a palindrome, if you ignore spaces and punctuation
Palindrome recognition program

- Read string, storing each character in both a stack and a queue
- Once string is read, pop stack and dequeue queue, comparing characters
- if all characters have matched when both structures are empty, you’ve got a palindrome
import java.util.LinkedList; // implements Queue interface
import java.util.Queue;
import java.util.Stack;
import java.util.Scanner;

public class Palindrome {


Palindrome recognition program

public static void main(String[] args) {
    Scanner stdin = new Scanner(System.in);
    String line;
    do {
        System.out.print("Your expression (or return to end): ");
        line = stdin.nextLine();
        if (is_palindrome(line))
            System.out.println("That is a palindrome.");
        else
            System.out.println("That is not a palindrome.");
    } while (line.length() != 0);
}
public static boolean is_palindrome(String input) {
    Queue<Character> q = new LinkedList<Character>(
    Stack<Character> s = new Stack<Character>(
    Character letter;
    int mismatches = 0;

    int i;
    for (i = 0; i < input.length(); i++) {
        letter = input.charAt(i);
        if (Character.isLetter(letter)) {
            q.add(letter);
            s.push(letter);
        }
    }
}
Palindrome recognition program

```java
while (!q.isEmpty()) {
    if (q.remove() != s.pop())
        return false;
}
return true;
```

} // end of method

} // end of class

```
Simulations – a brief intro

• The code on the next couple of slides gives a very simple example of a class that simulates a real-world situation
• In this case, the situation is simple one: simulate the action of a traffic light
• It would be a lot more interesting if it was graphical, but then you would stay awake, and who needs that?
TrafficLight simulator

• Inputs:
  – amount of time light should be green or red
  – total time to simulate

• Outputs: current state of light
public class TrafficLight {
    private int redSpan; // Seconds that the light stays red
    private int greenSpan; // Seconds that the light stays green
    private boolean nowRed; // True if the light is now red
    private int secondsUntilChange; // Seconds until the next color change

    // member variables represent amount of time light stays green or red, current color of light, and time left in current color
    // constructor sets values for red & green time, starts red light timer
public TrafficLight(int r, int g) {
    if (r <= 0)
        throw new IllegalArgumentException("r <= 0: " + r);
    if (g <= 0)
        throw new IllegalArgumentException("g <= 0: " + g);
    redSpan = r;
    greenSpan = g;
    nowRed = true;
    secondsUntilChange = r;
}
TrafficLight simulator – accessor methods

```java
public boolean isRed() {
    return nowRed;
}

public boolean isGreen() {
    return !nowRed;
}
```
public void simulateTime(int t) {
    while (t >= secondsUntilChange) {
        t -= secondsUntilChange;
        nowRed = !nowRed;
        secondsUntilChange = nowRed ? redSpan : greenSpan;
    }
    secondsUntilChange -= t;
}
TrafficLight simulator – test driver

public static void main (String [] args) {
    Scanner kb = new Scanner(System.in);
    int simTime, r, g;
    System.out.print("Enter length of simulation (in seconds): ");
    simTime = kb.nextInt();
    System.out.print("Enter number of seconds light stays red: ");
    r = kb.nextInt();
    System.out.print("Enter number of seconds light stays green: ");
    g = kb.nextInt();
}
TrafficLight simulator – test driver continued

TrafficLight tl = new TrafficLight(r,g);
    for (int x = 0; x < simTime; x++)
    {
        tl.simulateTime(x);
        if (tl.isGreen())
            System.out.println("Green");
        else
            System.out.println("Red");
    }
Using queues for simulation

• Many real-world situations can be simulated by employing a queue or queues to represent clients waiting for service

• “Clients” might be cars at a traffic light, customers at a grocery store, jobs awaiting CPU time in a computer, etc.

• Clients are represented by number indicating time of entry into the queue
Queue simulation program

• Inputs:
  – amount of time needed to serve one client
  – probability of customer arriving at any given time
  – length of simulation

• Outputs:
  – number of clients served during simulation
  – average time spent waiting in line
Queue simulation program: required data structures

• Queue of clients -- represented by time stamp (second when user entered queue)
• Server -- an object that simulates the service to be performed -- includes:
  – start( ) method
  – reduceRemainingTime( ) method
  – isBusy( ) method
  – variables secondsForService, timeLeft
  – constructor that establishes variable values
Queue simulation program: required data structures

• ClientGenerator: reports whether or not a client has arrived in a given second, using probability input to determine odds of arrival

• Averager: tracks total number of customers served and average time spent waiting in queue
Simulation: main method

```java
import java.util.*;
public class CarWash {
    public static void main(String[] args) {
        final int WASHTIME = 240;
        final double ARRIVALPROB = 0.0025;
        final int TOTALTIME = 6000;

        carWashSimulate(WASHTIME, ARRIVALPROB, TOTALTIME);
    }
}
```
Simulation: carWashSimulate()

public static void carWashSimulate (int washTime, double arrivalProb, int totalTime) {
    Queue<Integer> arrivalTimes = new LinkedList<Integer>( );
    int next;
    ClientGenerator arrival = new ClientGenerator(arrivalProb);
    Server machine = new Server(washTime);
    Averager waitTimes = new Averager( );
    int currentSecond;
    System.out.println("Seconds to wash one car: " + washTime);
    System.out.println("Probability of customer arrival during a second: ");
    System.out.println(arrivalProb);
    System.out.println("Total simulation seconds: " + totalTime);
if (washTime <= 0 || arrivalProb < 0 || arrivalProb > 1 || totalTime < 0)
    throw new IllegalArgumentException("Values out of range");
for (currentSecond = 0; currentSecond < totalTime; currentSecond++) {
    if (arrival.query())
        arrivalTimes.add(currentSecond);
    if (!machine.isBusy() && !arrivalTimes.isEmpty()) {
        next = arrivalTimes.remove();
        waitTimes.addNumber(currentSecond - next);
        machine.start();
    } // end if
} // end for
// Subtract one second from the time in the current wash cycle.
    machine.reduceRemainingTime();
}  // end of for loop

// Write the summary information about the simulation.
    System.out.println("Customers served: " +
    waitTimes.howManyNumbers());
    if (waitTimes.howManyNumbers() > 0)
        System.out.println("Average wait: " +
        waitTimes.average() + " sec");
}  // end method
}  // end class
public class Server {
    private int secondsForService;
    private int timeLeft;

    public Server(int s) {
        secondsForService = s;
        timeLeft = 0;
    }
}
public boolean isBusy( ) {
    return (timeLeft > 0);
}

public void reduceRemainingTime( ) {
    if (timeLeft > 0)
        timeLeft--;
}
public void start( ) {
    if (timeLeft > 0) {
        throw new IllegalStateException
            ("Server is already busy.");
    
    timeLeft = secondsForService;
}
} // end class
public class ClientGenerator  {
    private double probability;

    public ClientGenerator(double p) {
        if (((p < 0) || (1 < p))
            throw new IllegalArgumentException
                ("Illegal p: "+ p);
        probability = p;
    }
}
public boolean query() {
    return (Math.random() < probability);
}
}  // end class
public class Averager {
    private int count;
    private double sum;

    public Averager() {
        count = 0;
        sum = 0;
    }
}
Averager class

public void addNumber(double value) {
    if (count == Integer.MAX_VALUE)
        throw new IllegalStateException
            ("Too many numbers");
    count++;
    sum += value;
}

queues2  30
Averager class

public double average( ) {
    if (count == 0)
        return Double.NaN;
    else
        return sum/count;
}

public int howManyNumbers( ) {
    return count;
}
} // end class