```java
import java.util.*;
/**********************************************
 * Manages MarketPlace operation.
 *
 * @author Shane Stacy
 * @version 12/6/2017
 *********************************************/
public class MarketPlace
{
    /**  the average arrival time    */
    private double avgArrivalTime = 0.0;

    /**  the average service time    */
    private double avgServiceTime = 0.0;

    /**  the number of cashiers      */
    private int numCashiers;

    /**  is the checkout area visible?  */
    private boolean checkoutAreaVisible = false;

    /**  the current time        */
    private double currentTime = 0.0;

    /**  the total wait time      */
    private double totalWaitTime = 0.0;

    /**  the average wait time    */
    private double avgWaitTime = 0.0;

    /**  the number of customers    */
    private int numCustomers = 0;

    /**  the length of the longest line   */
    private int longestLine = 0;


    String results = "";
    ArrayList<Customer> theLine;
    Customer[] theCashiers;
    PriorityQueue<GVevent> theEvents;
    GVrandom rand;
    GVdeparture nextEvent;

    /**                    */
    private final int openTime = 600;

    /**                    */
    private final int closeTime = 1080;

    /**************************************************
     * Creates the MarketPlace
     *************************************************/
    public MarketPlace() {
        avgArrivalTime = 2.5;
        avgServiceTime = 6.6;
        numCashiers = 3;
        avgWaitTime = 0.0;
        checkoutAreaVisible = false;
        currentTime = openTime;
        rand = new GVrandom();
        theCashiers = new Customer[numCashiers];

        results = "";

        theLine = new ArrayList<Customer>();
        theEvents = new PriorityQueue<GVevent>();

    }

    /***************************************************
     * Gets the number of cashiers in the store
     * @return int numCashiers the number of cashiers
     **************************************************/
    public int getNumCashiers() {
        return numCashiers;
    }

    /***************************************************
     * Gets the arrival time
     * @return double avgArrivalTime the average arrival time
     **************************************************/
    public double getArrivalTime() {
        return avgArrivalTime;
    }

    /***************************************************
     * Gets the service time
     * @return double avgServiceTime the average service time
     **************************************************/
    public double getServiceTime() {
        return avgServiceTime;
    }

    /***************************************************
     * The number of customers served
     * @return int the list size
     **************************************************/
    public int getNumCustomersServed() {
        return numCustomers;
    }

    /***************************************************
     * Gets the store report
     * @return results the results of the report
     **************************************************/
    public String getReport() {
        return results;
    }

    /***************************************************
     * Gets the length of the longest line
     * @return the length of the longest line
     **************************************************/
    public int getLongestLineLength() {
        return longestLine;
    }

    /**************************************************
     * Gets the average wait time
     * @return avgWaitTime the average wait time
     **************************************************/
    public  double getAverageWaitTime() {
        double avgWaitTime = totalWaitTime / numCustomers;
        return avgWaitTime;

    }

    /**************************************************
     * Initialize some variables
     * @param int num the number of cashiers
     * @param double s the average service time
     * @param double a the average arrival time
     * @param boolean ck is the checkout area visible
     **************************************************/
    public void setParameters(int num, double s, double a, boolean ck) {

        numCashiers = num;
        avgServiceTime = s;
        avgArrivalTime = a;
        checkoutAreaVisible = ck;
    }

    /*
     * End of Part 1
     * Start of Part 2
     *
     */
    /**************************************************
     * A customer gets in line
     **************************************************/
    public void customerGetsInLine() {
        Customer person = new Customer(currentTime);
        theLine.add(person);
        // if the line reaches a new daily high set longestLine equal to it
        if (theLine.size() > longestLine) {
            longestLine = theLine.size();
        }
        int i = cashierAvailable();
        // if there is a cashier available, move the next customer to it
        if (i != -1 && theLine.size() > 0) {
            customerToCashier(cashierAvailable());
        }
        //  keep customers coming in if the store is open
        if (currentTime < closeTime) {
            double futureTime = randomFutureTime(avgArrivalTime);
            GVarrival arrive = new GVarrival(this, futureTime);
            theEvents.add(arrive);
        }


    }

    /**************************************************
     * A customer pays
     * @param int num the cashier number
     **************************************************/
    public void customerPays (int num) {
        // if someone is in line move them to the next available cashier
        if (theLine.size() > 0) {
        customerToCashier(num);
    }
    else {
        theCashiers[num] = null;  // otherwise the cashier is idle
    }


    }

    /**************************************************
     * Reset the variables
     **************************************************/
    public void reset() {
        currentTime = 600;
        totalWaitTime = 0.0;
        avgWaitTime = 0.0;
        numCustomers = 0;
        theLine = new ArrayList<Customer>();
        theEvents = new PriorityQueue<GVevent>();

    }

    /*******************************************************
     * Return the index of the first available cashier
     * @return int the index of the first available cashier
     *******************************************************/
    private int cashierAvailable() {
        int emptyCashier = 0;
        //look for an open cashier
        for(int i = 0; i < theCashiers.length; i++) {
            if (theCashiers[i] == null) {
                emptyCashier = i;
                return emptyCashier;
            }

        }
        return -1;

    }

    /**************************************************
     * Returns a future time
     * @param double avg random number generated by GVrandomnextPosition
     * @return double the future time
     ***************************************************/
    private double randomFutureTime (double avg) {
        double futureTime = 0.0;

        futureTime = currentTime + rand.nextPoisson(avg);
        return futureTime;
    }

    /****************************************************
     * Moves customer at front of line to available cashier
     * @param int num the next available cashier
     ***************************************************/
    private void customerToCashier (int num) {
        double futureTime = 0.0;
        Customer c = theLine.remove(0);
        theCashiers[num] = c;
        numCustomers++;
        totalWaitTime = totalWaitTime + (currentTime - c.getArrivalTime());
        futureTime = randomFutureTime(avgServiceTime);
        nextEvent = new GVdeparture(this, futureTime, num);
        theEvents.add(nextEvent);

    }

    /****************************************************
     * Primary method to control simulation from beginning to end
     ***************************************************/
    public void startSimulation() {
        reset();
        GVarrival first = new GVarrival(this, currentTime);
        theEvents.add(first);
        // run the store until nothing is left to do
        while(!theEvents.isEmpty()) {
            GVevent e = theEvents.poll();
            currentTime = e.getTime();
            e.process();
            if (checkoutAreaVisible == true) {
            showCheckoutArea();
        }
        System.out.println(currentTime);
        }

        createReport();

    }

    /****************************************************
     * Creates the customer
     * @param double time the customer's arrival time
     ***************************************************/
    private void showCheckoutArea() {
        String cashierString = "";
        String customerString = "";
        //look for open cashiers and put a C
        for (int i = 0; i < theCashiers.length; i++) {
            if (theCashiers[i] == null) {
                cashierString = cashierString + "C";
            }
            else {
                cashierString = cashierString + "-";  // otherwise put a -
            }

        }
        //  put a * to represent the line
        for (int b = 0; b < theLine.size(); b++) {
            customerString = customerString + "*";
        }
        results = results + formatTime(currentTime) + " " + cashierString + " " + customerString + "\n";

    }

    /****************************************************
     * Creates the report
     ***************************************************/
    private void createReport() {
        avgWaitTime = totalWaitTime / numCustomers;
        results += "Simulation Parameters";
        results += "\n Number of cashiers: " + numCashiers;
        results += "\n Average arrival: " + avgArrivalTime;
        results += "\n Average service: " + avgServiceTime;
        results += "\n \n Results";
        results += "\n" + "Average wait time: " + getAverageWaitTime() + " mins";
        results += "\n" + "Max Line Length: " + getLongestLineLength() + " at " + formatTime(currentTime);
        results += "\n" + "Customers served: " + getNumCustomersServed();
        results += "\n" + "Last departure: " + formatTime(currentTime);

    }

    /****************************************************
     * Formats a time given the time in minutes
     * @param double mins time in minutes
     * @return String a time within a String
     ***************************************************/
    public String formatTime (double mins) {
        String formattedTime = "";
        String aMPM = "";
        String minutesString;
        String hoursString;
        int hours = (int) Math.round(mins / 60);
        int minutes = (int) Math.round(mins % 60);

        // if time is less than 60 minutes the hour is 12
        if (mins < 60) {
        hours = 12;

        }
        //  if time is greater than or qual to 720 minutes the time must be pm
        if (mins >= 720) {
         aMPM = "pm";

        }
        else {
            aMPM = "am";  //  otherwise it's am

        }
        // if time is greater or equal to 780 minutes, the hour starts again from 1
        if (mins >= 780) {
            hours = hours - 12;

        }

        minutesString = minutes + "";
        hoursString = hours + "";
        //  if minutes amount is less than 10 put a zero out front
        if (minutes < 10) {
         minutesString = "0" + minutes;

        }
        formattedTime = hoursString + ":" + minutesString + "" + aMPM;

        return formattedTime;

    }
}
```