Q1: Normalization [total 12 points]

import java.util.Arrays;  
import java.util.Scanner;  
  
public class Normalization {  
 // return the biggest elements of the input array, i.e., arr[] here.  
 // do NOT use the java-built in function Math.max [2 points]  
 public static double max(double[] arr){  
 // your solutions start here  
 double v = arr[0];  
 for (double value : arr) {  
 if (v < value) {  
 v = value;  
 }  
 }  
 return v;  
 }  
 // return the smallest elements of the input array, i.e., arr[] here.  
 // do NOT use the java-built in function Math.min [2 points]  
 public static double min(double[] arr){  
 // your solutions start here  
 double v = arr[0];  
 for (double value : arr) {  
 if (v > value) {  
 v = value;  
 }  
 }  
 return v;  
 }  
 // scaling the input array, i.e., arr[] here, so that each element is between 0 and 1  
 // by subtracting the minimum value from each element  
 // and then dividing each element by the difference  
 // between the minimum and maximum values. [4 points]  
 public static void scale(double[] arr){  
 // your solutions start here  
 if (arr.length == 0) {  
 throw new IllegalArgumentException("No valid input");  
 }  
 double min = *min*(arr);  
 double max = *max*(arr);  
 if (max == min) {  
 throw new IllegalArgumentException("Numbers all equal");  
 } else {  
 for (int i = 0; i < arr.length; i++) {  
 arr[i] = (arr[i] - min) / (max - min);  
 }  
 }  
 }  
 // get user’s input and output the scaled array [4 points]  
 public static void main(String[] args)  
 {  
 // ask user to enter some float-point or integer numbers  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.println("Enter some float-point or integer numbers (enter 'end' or non-numerical to terminate): ");  
 // and put them into java array with type ‘double’  
 double[] arr = new double[0];  
 while (scanner.hasNextInt() || scanner.hasNextDouble()) {  
 arr = Arrays.*copyOf*(arr, arr.length + 1);  
 arr[arr.length - 1] = Double.*parseDouble*(scanner.next());  
 }  
// System.out.println(Arrays.toString(arr));  
 // then call the method ‘scale’  
 try {  
 *scale*(arr);  
 // then output the ‘scaled’ (or normalized) array.  
 System.*out*.println("Scaled array: " + Arrays.*toString*(arr));  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("Exception thrown: " + e);  
 } finally {  
 scanner.close();  
 }  
 }  
}

Scanner scanner = new Scanner(System.*in*);  
 System.*out*.println("Enter some float-point or integer numbers (enter 'end' or non-numerical to terminate): ");  
 // and put them into java array with type ‘double’  
 double[] arr = new double[0];  
 while (scanner.hasNextInt() || scanner.hasNextDouble()) {  
 arr = Arrays.*copyOf*(arr, arr.length + 1);  
 arr[arr.length - 1] = Double.*parseDouble*(scanner.next());  
 }  
// System.out.println(Arrays.toString(arr));  
 // then call the method ‘scale’  
 try {  
 *scale*(arr);  
 // then output the ‘scaled’ (or normalized) array.  
 System.*out*.println("Scaled array: " + Arrays.*toString*(arr));  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("Exception thrown: " + e);  
 } finally {  
 scanner.close();  
 }  
 }  
}

Q2: CreditCard.java [10 points]

public class CreditCard  
{  
  
 private String customer;  
 private String bank;  
 private String account;  
 private int limit; // credit limit (the maximum you may owe)   
 protected double balance; // current balance (the amount you owe)   
  
  
 public CreditCard(String cust, String bk, String acnt, int lim, double initialBal) {  
 customer = cust;  
 bank = bk;  
 account = acnt;  
 limit = lim;  
 balance = initialBal;  
 if (initialBal>lim) {throw new IllegalArgumentException("Initial balance out of domain");}  
 if (lim<0) {throw new IllegalArgumentException("Credit limit out of domain");}  
 }  
  
  
 public CreditCard(String cust, String bk, String acnt, int lim) {  
  
 // call the previous constructor, but with an initial a balance of zero as default  
 this(cust, bk, acnt, lim, 0);  
 }  
  
  
 // develop get/access methods for each attribute defined above (total 5)  
 public String getCustomer() { return customer; }  
 public String getBank() { return bank; }  
 public String getAccount() { return account; }  
 public int getLimit() { return limit; }  
 public double getBalance() { return balance; }  
 // Charges the given price to the card, assuming sufficient credit limit.  
 public boolean charge(double price)  
 {  
 // return false, i.e. refuse the charge, if charge surpasses the limit  
 if (price<0) {  
 throw new IllegalArgumentException("Cannot charge negative price");  
 }  
 else if (price + balance > limit) {  
 return false;  
 }  
 // Return true otherwise (the charge is successful).  
 else {  
 balance += price;  
 return true;  
 }  
 }  
  
  
 // Processes customer payment that reduces balance.  
 public void makePayment(double amount)  
 {  
 if (amount<0){  
 throw new IllegalArgumentException("Cannot make negative payment");  
 } else {  
 balance -= amount;  
 }  
 }  
  
 // Print a card's information  
 public void printSummary() {  
 System.*out*.println("Customer = " + customer);  
 System.*out*.println("Bank = " + bank);  
 System.*out*.println("Account = " + account);  
 System.*out*.println("Balance = " + balance);  
 System.*out*.println("Limit = " + limit);  
 }  
  
 public static void main(String[] args) {  
  
 // define and initialize an array of CreditCard with 3 elements  
 CreditCard[] cc = new CreditCard[3];  
  
 // create 3 instances of CreditCard with different information  
 // and add them into the array above  
 try {  
 cc[0] = new CreditCard("User1", "Bank1", "111111", 5000);  
 cc[1] = new CreditCard("User2", "Bank2", "222222", 10000, 1000);  
 cc[2] = new CreditCard("User3", "Bank3", "222222", 2500, 400);  
 // for each element (card), charge them different prices  
 if (cc[0].charge(10000)) {  
 System.*out*.println("For"+cc[0].getCustomer()+": charge successful, balance after charge: " + cc[0].getBalance());  
 } else {  
 System.*out*.println("For"+cc[0].getCustomer()+": charge failed, charge surpasses the limit");  
 }  
 if (cc[1].charge(5000)) {  
 System.*out*.println("For"+cc[1].getCustomer()+": charge successful, balance after charge: " + cc[1].getBalance());  
 } else {  
 System.*out*.println("For"+cc[1].getCustomer()+": charge failed, charge surpasses the limit");  
 }  
 if (cc[2].charge(1000)) {  
 System.*out*.println("For"+cc[2].getCustomer()+": charge successful, balance after charge: " + cc[2].getBalance());  
 } else {  
 System.*out*.println("For"+cc[2].getCustomer()+": charge failed, charge surpasses the limit");  
 }  
  
 // using new for loop to  
 // (1) print out each card's information  
 // (2) for each card, continuously makePayment '200' so long as the balance is greater than 200  
 // and print out the balance correspondingly  
 for (CreditCard i : cc) {  
 i.printSummary();  
 }  
 for (CreditCard i : cc) {  
 while (i.getBalance() > 200) {  
 i.makePayment(200);  
 System.*out*.println("Balance after payment: " + i.getBalance());  
 }  
 }  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("Exception thrown: " + e);  
 }  
 }  
}

balance += price;  
 return true;  
 }  
 }  
  
  
 // Processes customer payment that reduces balance.  
 public void makePayment(double amount)  
 {  
 if (amount<0){  
 throw new IllegalArgumentException("Cannot make negative payment");  
 } else {  
 balance -= amount;  
 }  
 }  
  
 // Print a card's information  
 public void printSummary() {  
 System.*out*.println("Customer = " + customer);  
 System.*out*.println("Bank = " + bank);  
 System.*out*.println("Account = " + account);  
 System.*out*.println("Balance = " + balance);  
 System.*out*.println("Limit = " + limit);  
 }  
  
 public static void main(String[] args) {  
  
 // define and initialize an array of CreditCard with 3 elements  
 CreditCard[] cc = new CreditCard[3];  
  
 // create 3 instances of CreditCard with different information  
 // and add them into the array above  
 try {  
 cc[0] = new CreditCard("User1", "Bank1", "111111", 5000);  
 cc[1] = new CreditCard("User2", "Bank2", "222222", 10000, 1000);  
 cc[2] = new CreditCard("User3", "Bank3", "222222", 2500, 400);  
 // for each element (card), charge them different prices  
 if (cc[0].charge(10000)) {  
 System.*out*.println("For"+cc[0].getCustomer()+": charge successful, balance after charge: " + cc[0].getBalance());  
 } else {  
 System.*out*.println("For"+cc[0].getCustomer()+": charge failed, charge surpasses the limit");  
 }  
 if (cc[1].charge(5000)) {  
 System.*out*.println("For"+cc[1].getCustomer()+": charge successful, balance after charge: " + cc[1].getBalance());  
 } else {  
 System.*out*.println("For"+cc[1].getCustomer()+": charge failed, charge surpasses the limit");  
 }  
 if (cc[2].charge(1000)) {  
 System.*out*.println("For"+cc[2].getCustomer()+": charge successful, balance after charge: " + cc[2].getBalance());  
 } else {  
 System.*out*.println("For"+cc[2].getCustomer()+": charge failed, charge surpasses the limit");  
 }  
  
 // using new for loop to  
 // (1) print out each card's information  
 // (2) for each card, continuously makePayment '200' so long as the balance is greater than 200  
 // and print out the balance correspondingly  
 for (CreditCard i : cc) {  
 i.printSummary();  
 }  
 for (CreditCard i : cc) {  
 while (i.getBalance() > 200) {  
 i.makePayment(200);  
 System.*out*.println("Balance after payment: " + i.getBalance());  
 }  
 }  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("Exception thrown: " + e);  
 }  
 }  
}

System.*out*.println("For"+cc[2].getCustomer()+": charge failed, charge surpasses the limit");  
 }  
  
 // using new for loop to  
 // (1) print out each card's information  
 // (2) for each card, continuously makePayment '200' so long as the balance is greater than 200  
 // and print out the balance correspondingly  
 for (CreditCard i : cc) {  
 i.printSummary();  
 }  
 for (CreditCard i : cc) {  
 while (i.getBalance() > 200) {  
 i.makePayment(200);  
 System.*out*.println("Balance after payment: " + i.getBalance());  
 }  
 }  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("Exception thrown: " + e);  
 }  
 }  
}

Q3: TicTacToe.java [5 points]

public class TicTacToe  
{  
 public static final int *X* = 1, *O* = -1;   
 public static final int *EMPTY* = 0;   
 private int board[][] = new int[3][3];   
 private int player;   
  
 public TicTacToe() { clearBoard(); }  
  
  
 // Complete the method so that it resets all cells to be empty  
 // and set ‘X’ as the first player  
 public void clearBoard()   
 {  
 for (int i=0; i<3; i++){  
 for (int j=0; j<3; j++){  
 board[i][j]=*EMPTY*;  
 }  
 }  
 player = *X*;  
 }  
  
 // Puts an X or O mark at position i,j.  
 */\*\* Complete the method so that it prints the relevant   
 \* message when the player tries to put a mark outside the board or an occupied  
 \* position.  
 \*/* public void putMark(int i, int j)  
 {  
 boolean isOutsideBoard = i<0||i>=3||j<0||j>=3;  
 boolean isOccupiedPosition = board[i][j] != *EMPTY*;  
 try {  
 if (isOutsideBoard){throw new IllegalArgumentException("isOutsideBoard");}  
 if (isOccupiedPosition){throw new IllegalArgumentException("isOccupiedPosition");}  
 } catch (IllegalArgumentException e1) {  
 System.*out*.println("Exception thrown: " + e1);  
 return;  
 }  
 board[i][j] = player; // place the mark for the current player  
 player = - player; // switch players (uses fact that O = - X)  
 }  
  
 // Checks whether the board configuration is a win for the given player.   
 public boolean isWin(int mark)   
 {  
 return ((board[0][0] + board[0][1] + board[0][2] == mark\*3) // row 0  
 || (board[1][0] + board[1][1] + board[1][2] == mark\*3) // row 1  
 || (board[2][0] + board[2][1] + board[2][2] == mark\*3) // row 2  
 || (board[0][0] + board[1][0] + board[2][0] == mark\*3) // column 0  
 || (board[0][1] + board[1][1] + board[2][1] == mark\*3) // column 1  
 || (board[0][2] + board[1][2] + board[2][2] == mark\*3) // column 2  
 || (board[0][0] + board[1][1] + board[2][2] == mark\*3) // diagonal  
 || (board[2][0] + board[1][1] + board[0][2] == mark\*3)); // rev diag  
 }  
  
 // Returns the winning player's code, or 0 to indicate a tie (or unfinished game).  
 public int winner()   
 {  
 if (isWin(*X*))  
 return(*X*);  
 else if (isWin(*O*))  
 return(*O*);  
 else  
 return(0);  
 }  
  
 // Complete the method so that it returns a string representing/showing the current board  
 // Hint: you may use the class StringBuilder to fulfil the task  
 public String toString()   
 {  
 StringBuilder sb = new StringBuilder();  
 for (int i=0; i<3; i++) {  
 for (int j=0; j<3; j++) {  
 if (board[i][j] == *X*) {sb.append("X");}  
 else if (board[i][j] == *O*) {sb.append("O");}  
 else {sb.append(" ");}  
 if (j<2) {sb.append("|");}  
 }  
 if (i<2) {sb.append("\n-----\n");}  
 }  
 return sb.toString();  
 }  
  
   
 public static void main(String[] args)   
 {  
 TicTacToe game = new TicTacToe();  
  
 /\* X moves: \*/ /\* O moves: \*/  
 game.putMark(1,1); game.putMark(0,2);  
 game.putMark(2,2); game.putMark(0,0);  
 game.putMark(0,1); game.putMark(2,1);  
 game.putMark(1,2); game.putMark(1,0);  
 game.putMark(2,0);  
  
 System.*out*.println(game);  
 int winningPlayer = game.winner();  
 String[] outcome = {"O wins", "Tie", "X wins"}; // rely on ordering  
 System.*out*.println(outcome[1 + winningPlayer]);  
 }  
}

// row 1  
 || (board[2][0] + board[2][1] + board[2][2] == mark\*3) // row 2  
 || (board[0][0] + board[1][0] + board[2][0] == mark\*3) // column 0  
 || (board[0][1] + board[1][1] + board[2][1] == mark\*3) // column 1  
 || (board[0][2] + board[1][2] + board[2][2] == mark\*3) // column 2  
 || (board[0][0] + board[1][1] + board[2][2] == mark\*3) // diagonal  
 || (board[2][0] + board[1][1] + board[0][2] == mark\*3)); // rev diag  
 }  
  
 // Returns the winning player's code, or 0 to indicate a tie (or unfinished game).  
 public int winner()   
 {  
 if (isWin(*X*))  
 return(*X*);  
 else if (isWin(*O*))  
 return(*O*);  
 else  
 return(0);  
 }  
  
 // Complete the method so that it returns a string representing/showing the current board  
 // Hint: you may use the class StringBuilder to fulfil the task  
 public String toString()   
 {  
 StringBuilder sb = new StringBuilder();  
 for (int i=0; i<3; i++) {  
 for (int j=0; j<3; j++) {  
 if (board[i][j] == *X*) {sb.append("X");}  
 else if (board[i][j] == *O*) {sb.append("O");}  
 else {sb.append(" ");}  
 if (j<2) {sb.append("|");}  
 }  
 if (i<2) {sb.append("\n-----\n");}  
 }  
 return sb.toString();  
 }  
  
   
 public static void main(String[] args)   
 {  
 TicTacToe game = new TicTacToe();  
  
 /\* X moves: \*/ /\* O moves: \*/  
 game.putMark(1,1); game.putMark(0,2);  
 game.putMark(2,2); game.putMark(0,0);  
 game.putMark(0,1); game.putMark(2,1);  
 game.putMark(1,2); game.putMark(1,0);  
 game.putMark(2,0);  
  
 System.*out*.println(game);  
 int winningPlayer = game.winner();  
 String[] outcome = {"O wins", "Tie", "X wins"}; // rely on ordering  
 System.*out*.println(outcome[1 + winningPlayer]);  
 }  
}

System.*out*.println(game);  
 int winningPlayer = game.winner();  
 String[] outcome = {"O wins", "Tie", "X wins"}; // rely on ordering  
 System.*out*.println(outcome[1 + winningPlayer]);  
 }  
}