**CS230 Assignment 3**

**Author:** Diana Eastman

**Date Submitted:** February 19, 2013

**Collaborators:** None.

**Notes:**

1. No known bugs.

2. The programs work as indicated in the comments and Javadocs.

3. To demonstrate that the intended functionality has been achieved, I have included a soft copy of testing typescript.

4. The methods from Sort.java are instance methods in GradSchools.java

/\*\*

\* School.java

\* Purpose: Defines the CS program ratings in terms of Academics, Publication Impact,

\* Research, and Overall Rating of a single School

\* Written by: Diana Eastman

\* Modified date: February 19, 2013

\*/

public class School {

private String name;

private int academicRating;

private int researchRating;

private int pubImpactRating;

private int overallRating;

public int rank;

/\*\*

\* School Constructor

\* @param name

\* @param aRating (Academic rating)

\* @param rRating (Research rating)

\* @param pRating (Publication impact rating)

\*/

public School (String name, int aRating, int rRating, int pRating)

{

this.name = name;

academicRating = aRating;

researchRating = rRating;

pubImpactRating = pRating;

overallRating = 1;

rank = 0;

}

/\*\*

\* Getter for School name.

\* @return name

\*/

public String getName()

{

return name;

}

/\*\*

\* Getter for academicRating.

\* @return academicRating

\*/

public int getAcademicRating()

{

return academicRating;

}

/\*\*

\* Getter for ResearchRating.

\* @return researchRating

\*/

public int getResearchRating()

{

return researchRating;

}

/\*\*

\* Getter for pubImpactRating.

\* @return pubImpactRating

\*/

public int getPubImpactRating() {

return pubImpactRating;

}

/\*\*

\* Getter for overallRating.

\* @return overallRating

\*/

public int getOverallRating()

{

return overallRating;

}

/\*\*

\* Compute overallRating by multiplying each factor by a given weight (1-5)

\* @return overallRating

\*/

public int computeRating (int aweight, int rweight, int pweight)

{

overallRating = (aweight\*academicRating) + (rweight\*researchRating) + (pweight\*pubImpactRating);

return overallRating;

}

/\*\*

\* toString() method to display School Class

\* @return String

\*/

public String toString()

{

String results ="";

results += "Name: " + name + "\n";

results += "Academics: " + academicRating + "\n";

results += "Research: " + researchRating + "\n";

results += "Publications: " + pubImpactRating + "\n";

results += "Overall rating: " + overallRating + "\n";

results += "Rank: " + rank + "\n";

results += "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n";

return results;

}

//Testing method

public static void main(String args[])

{

School mit = new School("MIT", 10, 10, 7);

School stanford = new School("Stanford", 8, 10, 9);

School cmu = new School("CMU", 7, 8, 6);

School berkeley = new School("UC Berkeley", 9, 9, 9);

berkeley.computeRating(4, 5, 3);

stanford.computeRating(4, 5, 3);

cmu.computeRating(4, 5, 3);

mit.computeRating(4, 5, 3);

System.out.println(mit);

System.out.println(stanford);

System.out.println(cmu);

System.out.println(berkeley);

}

} // End School Class

/\*\*

\* GradSchools.java

\* Purpose: Stores information about a collection of CS grad schools and provides methods to sort the schools

\* according to four, weighted factors (Academics, Publication Impact, Research, and Overall Rating)

\* Written by: Diana Eastman

\* Modified date: February 19, 2013

\*/

import java.util.\*;

public class GradSchools {

private School[] collection;

private int count;

private int test;

/\*\*

\* School Constructor

\* Creates a collection of schools

\* and initializes count variable to 0

\*/

public GradSchools ()

{

collection = new School[4]; //Set at 4 to show that the addSchool method works when 5th school added in main

count = 0;

}

/\*\*

\* Loops through command line parameters,

\* passing to isInRange helper method; returns false

\* when encounters integer in the array that is out of range

\* @param integer array of weights arr

\* @return boolean

\*/

public static boolean allInRange(int[] arr)

{

for (int i = 0; i < arr.length; i ++)

{

if (!isInRange(arr[i])) return false;

}

return true;

}

/\*\*

\* Tests if integer is between 1 and 5

\* @param integer i

\* @return boolean

\*/

private static boolean isInRange(int i)

{

return i > 0 && i < 6;

}

/\*\*

\* Adds a school to the collection array and increments count variable

\* @param name, aRating (Academic rating), rRating (research rating),

\* pRating (Publication Impact rating)

\*/

public void addSchool (String name, int aRating, int rRating, int pRating)

{

if (count==collection.length) //Calls increaseSize() if count=current size of collection array so another school can be added

increaseSize();

collection[count] = new School(name, aRating, rRating, pRating);

count ++;

}

/\*\*

\* Doubles the size of the collection array by creating a temp array twice as large

\* and copying the collection array into the new temp array

\* from Java Foundations

\*/

private void increaseSize ()

{

School[] temp = new School[collection.length\*2];

for (int i=0; i <collection.length; i++)

{

temp[i]=collection[i];

}

collection = temp;

}

/\*\*

\* Computes the weighted rating of each school, based on command line

\* parameters

\* @param, aweight (weight for Academics), rweight (weight for Research), pweight (weight for Pub. Impact)

\*/

private void computeRating (int aweight, int rweight, int pweight)

{

int i =0;

for (i = 0; i < count; i++)

{

collection[i].computeRating(aweight, rweight, pweight);

}

}

/\*\*

\* Loops through the Schools in the collection and stores the appropriate factor (based on input parameter)

\* into the rank variable; calls sortArray() method

\* @param rankbyElement (determines which factor to sort on)

\*/

private void rankSchools (String rankbyElement)

{

int i =0;

for (i = 0; i < count; i ++)

{

if (rankbyElement.equals("Academics"))

{

collection[i].rank = collection[i].getAcademicRating();

}

else if (rankbyElement.equals("Research"))

{

collection[i].rank = collection[i].getResearchRating();

}

else if (rankbyElement.equals("Publications"))

{

collection[i].rank = collection[i].getPubImpactRating();

}

else

{

collection[i].rank = collection[i].getOverallRating();

}

}

sortArray();

}

/\*

\* Sorts the integers in the input array in descending order and calls printOut()

\* Explanation (Task 1):

\* Works by dividing the array to be sorted into two parts: the part that is already sorted

\* and the part that isn't (initially, the sorted portion is empty).

\* At each step in the algorithm, the sorted portion of the array is extended by one element.

\* The inner for loop finds the largest element in the unsorted portion of the array and

\* swaps it with the element with the index as the first counter (at first pass through

\* elements at [0] and [1]) Array size - 1 passes through the loop are required to sort the entire array.

\* Modified from Sort.java

\*/

public void sortArray ()

{

int maxNum; // maximum integer so far

int maxIndex; // index of maximum integer

int i, j;

for (j = count - 1; j > 0; j--)

{

maxIndex = 0;

maxNum = collection[0].rank;

for (i = 1; i <= j; i++)

if (collection[i].rank < maxNum) //descending order

{

maxNum = collection[i].rank;

maxIndex = i;

}

swap(collection, maxIndex, j);

}

printOut(collection);

}

/\*\*

\* exchanges the contents of locations i and j in the input array

\*/

private void swap (School[] collection, int i, int j)

{

School temp = collection[i];

collection[i] = collection[j];

collection[j] = temp;

}

/\*\*

\* Prints the name of each School in the collection in descending order after sorting

\*/

private void printOut (School[] collection)

{

int i=0;

for (i = 0; i<count; i++)

{

System.out.println(collection[i].getName());

}

}

/\*\*

\* toString method to print out GradSchool object

\* @return String

\*/

final public String toString()

{

String result ="";

result += "There are " + count + " schools in the database \n";

for (int i = 0; i < count; i++) {

result += collection[i].toString();

}

return result;

}

//Testing method - takes in three parameters from the command line

public static void main(String args[])

{

if ((args.length != 3)) //Exit if three weights not provided

{

System.out.println("Please provide 3 weights between 1 and 5 for Academics, Research and Publications");

System.exit(0);

}

try {

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

int rweight = Integer.parseInt(args[1]);

int pweight = Integer.parseInt(args[2]);

int [] arr = {aweight, rweight, pweight};

//Calls allInRage method and proceeds if true

if (allInRange(arr))

{

/\*Instantiate GradSchools object and add schools to database\*/

GradSchools myschools = new GradSchools();

myschools.addSchool("MIT", 10, 10, 7);

myschools.addSchool("Stanford", 8, 10, 9);

myschools.addSchool("CMU", 7, 8, 6);

myschools.addSchool("UC Berkeley", 9, 9, 9);

myschools.addSchool("Cornell", 8, 9, 6); //Add 5th school to show that increaseSize() method functions properly

myschools.computeRating(aweight, rweight, pweight);

/\*Print output\*/

System.out.println(myschools);

System.out.println("\n" + "Ranking of schools from highest to lowest using Academics as a factor:");

myschools.rankSchools("Academics");

System.out.println("\n" + "Ranking of schools from highest to lowest using Publication Impact as a factor:");

myschools.rankSchools("Publications");

System.out.println("\n" + "Ranking of schools from highest to lowest using Research as a factor:");

myschools.rankSchools("Research");

System.out.println( "\n" + "Ranking of schools from highest to lowest using Overall as a factor:");

myschools.rankSchools("Overall");

}

else

{

System.out.println("One or more values are out of range.");

}

}

catch (NumberFormatException e) {

System.out.println("Sorry, one of your inputs is not an integer.");

System.out.println("Please provide 3 weights between 1 and 5, for Academics Research and Publications");

}

}

} // End GradSchool Class

/\*\*

\* Account.java

\* Purpose: Defines the Abstract Account Class for a bank account

\* Written by: Diana Eastman

\* Modified date: February 19, 2013

\* Methods:

\* Abstract method withdraw() - to be implemented in inheriting classes

\* method toString()

\* method getBalance()

\* method getAccountType()

\* final method deposit()

\*/

import java.text.NumberFormat; //used to format output

import java.util.\*; //used for Random number generation

public abstract class Account {

private int accountNumber;

protected double balance;

protected String type;

// Get currency and percent instances to use in formatting printed values

protected static NumberFormat fmtCurrency = NumberFormat.getCurrencyInstance();

protected static NumberFormat fmtPerCent = NumberFormat.getPercentInstance();

/\*\*

\* Account Constructor

\* @param balance the balance of the account-required to set up an account

\* @param type the type of account (Savings Account or Checking Account)

\* @exception IllegalArgumentException thrown if object constructed with negative balance

\* Exceptions are needed here to ensure proper construction of Account object. The other way to

\* accomplish this would be to take in the parameters, interactively, from the user by way of the

\* Scanner class. Since this program did not have to be interactive, I do the validation here.

\*/

public Account(double balance, String type)

{

this.balance = balance;

this.type = type;

if (this.balance <= 0)

{

throw new IllegalArgumentException(

"Balance out of range: " + this.balance);

}

Random rand = new Random();

int min =1000000;

int max =9999999;

accountNumber = rand.nextInt(max - min + 1) + min;

}

/\*\*

\* Final deposit method

\* Deposit must be greater than zero

\*/

protected final void deposit(double amount)

{

if (amount > 0)

{

balance += amount;

}

else

{

System.out.println("You've tried to deposit an invalid amount. Transaction cancelled.");

}

}

/\*\*

\* Account balance getter method

\* @return double balance

\*/

public double getBalance()

{

return balance;

}

/\*\*

\* Account type getter method (Checking or Savings)

\* @return String account type

\*/

public String getAccountType()

{

return type;

}

/\*\*

\* Method abstract withdraw()

\* To be implemented by the inheriting class.

\*/

abstract public void withdraw(double amount);

/\*\*

\* Method to Display the Class

\* @return String

\*/

public String toString() {

String result = "";

result += ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" + "\n");

result += (type + " # " + accountNumber + " has balance: " + fmtCurrency.format(getBalance()));

return result;

}

} // End Abstract Account Class

/\*\*

\* CheckingAccount.java

\* Purpose: Defines the Class for a Checking Account (extends Account.java)

\* Written by: Diana Eastman

\* Modified date: February 19, 2013

\* Methods:

\* method compareTo()

\* method to String()

\* final method deposit()

\*/

//Inherits acccount number and methods deposit() and getbalance() from Parent Account.java

import java.text.DecimalFormat; //used to format output

public class CheckingAccount extends Account {

private final double minimumBalance = 50;

private final double overdraftFee = 25;

/\*\*

\* Account Constructor

\* @param balance the balance of the account; required to set up an account

\* @param type the account type (String), Checking or Savings

\*/

public CheckingAccount(double balance, String type)

{

super(balance, type);

}

/\*\*

\* Defines Abstract method withdraw from parent Account.java

\* @param amount the amount to withdraw

\*/

public void withdraw(double amount)

{

if (amount > 0) { //All withdrawals must be greater than zero or they are invalid

if (balance - amount < minimumBalance) { //Incur fee if drops below minimumBalance

balance = balance - amount - overdraftFee;

}

else {

balance -= amount;

}

}

else {

System.out.println("You've tried to withdraw an invalid amount. Transaction cancelled.");

amount = 0;

}

}

/\*\*

\* overdraftFee getter method

\* @return overdraftFee

\*/

public double getoverdraftFee()

{

return overdraftFee;

}

/\*\*

\* Method toString() prints Class - calls the toString() of parent Account.java

\* @return String

\*/

public String toString() {

String results = "";

results += super.toString() + "\n";

results += "Wellesley College Bank checking accounts have a minimum balance of: " +

fmtCurrency.format(minimumBalance) + "\n";

results += ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" + "\n");

return results;

}

//Testing

public static void main(String args[]) {

CheckingAccount c1= new CheckingAccount(100, "Checking Account");

System.out.println("The balance of checking account 1 is: " + c1.getBalance());

c1.withdraw(10.50);

System.out.println("The balance of checking account 1 is: " + c1.getBalance() + " after withdrawing $10.50");

System.out.println(c1);

}

} // End Checking Account Class

/\*\*

\* SavingsAccount.java

\* Purpose: Defines the Class for a Savings Account (extends Account.java)

\* Written by: Diana Eastman

\* Modified date: February 19, 2013

\* Methods:

\* public void method withdraw()

\* private void method addInterest()

\* public double method getInterest()

\* method to String()

\*/

//Inherits acccount number and methods deposit() and getbalance() from Parent Account.java

public class SavingsAccount extends Account {

private static final double interestRate = 0.055;

private double interest;

/\*\*

\* Account Constructor

\* @param balance the balance of the account; required to set up an account

\*/

public SavingsAccount(double balance, String type)

{

super(balance, type);

interest = 0;

}

/\*\*

\* Defines Abstract method withdraw from parent Account.java

\* @param amount the amount to withdraw

\*/

public void withdraw(double amount)

{

if ((balance >= amount) && (amount>0)) { //Savings Accounts have no overdraft fee- balance must be >=amount

balance = balance - amount;

}

else {

System.out.println("You've tried to withdraw an invalid amount. Transaction cancelled.");

amount = 0;

}

}

/\*\*

\* Add monthly interest

\*/

public void addInterest()

{

interest = balance \* (interestRate/12);

balance = balance + interest;

}

/\*\*

\* Interest getter

\* @return interest rate

\*/

public double getInterest()

{

return interest;

}

/\*\*

\* Method toString() prints Class - calls the toString() of parent Account.java

\* @return String

\*/

public String toString() {

fmtPerCent.setMaximumFractionDigits(1); // Display interest rate percentage to the tenths place value

String results = "";

results += super.toString() + "\n";

results += "The current interest rate of Wellesley College Bank savings accounts is: " +

fmtPerCent.format(interestRate) + "\n";

results += ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" + "\n");

return results;

}

//Testing

public static void main(String args[]) {

SavingsAccount s1= new SavingsAccount(100, "Savings Account");

System.out.println("The balance of savings account 1 is: " + s1.getBalance());

s1.deposit(1000);

s1.withdraw(800);

s1.withdraw(-25);

System.out.println("The balance of savings account 1 is: " + s1.getBalance() + " after depositing $1000");

s1.addInterest();

System.out.println("The interest you earned at the end of the month is: " + s1.getInterest());

System.out.println(s1);

}

} // End Savings Account Class

/\*\*

\* TestAccount.java

\* Purpose: Test Account.java, CheckingAccount.java and SavingsAccount.java

\* Written by: Diana Eastman

\* Modified date: February 19, 2013

\*/

import java.text.NumberFormat; //used to format output

public class TestAccount {

protected static NumberFormat fmtCurrency = NumberFormat.getCurrencyInstance();

public static void main(String[ ] args)

{

/\* CheckingAccount Class \*/

CheckingAccount wendyWellesley= new CheckingAccount(300, "Checking Account");

System.out.println("The initial balance of Wendy Wellesley's " + wendyWellesley.getAccountType() + " is: " +

fmtCurrency.format(wendyWellesley.getBalance()));

wendyWellesley.withdraw(100.5);

System.out.println("The balance of Wendy Wellesley's " + wendyWellesley.getAccountType() + " is " +

fmtCurrency.format(wendyWellesley.getBalance()) + " after withdrawing $100.50");

wendyWellesley.withdraw(-100.5);

System.out.println("The balance of Wendy Wellesley's " + wendyWellesley.getAccountType() + " is " +

fmtCurrency.format(wendyWellesley.getBalance()) + " after trying to withdraw $-100.50");

wendyWellesley.deposit(25);

System.out.println("The balance of Wendy Wellesley's " + wendyWellesley.getAccountType() + " is " +

fmtCurrency.format(wendyWellesley.getBalance()) + " after depositing $25");

wendyWellesley.withdraw(400);

System.out.println("The balance of Wendy Wellesley's " + wendyWellesley.getAccountType() + " is " +

fmtCurrency.format(wendyWellesley.getBalance()) + " after withdrawing $400 " +

"and incurring an overdraft fee of " + fmtCurrency.format(wendyWellesley.getoverdraftFee()));

System.out.println(wendyWellesley); //Print instance (Wendy Wellesley) of Checking Account object

/\* SavingsAccount Class \*/

SavingsAccount winnieWellesley= new SavingsAccount(100, "Savings Account");

System.out.println("The initial balance of Winnie Wellesley's " + winnieWellesley.getAccountType() + " is " +

fmtCurrency.format(winnieWellesley.getBalance()));

winnieWellesley.withdraw(100.5);

System.out.println("The balance of Winnie Wellesley's " + winnieWellesley.getAccountType() + " is " +

fmtCurrency.format(winnieWellesley.getBalance()) + " after attempting to withdraw $100.50");

winnieWellesley.deposit(3000.5);

System.out.println("The balance of Winnie Wellesley's " + winnieWellesley.getAccountType() + " is " +

fmtCurrency.format(winnieWellesley.getBalance()) + " after depositing $3000.50");

winnieWellesley.addInterest(); //Simulate end of month and add interest

System.out.println("After one month, Winnie Wellesley's " + winnieWellesley.getAccountType() + " earned " +

fmtCurrency.format(winnieWellesley.getInterest()) + " in interest");

System.out.println(winnieWellesley); //Print instance (Winnie Wellesley) of Savings Account object

/\* Test border cases \*/

CheckingAccount zeroCase= new CheckingAccount(0, "Checking Account");

CheckingAccount negativeCase= new CheckingAccount(-100, "Savings Account");

}//End main()

} //End TestAccount