**Shay Walker Project 1 2-5-2018**

**Professor Sabal**

**Data Structures & Algorithms**

R-2.4) Assume that we change the CreditCard class so that instance variable balance has private visibility. Why is the following implementation of the PredatoryCreditCard.charge method flawed?

public boolean charge(double price) {

boolean isSuccess = super.charge(price);

if(!isSuccess)

charge(5); //penalty for late fee

return isSuccess;

}

\*(Goodrich, page 97)

If super.charge(price); returns false then charge(5); ,is called and you are now one layer deeper. With this code and code fragment 1.5 from page 42, if the statement keeps returning false then charge(5); will continued to be called in deeper and deeper levels. This will cause problems (recursive algorithm).

R-2.6) Give a short fragment of java code that uses the progression classes form section 2.2.3 to find the eight value of a Fibonacci progression that starts with 2 and 2 as its first two values.

After much confusion and help with review in class, it was pointed out that really what this question is asking is for us to make a test that will output up to the first eight terms starting with 2,2. The code is complete for the fibonacci progression, therefore only two lines are needed to make a test work and provide the solution.

public class FibonacciTest extends TestCase {

@Test

public void test () {

FibonacciProgression fp=new FibonacciProgression ((long)2,(long)2);

fp.printProgression(8);

}

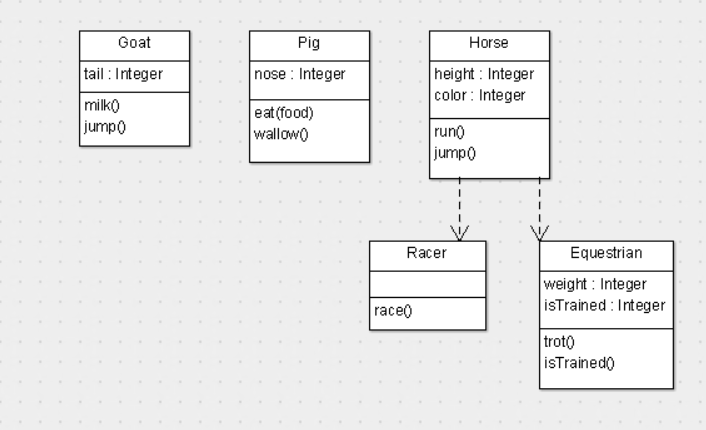
}

R 2.11

Calling the main() method of the Maryland class will provide the output of (1)Reading it, then

(2)Shipping it,Then (3) Buy it, then (4) Read it, then (5)Box it, then (6)Read it yet again.

R 2.12



C.19

First listed is the creditCard Class which is code directly from the book. Beneath is the predatoryCreditCard Class which extends the ceditCard Class. Therefore, the predaory CreditCard Class will meet inheritance standards, and will use some of the code from its parent class creditCard. All changes will be highlighted in yellow.

/\*

\* Copyright 2014, Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser

\*

\* Developed for use with the book:

\*

\* Data Structures and Algorithms in Java, Sixth Edition

\* Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser

\* John Wiley & Sons, 2014

\*

\* This program is free software: you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation, either version 3 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program. If not, see <http://www.gnu.org/licenses/>.

\*/

/\*\*

\* A simple model for a consumer credit card.

\*

\* @author Michael T. Goodrich

\* @author Roberto Tamassia

\* @author Michael H. Goldwasser

\*/

public class CreditCard {

// Instance variables:

private String customer; // name of the customer (e.g., "John Bowman")

private String bank; // name of the bank (e.g., "California Savings")

private String account; // account identifier (e.g., "5391 0375 9387 5309")

private int limit; // credit limit (measured in dollars)

protected double balance; // current balance (measured in dollars)

// Constructors:

/\*\*

\* Constructs a new credit card instance.

\* @param cust the name of the customer (e.g., "John Bowman")

\* @param bk the name of the bank (e.g., "California Savings")

\* @param acnt the account identifier (e.g., "5391 0375 9387 5309")

\* @param lim the credit limit (measured in dollars)

\* @param initialBal the initial balance (measured in dollars)

\*/

public CreditCard(String cust, String bk, String acnt, int lim, double initialBal) {

customer = cust;

bank = bk;

account = acnt;

limit = lim;

balance = initialBal;

}

/\*\*

\* Constructs a new credit card instance with default balance of zero.

\* @param cust the name of the customer (e.g., "John Bowman")

\* @param bk the name of the bank (e.g., "California Savings")

\* @param acnt the account identifier (e.g., "5391 0375 9387 5309")

\* @param lim the credit limit (measured in dollars)

\*/

public CreditCard(String cust, String bk, String acnt, int lim) {

this(cust, bk, acnt, lim, 0.0); // use a balance of zero as default

}

// Accessor methods:

/\*\* Returns the name of the customer. \*/

public String getCustomer() { return customer; }

/\*\* Returns the name of the bank \*/

public String getBank() { return bank; }

/\*\* Return the account identifier. \*/

public String getAccount() { return account; }

/\*\* Return the credit limit. \*/

public int getLimit() { return limit; }

/\*\* Return the current balance. \*/

public double getBalance() { return balance; }

// Update methods:

/\*\*

\* Charges the given price to the card, assuming sufficient credit limit.

\* @param price the amount to be charged

\* @return true if charge was accepted; false if charge was denied

\*/

public boolean charge(double price) { // make a charge

if (price + balance > limit) // if charge would surpass limit

return false; // refuse the charge

// at this point, the charge is successful

balance += price; // update the balance

return true; // announce the good news

}

/\*\*

\* Processes customer payment that reduces balance.

\* @param amount the amount of payment made

\*/

public void makePayment(double amount) { // make a payment

balance -= amount;

}

// Utility method to print a card's information

public static void printSummary(CreditCard card) {

System.out.println("Customer = " + card.customer);

System.out.println("Bank = " + card.bank);

System.out.println("Account = " + card.account);

System.out.println("Balance = " + card.balance); // implicit cast

System.out.println("Limit = " + card.limit); // implicit cast

}

public static void main(String[] args) {

CreditCard[] wallet = new CreditCard[3];

wallet[0] = new CreditCard("John Bowman", "California Savings",

"5391 0375 9387 5309", 5000);

wallet[1] = new CreditCard("John Bowman", "California Federal",

"3485 0399 3395 1954", 3500);

wallet[2] = new CreditCard("John Bowman", "California Finance",

"5391 0375 9387 5309", 2500, 300);

for (int val = 1; val <= 16; val++) {

wallet[0].charge(3\*val);

wallet[1].charge(2\*val);

wallet[2].charge(val);

}

for (CreditCard card : wallet) {

CreditCard.printSummary(card); // calling static method

while (card.getBalance() > 200.0) {

card.makePayment(200);

System.out.println("New balance = " + card.getBalance());

}

}

}

}

/\*

Output of main:

Customer = John Bowman

Bank = California Savings

Account = 5391 0375 9387 5309

Balance = 408.0

Limit = 5000

New balance = 208.0

New balance = 8.0

Customer = John Bowman

Bank = California Federal

Account = 3485 0399 3395 1954

Balance = 272.0

Limit = 3500

New balance = 72.0

Customer = John Bowman

Bank = California Finance

Account = 5391 0375 9387 5309

Balance = 436.0

Limit = 2500

New balance = 236.0

New balance = 36.0

\*/

predatoryCreditCard Class

/\*

\* Copyright 2014, Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser

\*

\* Developed for use with the book:

\*

\* Data Structures and Algorithms in Java, Sixth Edition

\* Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser

\* John Wiley & Sons, 2014

\*

\* This program is free software: you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation, either version 3 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program. If not, see <http://www.gnu.org/licenses/>.

\*/

public class PredatoryCreditCard extends CreditCard {

// Additional instance variable

private double apr; // annual percentage rate

// Essentially makes this information available to all section of code to use

private boolean paymentMade

// Essentially makes this information available to all sections of code to use (like a Global var)

private double minimumPayment

/\*\*

\* Constructs a new credit card instance.

\*

\* @param cust the name of the customer (e.g., "John Bowman")

\* @param bk the name of the bank (e.g., "California Savings")

\* @param acnt the account identifier (e.g., "5391 0375 9387 5309")

\* @param lim the credit limit (measured in dollars)

\* @param initialBal the initial balance (measured in dollars)

\* @param rate the annual percentage rate (e.g., 0.0825 for 8.25% APR)

\*/

public PredatoryCreditCard(String cust, String bk, String acnt, int lim,

double initialBal, double rate) {

super(cust, bk, acnt, lim, initialBal); // initialize superclass attributes

apr = rate;

// States that payment has not been made, and calls paymentMade when it is equal to False.

paymentMade = false;

//States that the minimum payment is equal to two percent of the initial balance calculated in //creditCard Class

minimumPayment = initialBal \* 0.02;

}

/\*\* Assess monthly interest on any outstanding balance. \*/

public void processMonth() {

if (balance > 0) { // only charge interest on a positive balance

double monthlyFactor = Math.pow(1 + apr, 1.0/12); // compute monthly rate

balance \*= monthlyFactor; // assess interest

//If the payment is not made on time or otherwise, he consumers balanced is charged $25

if (paymentMade == false)

charge(25);

}

//Sets paymentMade to False

paymentMade = false;

}

@Override

/\*\*

\* Charges given price to the card, assuming sufficient credit limit.

\* Returns true if charge was processed.

\* Returns false and assesses $5 fee if charge is denied.

\*/

public boolean charge(double price) {

boolean isSuccess = super.charge(price); // call inherited method

if (!isSuccess)

balance += 5; // assess a \$5 penalty

//Calculates the minimum payment by multiplying the current balance by 2%

minimumPayment = balance \* 0.02;

return isSuccess;

}

public static void main(String[] args) {

PredatoryCreditCard card = new PredatoryCreditCard("Michael", "Payday", "1111 1111 1111 1111", 2500, 0, 0.0825);

card.charge(1000);

CreditCard.printSummary(card);

card.processMonth(); // interest charged

CreditCard.printSummary(card);

card.charge(10000); // should be rejected, with penalty

CreditCard.printSummary(card);

}

public static void makePayment(double amount) {

// Method to track whether cosumers pay the minimum balance or not.

if (amount >= minimumPayment) {

// setting to true makes the code not loop, and overcharge the consumer

paymentMade = true;

}

// super. makePayment call this method out of the Parent class (Credit Card)

super.makePayment(amount)

}

}

P.35

Below is the WordList Class and the TestProject 1. Comments will be highlighted in green to give a basic explanation of what this code does.

public class WordList {

private String[] listOfWords;

public WordList(String testInput) {

//Defines The String and calls the testImput to be parsed by spaces

listOfWords = testInput.split(" ");

//Outputs the number of words in the String

System.out.println(listOfWords.length);

}

public static void main(String[] args) {

// TODO Auto-generated method stub

}

public Integer wordCount(String word) {

// TODO Auto-generated method stub

// Starts count at 0

Integer count = 0;

// The counter must be less than the length of the String to count another word

for (int wordIndex=0;wordIndex<listOfWords.length;wordIndex++) {

//Still counts repeated words

if (listOfWords[wordIndex].equalsIgnoreCase(word)) count ++;

}

System.out.println(count);

return count;

}

}

**Test:**

// Uses other junits to complete

import static org.junit.Assert.\*;

import org.junit.Test;

public class TestProject1 {

@Test

public void test() {

// Defines test input string for testing

String testInput = "Red Blue Green Pink Red Orange Yellow Blue Green Pink

Purple";

WordList counter = new WordList(testInput);

// Counts expected repeated words

assertEquals(Integer.valueOf(2),counter.wordCount("Red"));

}

}