­­For this homework assignment, you will be writing code in support of a Dessert Shoppe which sells candy by the pound, cookies by the dozen, ice cream, and sundaes (ice cream with a topping). Your software will be used for the checkout system.

To do this, you will implement an inheritance hierarchy of classes derived from a **DessertItem** *abstract* superclass.

The **Candy**, **Cookie**, and **IceCream** classes will be derived from the **DessertItem** class.

The **Sundae** class will be derived from the **IceCream** class.

You will also write a **Checkout** class which maintains a list (Vector) of **DessertItem's**.

**The DessertItem Class**

The **DessertItem** class is an *abstract superclass* from which specific types of **DessertItems** can be derived. It contains only one data member, a name. It also defines a number of methods. All of the **DessertItem** class methods except the **getCost()** method are defined in a generic way in the file Class-DessertItem, provided for you along with the other homework specific files in the directory. The **getCost()** method is an *abstract method* that is not defined in the **DessertItem** class because the method of determining the costs varies based on the type of item. Tax amounts should be rounded to the nearest cent. For example, the calculating the tax on a food item with a cost of 199 cents with a tax rate of 2.0% should be 4 cents. The complete specifications for the **DessertItem** class are provided in the file Class-DessertItem.

**package** Assignment5;

**public** **abstract** **class** DessertItem {

**protected** String name;

**public** DessertItem() {

**this**("");

}

**public** DessertItem(String name) {

**if** (name.length() <= DessertShoppe.***MAX\_ITEM\_NAME\_SIZE***)

**this**.name = name;

**else**

**this**.name = name.substring(0,DessertShoppe.***MAX\_ITEM\_NAME\_SIZE***);

}

**public** String getName() {

**return** name;

}

**public** **abstract** **int** getCost();

}

**The DessertShoppe Class**

The **DessertShoppe class**  contains constants such as the tax rate as well the name of the store, the maximum size of an item name and the width used to display the costs of the items on the receipt. Your code should use these constants wherever necessary! The DessertShoppe class also contains the **cents2dollarsAndCents**method which takes an integer number of cents and returns it as a String formatted in dollars and cents. For example, 105 cents would be returned as "1.05".

**package** Assignment5;

**public** **class** DessertShoppe {

**public** **final** **static** **double** ***TAX\_RATE*** = 6.5; // 6.5%

**public** **final** **static** String ***STORE\_NAME*** = "M & M Dessert Shoppe";

**public** **final** **static** **int** ***MAX\_ITEM\_NAME\_SIZE*** = 25;

**public** **final** **static** **int** ***COST\_WIDTH*** = 6;

**public** **static** String cents2dollarsAndCents(**int** cents) {

String s = "";

**if** (cents < 0) {

s += "-";

cents \*= -1;

}

**int** dollars = cents/100;

cents = cents % 100;

**if** (dollars > 0)

s += dollars;

s +=".";

**if** (cents < 10)

s += "0";

s += cents;

**return** s;

}

}

**The Derived Classes**

All of the classes which are derived from the **DessertItem** class must define a constructor. The **TestCheckout** class determine the parameters for the various constructors. Each derived class should be implemented by creating a file with the correct name, eg., **Candy.java**.

The **Candy** class should be derived from the **DessertItem** class. A **Candy** item has a *weight* and a *price per pound* which are used to determine its *cost*. For example, 2.30 lbs.of fudge @ .89 /lb. = 205 cents. The cost should be rounded to the nearest cent.

**package** Assignment5;

**public** **class** Candy **extends** DessertItem {

**protected** **double** weight;

**protected** **double** pricePerPound;

**public** Candy(String \_n, **double** \_ppp, **int** \_w) {

// using parent's constructor with name while storing its own properties

**super**(\_n);

pricePerPound = \_ppp;

weight = \_w;

}

**public** **int** getCost() {

**return** (**int**) Math.*round*(weight \* pricePerPound);

}

}

The **Cookie** class should be derived from the **DessertItem** class. A **Cookie** item has a *number* and a *price per dozen* which are used to determine its *cost*. For example, 4 cookies @ 399 cents /dz. = 133 cents. The cost should be rounded to the nearest cent.

**package** Assignment5;

**public** **class** Cookie **extends** DessertItem {

**protected** **double** number;

**protected** **double** pricePerDoze;

**public** Cookie(String \_n, **double** \_ppd, **int** \_number) {

**super**(\_n);

pricePerDoze = \_ppd;

number = \_number;

}

**public** **int** getCost() {

**return** (**int**) Math.*round*(number / 12 \* pricePerDoze);

}

}

The **IceCream** class should be derived from the **DessertItem** class. An **IceCream** item simply has a *cost*.

**package** Assignment5;

**public** **class** IceCream **extends** DessertItem {

**protected** **int** cost;

**public** IceCream(String \_n, **int** \_cost) {

**super**(\_n);

cost = \_cost;

}

**public** **int** getCost() {

**return** cost;

}

}

The **Sundae** class should be derived from the **IceCream** class. The *cost* of a Sundae is the *cost of the IceCream* plus the *cost of the topping*.

**package** Assignment5;

**public** **class** Sundae **extends** IceCream {

**protected** String topName;

**protected** **int** topCost;

**public** Sundae(String \_n0, **int** \_cost0, String \_n1, **int** \_cost1) {

// put the icecream name in icecream while putting top name and cost in

// a separate property

**super**(\_n0, \_cost0);

topName = \_n1;

topCost = \_cost1;

}

**public** **final** String getName() {

// return both the icecream name and the topping name

**return** name + " " + topName;

}

**public** **int** getCost() {

// return the sum of the icecream and the topping

**return** cost + topCost;

}}

**The Checkout Class**

The **Checkout** class, provides methods to enter dessert items into the cash register, clear the cash register, get the number of items, get the total cost of the items (before tax), get the total tax for the items, and get a String representing a receipt for the dessert items. The **Checkout** class must use a Vector to store the DessertItem's. The total tax should be rounded to the nearest cent. The complete specifications for the **Checkout** class are provided in the file Class-Checkout.

**package** Assignment5;

**public** **class** Checkout {

**protected** **int** size;

**protected** DessertItem[] dessertItems;

**protected** **int** amount;

**protected** **int** sum;

**protected** **final** **double** taxRate;

Checkout(){

size = 100;

dessertItems = **new** DessertItem[size];

amount = 0;

sum = 0;

taxRate = DessertShoppe.***TAX\_RATE***;

}

**public** **void** enterItem(DessertItem d){

dessertItems[amount] = d;

amount ++;

}

**public** **int** numberOfItems(){

**return** amount;

}

**public** **int** totalCost(){

//make sum into zero, and calculate price from every item

sum = 0;

**for**(**int** i = 0; i < amount; i ++){

sum += dessertItems[i].getCost();

}

**return** sum;

}

**public** **int** totalTax(){

//use the totalCost method

**return** (**int**)(Math.*round*(**this**.totalCost() \* taxRate / 100));

}

**public** **void** clear(){

//clear the array

**for**(DessertItem d : dessertItems){

d = **null**;

}

amount = 0;

sum = 0;

}

//override the toString method in java object

//this is for printing the object itself out

**public** String toString(){

String result = "Thank You! \n";

result += DessertShoppe.***STORE\_NAME*** + "\n";

result += "Purchased: ";

String totalPay = DessertShoppe.*cents2dollarsAndCents*( totalCost()+totalTax() );

**if**(totalPay.length() > DessertShoppe.***COST\_WIDTH***){

totalPay = totalPay.substring(0, DessertShoppe.***COST\_WIDTH***);

}

result += "$" + totalPay;

**return** result;

}}

## Testing

A simple testdriver, TestCheckout.java along with its expected output, are provided for you to test your class implementations. You can add additional tests to the driver to more thoroughly test your code.

**package** Assignment5;

**public** **class** TestCheckout {

**public** **static** **void** main(String[] args) {

Checkout checkout = **new** Checkout();

checkout.enterItem(**new** Candy("Peanut Butter Fudge", 2.55, 399));

checkout.enterItem(**new** IceCream("Vanilla Ice Cream",110));

checkout.enterItem(**new** Sundae("Chocolate Chip Ice Cream",145, "Hot Fudge", 150));

checkout.enterItem(**new** Cookie("Oatmeal Raisin Cookies", 4, 399));

System.***out***.println("\nNumber of items: " + checkout.numberOfItems() + "\n");

System.***out***.println("\nTotal cost: " + checkout.totalCost() + "\n");

System.***out***.println("\nTotal tax: " + checkout.totalTax() + "\n");

System.***out***.println("\nCost + Tax: " + (checkout.totalCost() + checkout.totalTax()) + "\n");

System.***out***.println(checkout);

checkout.clear();

checkout.enterItem(**new** IceCream("Strawberry Ice Cream",150));

checkout.enterItem(**new** Sundae("Vanilla Ice Cream",105, "Caramel", 50));

checkout.enterItem(**new** Candy("Gummy Worms", 1.33, 89));

checkout.enterItem(**new** Cookie("Chocolate Chip Cookies", 4, 399));

checkout.enterItem(**new** Candy("Salt Water Taffy", 1.5, 209));

checkout.enterItem(**new** Candy("Candy Corn",3.0, 109));

System.***out***.println("\nNumber of items: " + checkout.numberOfItems() + "\n");

System.***out***.println("\nTotal cost: " + checkout.totalCost() + "\n");

System.***out***.println("\nTotal tax: " + checkout.totalTax() + "\n");

System.***out***.println("\nCost + Tax: " + (checkout.totalCost() + checkout.totalTax()) + "\n");

System.***out***.println(checkout);

checkout.clear();

checkout.enterItem(**new** Candy("Whatever Candy", 2.30, 89));

checkout.enterItem(**new** Cookie("nono Cookie", 4, 399));

System.***out***.println("\nNumber of items: " + checkout.numberOfItems() + "\n");

System.***out***.println("\nTotal cost: " + checkout.totalCost() + "\n");

System.***out***.println("\nTotal tax: " + checkout.totalTax() + "\n");

System.***out***.println("\nCost + Tax: " + (checkout.totalCost() + checkout.totalTax()) + "\n");

System.***out***.println(checkout);

}}

**OUTPUT:**

Number of items: 4

Total cost: 1555

Total tax: 101

Cost + Tax: 1656

Thank You!

M & M Dessert Shoppe

Purchased: $16.56

Number of items: 6

Total cost: 1197

Total tax: 78

Cost + Tax: 1275

Thank You!

M & M Dessert Shoppe

Purchased: $12.75

Number of items: 2

Total cost: 338

Total tax: 22

Cost + Tax: 360

Thank You!

M & M Dessert Shoppe

Purchased: $3.60

**EXTRA CREDIT:**

1. Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one ascending sorted array.

You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively.

**ANSWER:**

**package** Assignment5;

**import** java.util.Arrays;

**public** **class** Merge {

**public** **void** merge(**int**[] nums1, **int** m, **int**[] nums2, **int** n) {

**int** i = 0;

**if** (n == 0)

**return**;

**while** (i != m) {

**if** (nums1[i] > nums2[0]) {

**int** t = nums1[i];

nums1[i] = nums2[0];

nums2[0] = t;

}

**int** j = 0;

**while** (j < n - 1 && nums2[j] > nums2[j + 1]) {

**int** t = nums2[j];

nums2[j] = nums2[j + 1];

nums2[j + 1] = t;

j++;

}

i++;

}

**int** j = 0;

**while** (j < n) {

nums1[i++] = nums2[j++];

}

}

**public** **static** **void** main(String[] args)

{

**int**[][] merge = {{8,9},{2,2}};

System.***out***.println(Arrays.*toString*(merge));

}

}

2.Given a matrix of *m* x *n* elements (*m* rows, *n* columns), return all elements of the matrix in spiral order.

For example,  
Given the following matrix:

{{1,2,3},

{4,5,6},

{7,8,9}}

You should return {1,2,3,6,9,8,7,4,5}.

**ANSWER:**

package Assignment5;

import java.util.ArrayList;

import java.util.Arrays;

public class TestSpiralOrder {

public static void main(String[] args)

{

int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

ArrayList<Integer> spiralOrder = spiralOrder(matrix);

System.out.println(Arrays.toString(spiralOrder.toArray()));

}

public static ArrayList<Integer> spiralOrder(int[][] matrix) {

ArrayList<Integer> spiralOrder

= new ArrayList<Integer>();

if(matrix == null) { return null; }

if(matrix.length == 0) { return spiralOrder; }

if(matrix[0].length == 0) { return spiralOrder; }

int midx = (matrix.length - 1) / 2;

int midy = (matrix[0].length - 1) / 2;

int x = 0;

int y = 0;

while(x <= midx && y <= midy) {

int rowLB = x;

int rowUB = matrix.length - 1 - x;

int colLB = y;

int colUB = matrix[0].length - 1 - y;

int i = rowLB;

int j = colLB;

for(; j <= colUB; j++) {

spiralOrder.add(matrix[i][j]);

}

j--;

i++;

for(; i <= rowUB; i++) {

spiralOrder.add(matrix[i][j]);

}

if(rowLB == rowUB) {

x++;

y++;

continue;

}

i--;

j--;

for(; j >= colLB; j--) {

spiralOrder.add(matrix[i][j]);

}

j++;

i--;

if(colLB == colUB) {

x++;

y++;

continue;

}

for(; i > rowLB; i--) {

spiralOrder.add(matrix[i][j]);

}

x++;

y++;

}

return spiralOrder;

}

}