Practice Final Answers

Trace

Write the output of the following program fragments: OUTPUT:

1.
```
System.out.println(“Three toed tree toads”);
System.out.println(“Goodbye”);
```
```
Three toed tree toads
Goodbye
```

2.
```
System.out.print(“You don’t”);
System.out.println(“ say!”);
```
```
You don’t say!
```

3.
```
System.out.print(“H\na”);
```
```
H
a
```

4.
```
System.out.print(“\”Ha\””);
```
```
"Ha"
```

5.
```
if(false || true){
  System.out.println(“XoX”);
}

System.out.println(“Got it”);
```
```
XoX
Got it
```

6.

```
if(true && false){                              B
  System.out.println(“A”);
} else{
  System.out.println(“B”);
}
```

7.

```
if(true || false){                              A
   System.out.println(“A”);                     B
}

if(!false){
  System.out.println(“B”);
}
```

8.   H
```
if(7 < 9 && true){                                    A
   System.out.println(“H”);
   if(!(false || false)){
     System.out.println(“A”);
   } else{
      System.out.println(“B”);
   }
} else {
   System.out.println(“Q”);
}
```

9.   Y
```
if(10 != 10){
  System.out.println(“X”);
} else if (6 > 2){
  System.out.println(“Y”);
} else{
  System.out.println(“Z”);
}
```

10.   3
```
String day = “Wed”;                                   4

switch(day){
  case “Mon”:
    System.out.println(“1”);
    break;
   case “Tues”:
     System.out.println(“2”);
     break;
    case “Wed”:
      System.out.println(“3”);
    case “Thurs”:
       System.out.println(“4”);
}
```

```
11.
int a = 3;                                          4
int b = a;                                          3

a = a + 1;

System.out.println(a);
System.out.println(b);
```

```
12.                                                 4
Square a = new Square();                            4
a.setLength(3);

Square b = a;

b.setLength(4);

System.out.println(a.getLength());
System.out.println(b.getLength());
```

```
13.                                                 4
int x = 3;
x++;
System.out.println(x);
```

```
14.                                                 2
int y = 3;
y--;
System.out.println(y);
```

15.

```
int z = 3;
z += 2;
System.out.println(z);
```

5

16.

```
int m = 7;
int n = 2;

System.out.println(m / n);
```

3

17.

```
int p = 13;
int q = 2;

System.out.println(p % q);
```

1

18.

```
double r =  5;
double s = 2;

System.out.println(r / s);
```

2.5

19.

```
int result = Math.pow(5,3);
System.out.println(result);
```

125

20.

```
int max = Integer.MAX_VALUE; //2147483647, the largest value of type int
int min = Integer.MIN_VALUE; //-2147483648, the smallest value of type int

max++;

System.out.println(max);
```

-2147483648

21.

```
char c = 66;  //the letter B
c++;

System.out.println(c);
```

C

22.

```
int x = 3 + 2 * 4 / 8;
System.out.println(x);
```

4

23.

```
int y = 3 * (2 + 1) * 5
System.out.println(y);
```

45

24.

```
String s = “1”;
String t = “2”;
String u = “3”;

System.out.println(s + t + u);
```

123

25.
```
String str = “  FasT  “;
String str2 = “Car”;

System.out.println(str.toLowerCase().trim()
+ str2);
```

```
fastCar
```

26.
```
int x = 2;
int y = 5;

System.out.println((double) y / (double) x);
```

```
2.5
```

27.
```
String s = “36”;
String t = “6”;

System.out.println(Integer.parseInt(s) + Integer.parseInt(t));
```

```
42
```

28.
```
String a = “”;

System.out.println(“b4”);

do{
System.out.println(“inside”);
a += “a”;
}while (a != “a”)

System.out.println(“after”);
```

```
//<INFINITE LOOP> the condition should have
been while(!a.equals(“a”));

//With that condition that output would have
been:

b4
inside
inside
after
```

29.
```
String a = “”;

System.out.println(“b4”);

while(a == “a”){
  System.out.println(“inside”);
}

System.out.println(“after”);
```

```
b4
after
```

30.
```
String a = “”;

System.out.println(“b4”);

for(int i=0;i < 3; i++){
  for(int j=0;j < 2; j++){
    a += “a”;
  }
}

System.out.println(a);
```

```
b4
aaaaaa
```

31.
```
String a = “”;

System.out.println(“b4”);

int w = 1;

while(a != “aaaa”){
  System.out.println(“inside” + w);

  for(int i=0; i<2; i++){
    System.out.println(“w“ + w + “ loop
iteration “ + i);
    a += “a”;
  }

  w++;
}

System.out.println(“finished”);
```

```
//<INFINITE LOOP>
//if you change the condition to:
//while(!a.equals(“aaaa”)) then
//the output is:

b4
inside1
w1 loop iteration 0
w1 loop iteration 1
inside2
w2 loop iteration 0
w2 loop iteration 1
finished
```

32. (3pts)   5

```
public class Trace32 {                                 6
  public static void main(String [] args){
    Trace32 n = new Trace32(5);
    System.out.println(n.getXxx());
    n.increment();
    System.out.println(n.toString());
  }

  public Trace32(int q){
    xxx = q;
  }

  public int getXxx(){
    return xxx;
  }

  public void increment(){
    xxx++;
  }

  public String toString(){
    return Integer.toString(getXxx());
  }

  private int xxx;
}
```

33. (3pts)

```
twosocks
two_socks
```

```
public class Trace33 {
  public static void main(String [] args){
    String t = "two";
    String s = "socks";
    String u = "_";

    System.out.println(Trace33.concat(t,
s));

    Trace33 t33 = new Trace33(t, u);

    System.out.println(t33.concat(s));
  }

  public Trace33(String x, String y){
   str = x;
   separator = y;
  }

  public static String concat(String a,
String b){
    return a + b;
  }

  public Stringconcat(String a){
   return this + separator + a;
  }

  public String toString(){
    return str;
  }

  private String str;
  private String separator;
}
```

34. (3pts)

```
public class Trace34{
  public static void main(String [] args){
    System.out.println(factorial(5));
  }

  public static int factorial(int n)
  {
    if (n <= 1){
        System.out.println(n);
        return 1;
    }else{
        System.out.println(n);
        return n*factorial(n-1);
    }
  }
}
```

```
5
4
3
2
1
120
```

35. (4pts)

```
public class Trace35{
  public static void main(String [] args){
    String abc = "abc";
    char a = 'a';

    System.out.println(f1(f2(abc), 3));
    System.out.println(f2(f1(abc, 3)));

    System.out.println(f1(f2(a), 3));
    System.out.println(f2(f1(a,3)));
  }

  public static String f1(char c, int n){
   String result = Character.toString(c);

   for(int i = 0; i < n; i++){
     result += Character.toString(c);
   }

   return result;
  }

  public static String f1(String c, int n){
    String result = c;

    for(int i = 0; i < n; i++){
      result += c;
    }

    return result;
  }

  public static String f2(String s){
   return s + s;
  }

  public static String f2(char c){
    return Character.toString(c) + Character.toString(c);
  }

}
```

```
abcabcabcabcabcabcabcabc
abcabcabcabcabcabcabcabc
aaaaaaaa
aaaaaaaa
```

Recognize Elements

Use the code provided as a separate hand-out for the cash register to answer the questions below.

1. List all of the classes included in this program.

Business, Product, ProductList, Register, RegisterList, Transaction, TransactionList, Util

2. Which class has a main function?

Business

3. What is the return type of the method getReceipt in the Register class?

String

4. Name the public methods (not including constructors) are in the Register class?

calculateTotal, scanProduct, acceptPayment, completeCheckout, startNewCheckout, getReceipt, getChange, toString

5. What are the signatures of the two Register constructors?

Register(int, ProductList, double)
Register(int, ProductList, double, double)

6. What are the parameters of the scanProduct() method in the Register class?

int id

7. Name a class and a span of lines (the numbers are to the left) where a while loop is being used.

Register, lines 11-22

8. Name a function and the class it belongs to that is responsible for saving data to a file.

SaveTransactionList in the class TransactionList

9. Name a class and a span of lines where a file is being read from.

ProductList, lines 11-21

10. Name all the member variables of the Transaction class.

momentOfPurchase;
registerId;
amountPaid;
changeGiven;
subTotal;
taxPaid;

Conceptual Understanding

Use the cash register provided separately to answer the conceptual questions below:

1. In the Util classes, what does it mean for the three variables owned by that class to be declared as public, static, and final?

Public means the variables can be accessed outside the class by other code/programmers. Static means the class owns the variable data instead of an object. Final means the variable's value cannot be changed and will remain the same for the entire program.

2. The cash register classes generate some of their objects by reading data from a file and then save new data to those files before the program closes. Why do they do this? What feature does this provide over a program that does not read and write to a file?

When a program runs, it's data is stored in RAM. The data remains there for as long as the program is running, but as soon as the program stops running, all of the data is freed for other programs to write over and is basically lost. By writing data to a file and retrieving that data when the program starts up, the program can remember what happened the last time the program ran and continue where it left off. In the case of the Register class, a register can remember how much money it has made and update itself as more checkout transactions occur.

3. The cash register classes make extensive use of arrays, Files, and Dates and Timestamps. Most public users of this code don’t see the arrays and other structures when they use the API. The implementation is hidden. What is the word for this idea of hiding the inner-workings and restricting data access while programming using an object-oriented design?

This is called the concept of encapsulation and can be used to ensure data manipulation occurs as the programmer intended even if another programmer cannot discuss how a class should be used with the original programmer. All the new programmer needs to know is what public methods exist and what their inputs and outputs are and possibly what they are supposed to be used for. Anything that happens inside the function can be safely ignored as long as the program is well-designed and properly encapsulated.

4. Notice that each class has its own toString() method that overrides the default implementation. What are some reasons you would want to override the default toString() method for your class?

The default implementation simply returns a hash id of an object which is frequently not what other programmers will have in mind when they are wanting to represent the object as output. By writing a new toString method, programmers are able to represent their object in a way that is meaningful for their programmer or other coders that need to see the state of an object.

5. Notice that the Transaction class has two constructors with different parameters sets. One is called when loading Transactions from a file and the other is called when scanning a product during a checkout session. Why do each of these use cases use different constructors?

The first constructor without the Timestamp parameter is for creating a new Transaction that hasn't existed before. The second constructor with the Timestamp parameter is for regenerating a Transaction that has been saved to a file. When a Transaction is loaded from a file, it already has a Timestamp from back when it was originally created. The second constructor provides the programmer with the option to use the timestamp stored in a file to retain the original date the Transaction was first created. The first constructor, without the parameter, provides the programmer the convenience of not having to generate a new Timestamp when a Transaction is created for the first time. This process is encapsulated and hidden from the programmer.

Use an Existing API

Pretend you are inside a new main method. Write code for this main method that will execute what is described.

1. Make a new ProductList and fill it with 3 products. Use this new ProductList to create a new Register with a registerId of value 1 and a tax rate of 4%.

```
ProductList pL = new ProductList();

pL.insertProduct(new Product(“apple”, 2.90));
pL.insertProduct(new Product(“banana”, 3.00));
pL.insertProduct(new Product(“carrot”, 2.00));

Register r = new Register(1, pL, 0.04);
```

2. Use the Util class provided to convert the String, “2019-11-27 16:09:15.018” into an object of type Timestamp.

```
Util.convertStringToTimestamp(“2019-11-27 16:09:15.018”);
```

3. Print the public consant (final variable) named CSV_START_LINE in the Util class to the screen.

```
System.out.println(Util.CSV_START_LINE);
```

4. Take the register you created in question 1 and scan the product with id 2 three times. After this, accept payment with $1024.25 as input, complete the checkout, get the amount of change due, print a receipt, and then start a new checkout session.

```
for(int i = 0; i < 3; i++){
  r.scanProduct(2);
}

r.acceptPayment(1024.25);
r.getChange();
r.completeCheckout();
System.out.println(r.getReceipt());
r.startNewCheckout();
```

5. **EXTRA CREDIT** Create an array of 100 cash registers. Generate your product list from a file named key-foods-products.csv. Make all 100 cash registers scan each item in the product list exactly one time. Then, accept payment, complete each of their checkouts, get the change due, and print a receipt. Make sure you start a new checkout session for each of the 100 registers after each receipt is printed.

```
Register [] registers = new Register[100];
ProductList ps = new ProductList("key-foods-products.csv");

for(int i=0; i < 100; i++){
  registers[i] = new Register(i, ps, 0.04);
}

for(int i=0; i < ps.length(); i++){
  for(int j=0; j < registers.length; j++){
     registers[j].scanProduct(i);
  }
}

for(int j=0; j < registers.length; j++){
   registers[j].acceptPayment(1000);
   registers[j].completeCheckout();
   registers[j].getChange();
   System.out.println(registers[j].getReceipt());
   registers[j].startNewCheckout();
}
```

Design a Program

Write an object-oriented model of a vending machine that dispenses candy. There are 4 different kinds of candy available inside the machine. Each kind of candy is held in a shelf slot that can hold 3 copies of the candy before that row of the vending machine must be refilled. A customer selects the candy by choosing the row (A or B) and the column (one or two). The available candies are named snickers ($1.00) , skittles ($1.25), twix ($0.75), and pretzels ($0.85). When the machine starts operating it is filled with 4 of each kind of candy. Every time a customer buys a candy, there is one less of that candy until there are none left at which point a customer who requests that candy will be denied sale and be informed that there are none left.

After a customer selects the candy they want. The vending machine requests payment. The customer then must pay the amount. If the customer pays money that is greater than or equal to the amount due, the machine dispenses the selected candy and gives the customer the change they are due. The machine keeps running until it is out of candy at which point it requests maintenance and says good by and the main() of our program stops execution.

Your program should have a class VendingMachine and a class Candy

The Candy class should have private member variables: name, price, column, row, and itemsRemaining (which will represent how many of that type of candy is left in the machine).

The VendingMachine class will have four private member variables of type Candy and one private member variable of type double named moneyMade that keeps track of how much money the machine has made.

The VendingMachine will have a constructor that takes no parameters and initializes the four member variables so that each candy has the appropriate features mentioned above. If there is a file called vending-machine.txt, the amount of each candy should be loaded from that file along with how much money the machine has made. If that file does not exist, the machine should start with zero dollars and be fully stocked with candy.

The VendingMachine will have a private method named acceptRequest(String column, String row) which will return true if that location in the machine has any candy left and false if it does not.

The VendingMachine will also have a public method named processPayment(String column, String row, double amountPaid) which will call acceptRequest(). If acceptRequest returns true, the processPayment() function will decrement the itemsRemaining of the selected candy, add the cost of the candy to the variable moneyMade and then return the amountPaid minus the cost of the candy purchased.

The VendingMachine will have a public method getCandyInfo(String column, String row) that will return the name, price, and itemsRemaining as a single String.

The VendingMachine will have a public fillMethod that will replenish the itemsRemaining for each candy in the vending machine to full capacity.

Before the program ends, it should save the state of the vending machine in the file named vending-machine.txt. Your program should be able to read from this file when you restart the program to retrieve how many items of candy and how much money is in the machine.

The Candy class should of course have getters and setters for its private member variables.

If you would like extra credit and desire to write less code, feel free to design the program using arrays.

```
public class Candy{

  public Candy(String name, double price, String row, String column){
    this.name = name;
    this.price = price;
    this.row = row;
    this.column = column;
    this.itemsRemaining = 3;
  }

  public void setItemsRemaining(int x){
    itemsRemaining = x;
  }

  public void reduceItemsRemaining(){
    itemsRemaining--;
  }

  public int itemCount(){
    return itemsRemaining;
  }

  public String getRow(){
    return row;
  }

  public String getColumn(){
    return column;
  }

  public double getCost(){
    return price;
  }

  public boolean isMatch(String row, String column){
    return this.row.equals(row) && this.column.equals(column);
  }

  public boolean hasItemsRemaining(){
    return itemsRemaining > 0;
  }

  public void fill(){
    itemsRemaining = MAX;
  }

  public String toString(){
    return name + " " + price + " " + itemsRemaining + " remaining.";
  }

  public static final int MAX = 3;
  private String name;
  private double price;
  private String row;
  private String column;
  private int itemsRemaining;
}
import java.io.File;
import java.util.Scanner;
import java.io.IOException;
import java.io.PrintWriter;


public class VendingMachine{
  public static void main(String[] args) throws IOException{
   VendingMachine v = new VendingMachine();

   v.getCandyInfo("A", "1");
   v.processPayment("B", "1", 10.00);

   v.saveVendingMachine();

  }
public VendingMachine() throws IOException{
    File f = new File(FILENAME);
    c1 = new Candy("snickers", 1.00, "A", "1");
    c2 = new Candy("skittles", 1.25, "A", "2");
    c3 = new Candy("twix", 0.75, "B", "1");
    c4 = new Candy("pretzels", 0.85, "B", "2");

    if (f.exists()){
      Scanner s = new Scanner(f);

      moneyMade = Double.parseDouble(s.nextLine());

      c1.setItemsRemaining(Integer.parseInt(s.nextLine()));
      c2.setItemsRemaining(Integer.parseInt(s.nextLine()));
      c3.setItemsRemaining(Integer.parseInt(s.nextLine()));
      c4.setItemsRemaining(Integer.parseInt(s.nextLine()));
    }
    else{
      moneyMade = 0.00;
    }
  }

  public double processPayment(String row, String column, double amountPaid){
    Candy c = getCandy(row, column);
    double returnAmount = 0.00;

    if(acceptRequest(row, column)){
      if(amountPaid > c.getCost()){
        c.reduceItemsRemaining();
        moneyMade += c.getCost();
        returnAmount = amountPaid - c.getCost();
      } else{
        System.out.println("Insufficient Funds. Add more cash.");
          returnAmount = amountPaid;
      }
    } else{
      System.out.println("There is none of that candy left");
      returnAmount = amountPaid;
    }

    return returnAmount;
  }

  public String getCandyInfo(String row, String column){
    return getCandy(row, column).toString();
  }

  public void fill(){
    c1.fill();
    c2.fill();
    c3.fill();
    c4.fill();
  }

  public void saveVendingMachine() throws IOException{
    File f = new File(FILENAME);
    PrintWriter x = new PrintWriter(f);
    x.print(this.moneyMade + "\n");

    x.print(this.c1.itemCount() + "\n");
    x.print(this.c2.itemCount() + "\n");
    x.print(this.c3.itemCount() + "\n");
    x.print(this.c4.itemCount() + "\n");
    x.close();
  }

  private boolean acceptRequest(String row, String column){
    return getCandy(row, column).hasItemsRemaining();
  }

  private Candy getCandy(String row, String column){
    Candy c = null;

    if(c1.isMatch(row, column)){
      c = c1;
    }else if(c2.isMatch(row, column)){
      c = c2;
    }else if(c3.isMatch(row, column)){
      c = c3;
    }else if(c4.isMatch(row, column)){
      c = c4;
    }
    return c;
  }

  public static final String FILENAME = "vending-machine.txt";
  private Candy c1;
  private Candy c2;
  private Candy c3;
  private Candy c4;
  private double moneyMade;
}
```

Practice Final Code

CSC110 Fall 2019

```
import java.io.IOException;

public class Business{
  public static void main(String [] args) throws IOException{

      ProductList pList = new ProductList("products.csv");
      TransactionList tList = new TransactionList("transactions.csv");
      RegisterList rList = new RegisterList("registers.csv", pList);

      System.out.println(pList);
      System.out.println(tList);
      System.out.println(rList);

      rList.getRegister(0).scanProduct(0);
      rList.getRegister(0).scanProduct(1);
      rList.getRegister(0).scanProduct(0);

      rList.getRegister(1).scanProduct(2);
      rList.getRegister(1).scanProduct(2);

      rList.getRegister(2).scanProduct(2);
      rList.getRegister(2).scanProduct(0);

      rList.getRegister(0).acceptPayment(100);
      rList.getRegister(1).acceptPayment(20);
      rList.getRegister(2).acceptPayment(35.75);

      for(int i = 0; i < rList.length(); i++){
        tList.insertTransaction(rList.getRegister(i).completeCheckout());
          System.out.println("Your transaction at register " + i + " is complete!");
          System.out.println("Your change is: " + rList.getRegister(i).getChange());
          System.out.println(rList.getRegister(i).getReceipt());

          System.out.println("Starting new checkout session...");
          rList.getRegister(i).startNewCheckout();
      }

      tList.saveTransactionList("transaction.csv");
      rList.saveRegisterList("registers.csv");
  }
}
```

```java
public class Product{
  public Product(String name, double price){
    this.name = name;
    this.price = price;
  }

  public String getName(){
    return name;
  }

  public double getPrice(){
    return price;
  }

  public String toString(){
    return "\"" + name + "\", '\"" + price + "\"\n";
  }

  private String name;
  private double price;
  private int productId;
}
```

```java
import java.io.File;
import java.util.Scanner;
import java.io.IOException;
import java.io.PrintWriter;

public class ProductList{
  public ProductList(String filename) throws IOException {
    Scanner scan = new Scanner(new File(filename));
    products = new Product[1000];
    this.length = 0;

    int i = 0;

    while(scan.hasNext()){
      String line = scan.nextLine();

      if(i != 0){
        String [] fields = line.split(",");

        String nameField = fields[0].replace("\"", "").trim();
        double priceField = Double.parseDouble(fields[1].replace("\"", "").trim());

        products[i - 1] = new Product(nameField, priceField);
      }
      i++;
    }

    if(i > 0){
      this.length = i - 1;
    }

  }

  public ProductList(){
   products = new Product[MAX_PRODUCTS];
   length = 0;
  }


  public ProductList(Product [] list){
    products = list;
    this.length = list.length;
  }

  public Product getProduct(int index){
    return products[index];
  }

  public double getPriceById(int id){
    return products[id].getPrice();
  }

  public String getNameById(int id){
    return products[id].getName();
  }

  public int length(){
    return length;
  }

  public void insertProduct(Product p){
    products[length] = p;
    length++;
  }

  public String toString(){
    String result = PRODUCTS_HEADER;

    for(int i = 0; i < length; i++){
      result += products[i];
    }

    return result;
  }

  public void saveProductList(String filename) throws IOException{
    File f = new File(filename);
    PrintWriter x = new PrintWriter(f);
    x.print(this);
    x.close();
  }

  public static final String CSV_START_LINE = "\"";
  public static final String CSV_SEPARATOR = "\", \"";
  public static final String CSV_END_LINE = "\"\n";
  public static final String PRODUCTS_HEADER = CSV_START_LINE +
                                               "Name" + CSV_SEPARATOR +
                                                      "Price" + CSV_END_LINE;
  public static final int MAX_PRODUCTS = 1000;
  private Product [] products;
  private int length;
}
```

```java
public class Register{
  public Register(int registerId, ProductList products, double salesTax){
    this.registerId = registerId;
    this.products = products;
    this.salesTaxRate = salesTax;
    this.scannedProductIds = new int[MAX_PRODUCT_LIST_SIZE];
    this.indexOfNextProduct = 0;
    this.receipt = "\n\nReceipt\n\n";
    this.moneyInRegister = 0;
  }

  public Register(int registerId, ProductList products, double salesTax, double moneyInRegister){
    this(registerId, products, salesTax);
    this.moneyInRegister = moneyInRegister;
  }


  public double calculateTotal(){
    return salesTaxDue + subTotalDue;
  }

  public void scanProduct(int id){
    scannedProductIds[indexOfNextProduct] = id;
    indexOfNextProduct++;

    double price = products.getPriceById(id);
    Product product = products.getProduct(id);

    subTotalDue += price;
    salesTaxDue += price * salesTaxRate;

    receipt += product;

  }

  public void acceptPayment(double amount){
    this.paymentAmount = amount;
  }

  public Transaction completeCheckout(){
    Transaction t = new Transaction(registerId, paymentAmount, subTotalDue, salesTaxDue, getChange());
    moneyInRegister += (paymentAmount - getChange());

    receipt += ("\nSubTotal: " + subTotalDue + "\n");
    receipt += ("Tax at " + salesTaxRate + "% :" + salesTaxDue + "\n");
    receipt += ("Total: " + calculateTotal() + "\n\n");
    receipt += "Thank You! Come again!\n\n";

    return t;
  }

  public void startNewCheckout(){
    clearCurrentCheckout();
  }


  private void clearCurrentCheckout(){
    scannedProductIds = new int[MAX_PRODUCT_LIST_SIZE];
    indexOfNextProduct = 0;
    salesTaxDue = 0;
    subTotalDue = 0;
    paymentAmount = 0;
    receipt = "\n\nReceipt\n\n";
  }

  public String getReceipt(){
    return receipt;
  }

  public double getChange(){
    return paymentAmount - calculateTotal();
  }

  public String toString(){
    return Util.CSV_START_LINE + registerId + Util.CSV_SEPARATOR + salesTaxRate + Util.CSV_SEPARATOR + moneyInRegister +
Util.CSV_END_LINE;
  }

  public static final int MAX_PRODUCT_LIST_SIZE = 1000;

  private ProductList products;
  private int [] scannedProductIds;
  private int registerId;
  private double moneyInRegister;
  private double salesTaxRate;
  private int indexOfNextProduct;

  private double salesTaxDue;
  private double subTotalDue;
  private double paymentAmount;
  private String receipt;
}
```

```
import java.io.File;
import java.util.Scanner;
import java.io.IOException;
import java.io.PrintWriter;

public class RegisterList{

  public RegisterList(String filename, ProductList pList) throws IOException{
    Scanner scan = new Scanner(new File(filename));
    registers = new Register[MAX_REGISTERS];
    this.length = 0;

    int i = 0;

    while(scan.hasNext()){
      String line = scan.nextLine();

      if(i !=0){
        String [] fields  = line.split(",");

          int registerId = Util.cleanFieldToInt(fields[0]);
        double salesTaxRate = Util.cleanFieldToDouble(fields[1]);
          double moneyInRegister = Util.cleanFieldToDouble(fields[2]);

          registers[i - 1] = new Register(registerId, pList, salesTaxRate, moneyInRegister);
      }

      i++;
    }

    if(i > 0){
     this.length = i - 1;
    }
  }

  public void insertRegister(Register r){
    registers[length] = r;
    length++;
  }

  public Register getRegister(int index){
    return registers[index];
  }


  public String toString(){
    String result = REGISTERS_HEADER;

    for(int i = 0; i < length; i++){
      result += registers[i];
    }

    return result;
  }

  public int length(){
    return length;
  }

  public void saveRegisterList(String filename) throws IOException{
    File f = new File(filename);
    PrintWriter x = new PrintWriter(f);
    x.print(this);
    x.close();
  }

  private Register [] registers;
  private int length;

  public static final int MAX_REGISTERS = 100;
  public static final String REGISTERS_HEADER = Util.CSV_START_LINE + "Register Id" + Util.CSV_SEPARATOR + "Sales Tax Rate" +
Util.CSV_SEPARATOR + "Money in Register" + Util.CSV_END_LINE;
}
```

```java
import java.sql.Timestamp;
import java.time.Instant;

public class Transaction{

  public Transaction(int registerId, double amountPaid, double subTotal, double taxPaid, double changeGiven){
    this.registerId = registerId;
    this.momentOfPurchase = new Timestamp(System.currentTimeMillis());
    this.amountPaid = amountPaid;
    this.subTotal = subTotal;
    this.taxPaid = taxPaid;
    this.changeGiven = changeGiven;
  }

  public Transaction(Timestamp t, int registerId, double amountPaid, double subTotal, double taxPaid, double changeGiven){
    this(registerId, amountPaid, subTotal, taxPaid, changeGiven);
    momentOfPurchase = t;
  }

  public Transaction(String transactionStr){
    //break apart the string and construct the object
  }

  public int getRegisterId(){
    return registerId;
  }

  public String getTimestamp(){
    return momentOfPurchase.toString();
  }

  public double getAmountPaid(){
    return amountPaid;
  }

  public double getSubTotal(){
    return subTotal;
  }

  public double getTaxPaid(){
    return taxPaid;
  }


  public String toString(){
    return CSV_START_LINE + getTimestamp() + CSV_SEPARATOR +
                              registerId + CSV_SEPARATOR +
                                 subTotal + CSV_SEPARATOR +
                                 taxPaid + CSV_SEPARATOR +
                                 amountPaid + CSV_SEPARATOR +
                                 changeGiven + CSV_END_LINE;
  }

  public static final String CSV_START_LINE = "\"";
  public static final String CSV_SEPARATOR = "\", \"";
  public static final String CSV_END_LINE = "\"\n";
  public static final String TRANSACTION_CSV_HEADING = CSV_START_LINE +
                                                     "Moment of Purchase" + CSV_SEPARATOR +
                                                              "Register ID" + CSV_SEPARATOR +
                                                              "SubTotal" + CSV_SEPARATOR +
                                                              "Tax Paid" + CSV_SEPARATOR +
                                                              "Amount Paid" + CSV_SEPARATOR +
                                                              "Change Given" + CSV_END_LINE;


  private Timestamp momentOfPurchase;
  private int registerId;
  private double amountPaid;
  private double changeGiven;
  private double subTotal;
  private double taxPaid;
}
```

```java
import java.io.File;
import java.sql.Timestamp;
import java.util.Scanner;
import java.io.IOException;
import java.io.PrintWriter;

public class TransactionList{
  public TransactionList(String filename) throws IOException {
    Scanner scan = new Scanner(new File(filename));
    transactions = new Transaction[MAX_TRANSACTIONS];
    this.length = 0;

    int i = 0;

    while(scan.hasNext()){
      String line = scan.nextLine();

      if(i != 0){
        String [] fields = line.split(",");

          Timestamp  momentOfPurchase = Util.convertStringToTimestamp(fields[0].replace("\"", ""));
          int registerId = Util.cleanFieldToInt(fields[1]);
          double amountPaid = Util.cleanFieldToDouble(fields[2]);
          double changeGiven = Util.cleanFieldToDouble(fields[3]);
          double subTotal = Util.cleanFieldToDouble(fields[4]);
          double taxPaid = Util.cleanFieldToDouble(fields[5]);

          transactions[i - 1] = new Transaction(momentOfPurchase, registerId, amountPaid, changeGiven, subTotal, taxPaid);

      }

      i++;
    }

    if(i > 0){
      this.length = i - 1;
    }

  }

  public Transaction getTransaction(int index){
    return transactions[index];
  }

  public void insertTransaction(Transaction t){
    transactions[length] = t;
    length++;
  }

  public int length(){
    return length;
  }

  public String toString(){
    String result = Transaction.TRANSACTION_CSV_HEADING;

    for(int i = 0; i < length; i++){
      result += transactions[i];
    }
    return result;

  }

  public void saveTransactionList(String filename) throws IOException{
    File f = new File(filename);
    PrintWriter x = new PrintWriter(f);
    x.print(this);
    x.close();
  }

  public static final int MAX_TRANSACTIONS = 11600;
  private Transaction [] transactions;
  private int length;

}
```

```java
import java.sql.Timestamp;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Util {
  public static Timestamp convertStringToTimestamp(String strDate) {
    try {
      DateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
      Date date = formatter.parse(strDate);
      Timestamp timeStampDate = new Timestamp(date.getTime());

      return timeStampDate;

    } catch (ParseException e) {
      System.out.println("Exception :" + e);

      return null;
    }
  }

  public static double cleanFieldToDouble(String field){
    return Double.parseDouble(field.replace("\"", "").trim());
  }

  public static int cleanFieldToInt(String field){
    return Integer.parseInt(field.replace("\"", "").trim());
  }

  public static String cleanField(String field){
    return field.replace("\"", "").trim();
  }

  public static final String CSV_START_LINE = "\"";
  public static final String CSV_SEPARATOR = "\", \"";
  public static final String CSV_END_LINE = "\"\n";

}
```