Kenneth, Jay, Victor, Jose

Wooyoung Kim

CSS 343, Sec. C

March 1, 2020

**Project #4 Design**

**Overview**

This is the design for a movie rental store that will automate its inventory tracking system. The program initializes the contents of the inventory from a text file, then a customer list from another text file, and processes a sequence of commands from a third text file using information stored about the customers and movies in the system. There are three types of movies that will be tracked: comedy, drama, and classics. Four types of actions are desired by the system: borrow, return, inventory, and history.

**Description of Main**

Main will be responsible for creating a new instance of a Business object which will initialize all the customer and movie data from text files. The business will have one method that will take in a string that relates to the name of the movie file and initialize the content of the stores’ inventory. After this method is called and the inventory is created, the main will call another function that will take in a string that relates to the name of a text file with customer data and initialize the stores’ list customers that will be executing transactions. Once the inventory and customer list are set, the main will then call a method that takes in a string related to the name of a text file with transaction data and then process an arbitrary sequence of commands from that text file. These commands will be stored in a queue, and the next function call of executeTransactions will execute them in order.

int main()

{

Business b;

b.buildMovies(“data4movies.txt”);

b.buildCustomers(“data4customers.txt”);

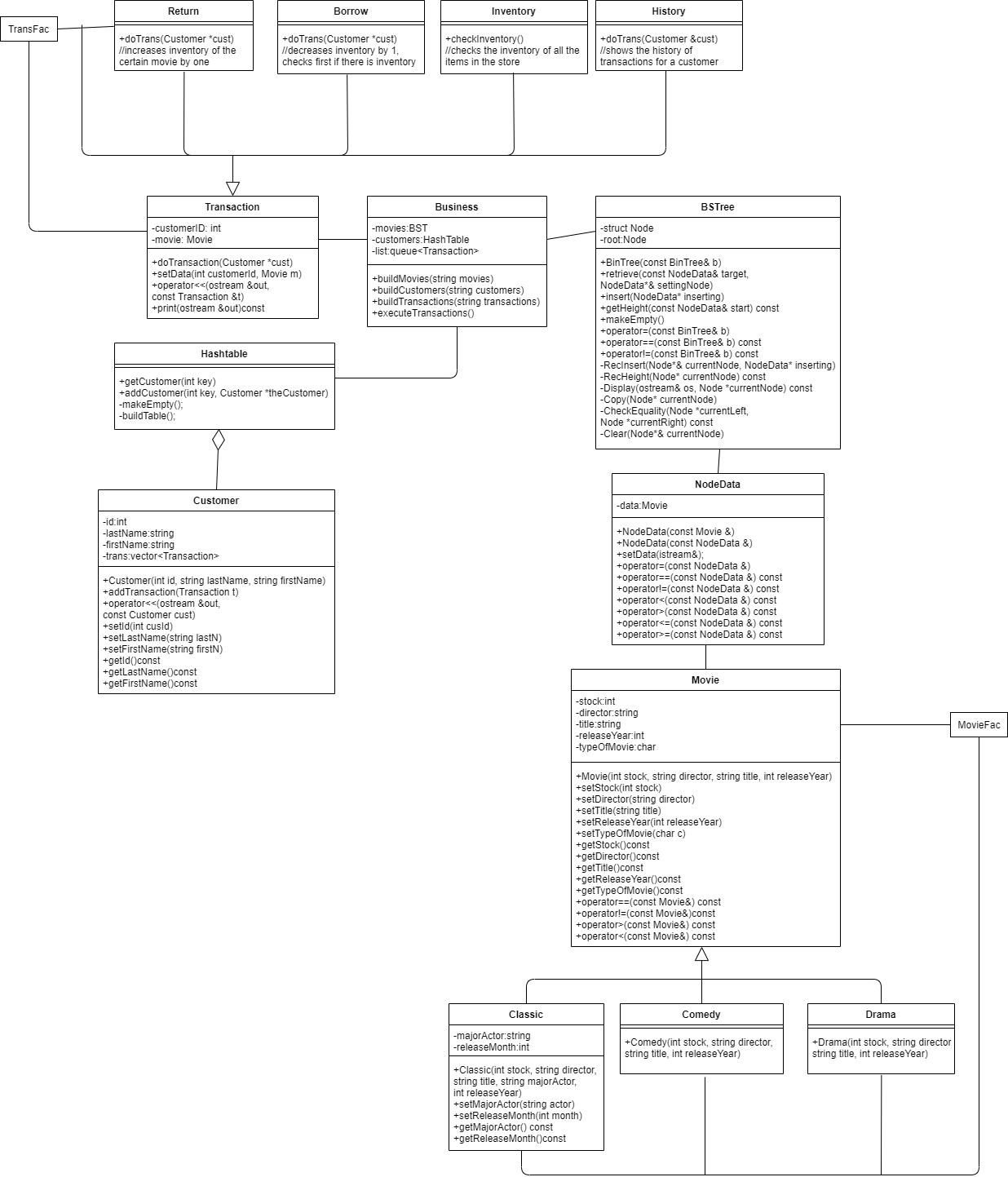
b.buildTransactions(“data4commands.txt”);

b.executeTransactions();

return 0;

};

**Class Diagram**

****

**Class Descriptions**

**Description:**

Represents a Transaction object that contains the information needed to complete a transaction in the movie rental system. It also serves as the superclass that will let other Transaction objects such as Borrow, Inventory, Return, and History, to be derived from.

Class Transaction

{

Public:

Transaction();

~Transaction();

void setData(int customerId, Movie m);

virtual void doTrans(Customer \*cust);

friend ostream& operator<<(ostream &out, const Transaction &t);

virtual void print(ostream &out)const;

private:

int customerId;

Movie movie;

};

**Description:**

Child class of transaction that will contain the implementation for completing the borrow request from the inventory system for a customer.

Class Borrow : public Transaction

{

public:

Borrow();

~Borrow();

void doTrans(Customer \*cust); //decreases inventory by 1, checks first if there is inventory

};

**Description:**

Child class of transaction that will contain the implementation for outputting the inventory of all the items in the store.

Class Inventory : public Transaction

{

public:

Inventory();

~Inventory();

bool checkInventory(); //checks the inventory of all the items in the store

};

**Description:**

Child class of transaction that will contain the implementation for completing the return request to the inventory system for a customer.

Class Return : public Transaction

{

public:

Return();

~Return();

void doTrans(Customer \*cust); //increases inventory of the certain movie by one

};

**Description:**

Child class of transaction that will contain the implementation for outputting all the transaction history for a customer.

Class History : public Transaction

{  
public:

History();

~History();

void doTrans(Customer &cust); //shows the history of transactions for a customer

};

**Description:**

This class represents a movie object that has a preset number of attributes that every movie object will contain. It also serves as the superclass that will let other movie objects such as Comedy, Drama, and Classic, to be derived from.

Class Movie

{

public:

Movie(int stock, string director, string title, int releaseYear);

~Movie();

void setStock(int stock);

void setDirector(string director);

void setTitle(string title);

void setReleaseYear(int releaseYear);

void setTypeOfMovie(char c);

int getStock()const;

string getDirector()const;

string getTitle()const;

int getReleaseYear()const;

char getTypeOfMovie()const;

bool operator==(const Movie&) const;

bool operator!=(const Movie&)const;

bool operator>(const Movie&) const;

bool operator<(const Movie&) const;

private:

int stock;

string director;

string title

int releaseYear;

char typeOfMovie;

};

**Description:**

Child class of Movie that will contain the data and information of a Comedy movie.

Class Comedy : public Movie

{

public:

Comedy(int stock, string director, string title, int releaseYear);

~Comedy();

};

**Description:**

Child class of Movie that will contain the data and information of a Drama movie.

Class Drama : public Movie

{

public:

Drama(int stock, string director, string title, int releaseYear);

~Drama();

};

**Description:**

Child class of Movie that will contain the data and information of a Classic movie. This class will be different from the other two child classes as it holds information for the major actor of the movie, and the movie also contains the release month along with the year.

Class Classic : public Movie

{

public:

Classic(int stock, string director, string title, string majorActor, int releaseMonth, int releaseYear);

~Classic();

void setMajorActor(string actor);

void setReleaseMonth(int month);

string getMajorActor() const;

int getReleaseMonth()const;

private:

string majorActor;

int releaseMonth;

};

**Description:**

The activity hub of the program, this class will create the inventory and customer databases for the movie rental store, and then process and execute an arbitrary set of commands and transactions from the text file. The customer and movie inventory information will be held here in the form of a Hashtable and BSTree.

Class Business

{

public:

Business();

~Business();

void buildMovies(string movies);

void buildCustomers(string customers);

void buildTransactions(string transactions);

void executeTransactions();

private:

BST movies;

Hashtable customers;

queue<Transaction> list;

};

**Description:**

This Object will contain the information of a customer so that it can be added to the customer database (Hashtable). Information that will be held here includes customer name, customer identification, and history of that specific customer's transactions.

Class Customer

{  
public:

Customer(int id, string lastName, string firstName);

~Customer();

bool addTransaction(Transaction t);

friend ostream& operator<<(ostream &out, const Customer &cust);

void setId(int cusId);

void setLastName(string lastN);

void setFirstName(string firstN);

int getId()const;

string getLastName()const;

string getFirstName()const;

private:

int id;

string lastName;

string firstName;

vector<Transaction> history;

};

**Description:**

Contains the implementation for a BSTree that will hold Movie objects and serve as the movie database for the movie rental store.

**class** BinTree

{

**friend** ostream & **operator**<<(ostream &, **const** BinTree& b);

**public**:

BinTree();

BinTree(**const** BinTree& b);

~BinTree();

**bool** retrieve(**const** NodeData& target, NodeData\*& settingNode);

**bool** insert(NodeData\* inserting);

**int** getHeight(**const** NodeData& start) **const**;

**void** makeEmpty();

BinTree& **operator**=(**const** BinTree& b);

**bool** **operator**==(**const** BinTree& b) **const**;

**bool** **operator**!=(**const** BinTree& b) **const**;

**private**:

**struct** Node

{

NodeData \*data = **NULL**;

Node \*right = **NULL**;

Node \*left = **NULL**;

};

Node \*root;

**bool** RecInsert(Node\*& currentNode, NodeData\* inserting);

**int** RecHeight(Node\* currentNode) **const**;

**void** Display(ostream& os, Node \*currentNode) **const**;

**void** Copy(Node\* currentNode);

**bool** CheckEquality(Node \*currentLeft, Node \*currentRight) **const**;

**void** Clear(Node\*& currentNode);

};

**Description:**

Contains the implementation for Nodes that will contain Movie object data which will be stored in the Tree.

**class** NodeData {

**friend** ostream & **operator**<<(ostream &, **const** NodeData &);

**public**:

NodeData();

~NodeData();

NodeData(**const** Movie &);

NodeData(**const** NodeData &);

NodeData& **operator**=(**const** NodeData &);

**bool** setData(istream&);

**bool** **operator**==(**const** NodeData &) **const**;

**bool** **operator**!=(**const** NodeData &) **const**;

**bool** **operator**<(**const** NodeData &) **const**;

**bool** **operator**>(**const** NodeData &) **const**;

**bool** **operator**<=(**const** NodeData &) **const**;

**bool** **operator**>=(**const** NodeData &) **const**;

**private**:

Movie data;

};

**Description:**

Contains the implementation for a HashTable that will hold Customer objects and serve as the customer database for the movie rental store.

class HashTable

{  
public:

Hashtable();

~Hashtable();

Customer\* getCustomer(int key);

void addCustomer(int key, Customer \*theCustomer);

Private:

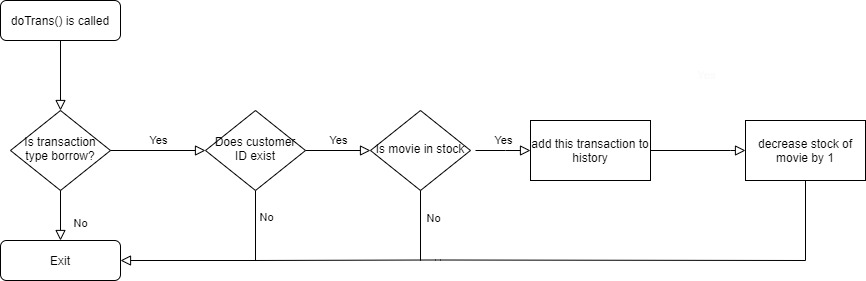
void makeEmpty();

void buildTable();

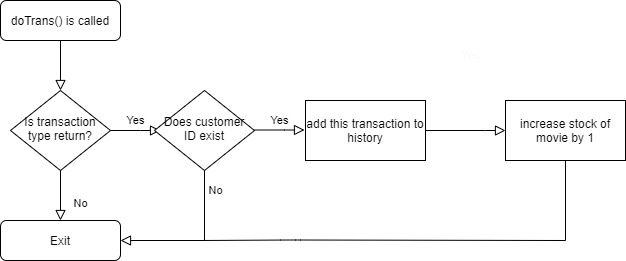
};

**High level Pseudocode (flowchart)**

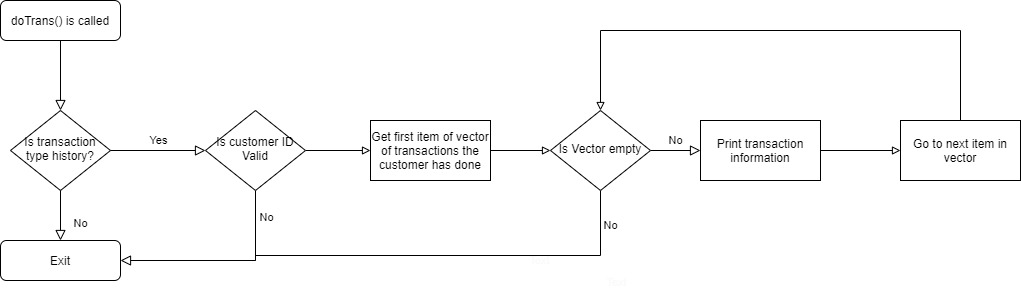
**Borrow: doTrans();**

****

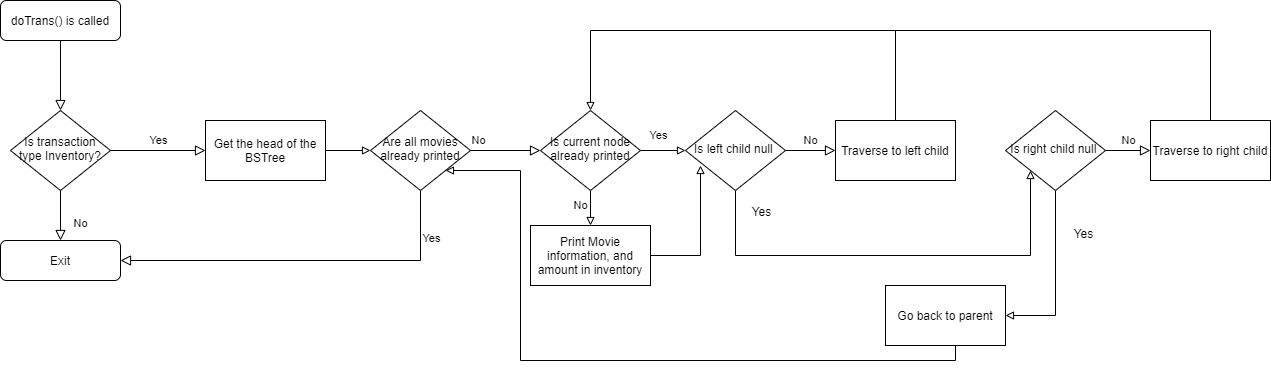
**Return: doTrans();**

****

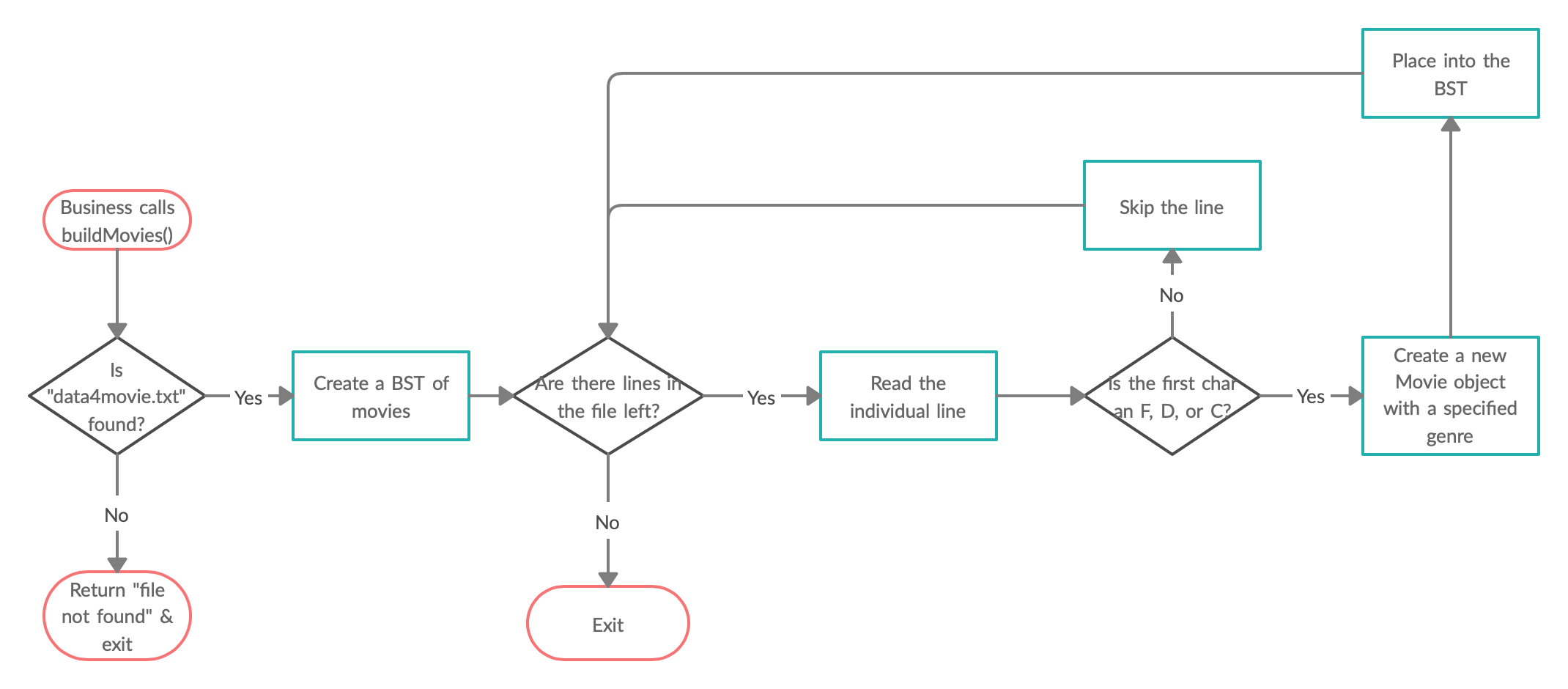
**History: doTrans();**

****

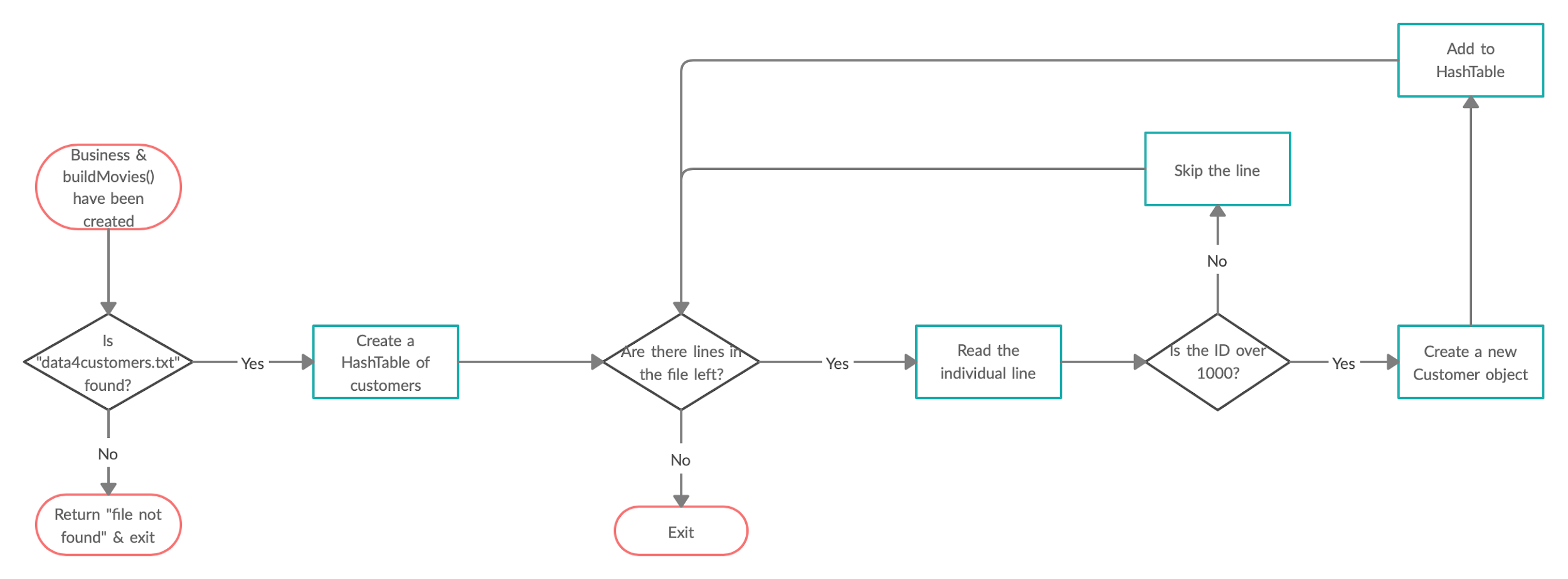
**Inventory: doTrans();**

****

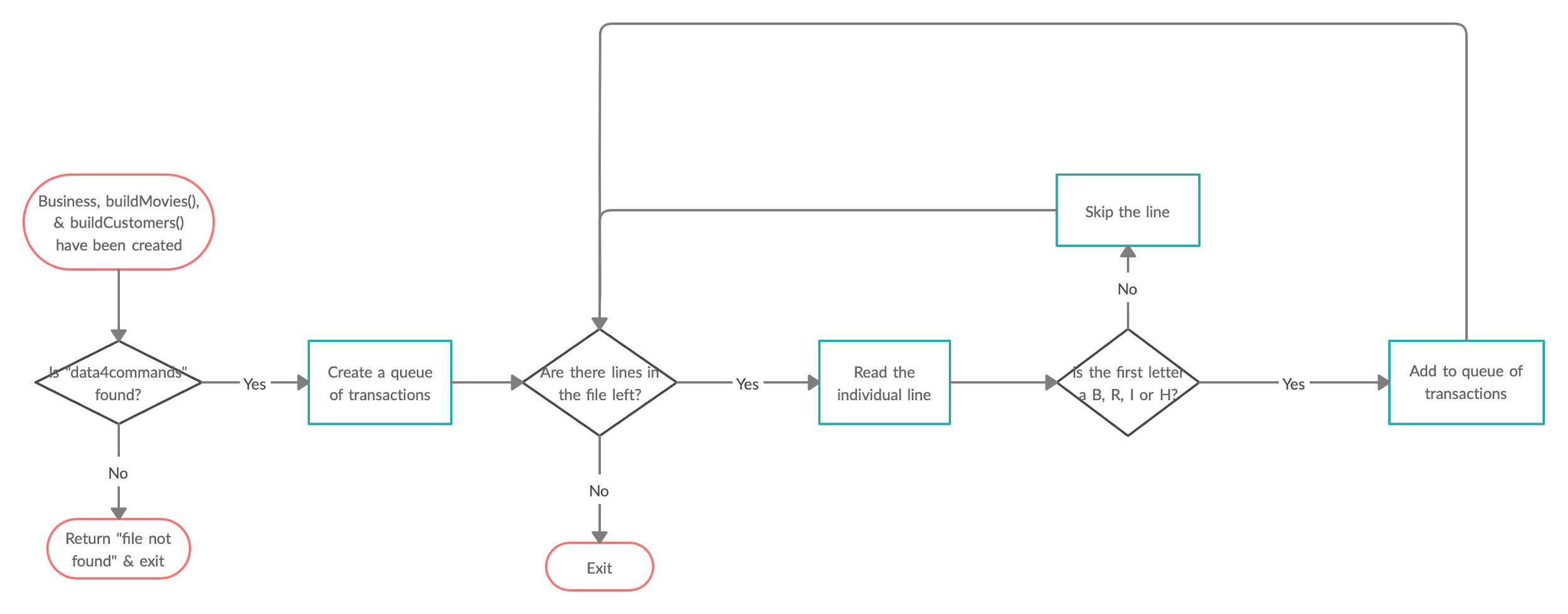
**Business: buildMovies();**

****

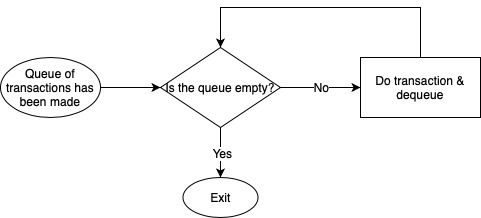
**Business: buildCustomers();**

****

**Business: buildTransactions();**

****

**Business: executeTransactions();**

****