Overview:
This program handles the inventory and sell/buy transactions for a collectibles store. Customers sell/buy history is tracked and can be output.

Input:

- Items will be uniquely identified by the sorting criteria. There will be no duplicates.
- Input files:
  - Input file identifies customers with a 3-digit unique number, no duplicate customers identifiers are included.
  - Input files have the correct formatting and no missing data.
  - Commas are used as delimiters.

Inventory File example:
The data file for initialization of the inventory lists each item on a separate line with a character denoting the type of item, number in inventory, year, grade, type/title/player, publisher/manufacture (if necessary).

```
M, 3, 2001, 65, Lincoln Cent
M, 10, 1913, 70, Liberty Nickel
C, 1, 1938, Mint, Superman, DC
C, 2, 2010, Excellent, X-Men, Marvel
Z, 4, 1986, Raging, Metallica, Master of Puppets
S, 9, 1989, Near Mint, Ken Griffey Jr., Upper Deck
S, 1, 1952, Very Good, Mickey Mantle, Topps
```

Customer Information File Example:
Customer information is stored in a similar file. Customers have a 3-digit ID number that uniquely identifies them.

```
001, Serena Williams
456, Keyser Soze
999, Pele
```

Transactions File Example:
Transactions are processed via commands stored in a file. S for Sell, B for Buy, D for Display, C for Customer, and H for History.

```
S, 001, S, 1989, Near Mint, Ken Griffey Jr., Upper Deck
B, 456, M, 1913, 70, Liberty Nickel
C, 999
D
X, 999, Z, 1986, Raging, Metallica, Master of Puppets
S, 000, Q, 2112, Gnarly, Windows, Microsoft
H
```

Output:

- display() outputs the entire inventory for the store, including the number of each item in inventory. It is displayed in table format with one line per item/transaction starting with coins, comics and then sports cards.
- customer() outputs all the transactions for a customer (in chronological order) including the item. It is displayed in table format with one line per transaction.
    - Chronological order is determined by the order in which customer transactions were performed.
- history() outputs the history for every customer, with the customers in alphabetical order. It is displayed in table format with one line per transaction

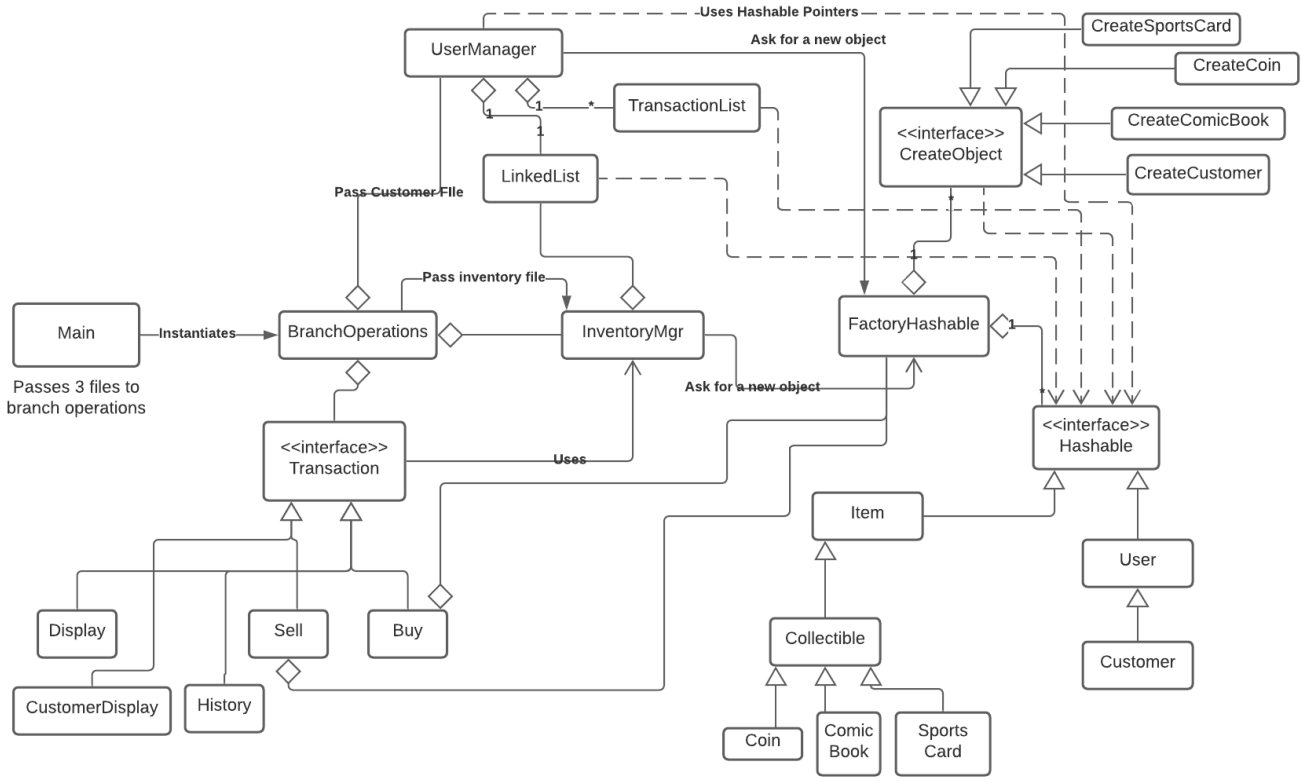
Error Handling:

- Input files have no incomplete data but may contain erroneous identifiers. Error checking for object/method identifiers will be done and message is output to console.
- Adding to inventory: If the item class does not exist an error message is output to console.
- Removing from inventory: if the transaction would cause item count to drop below zero an error message is output to console.

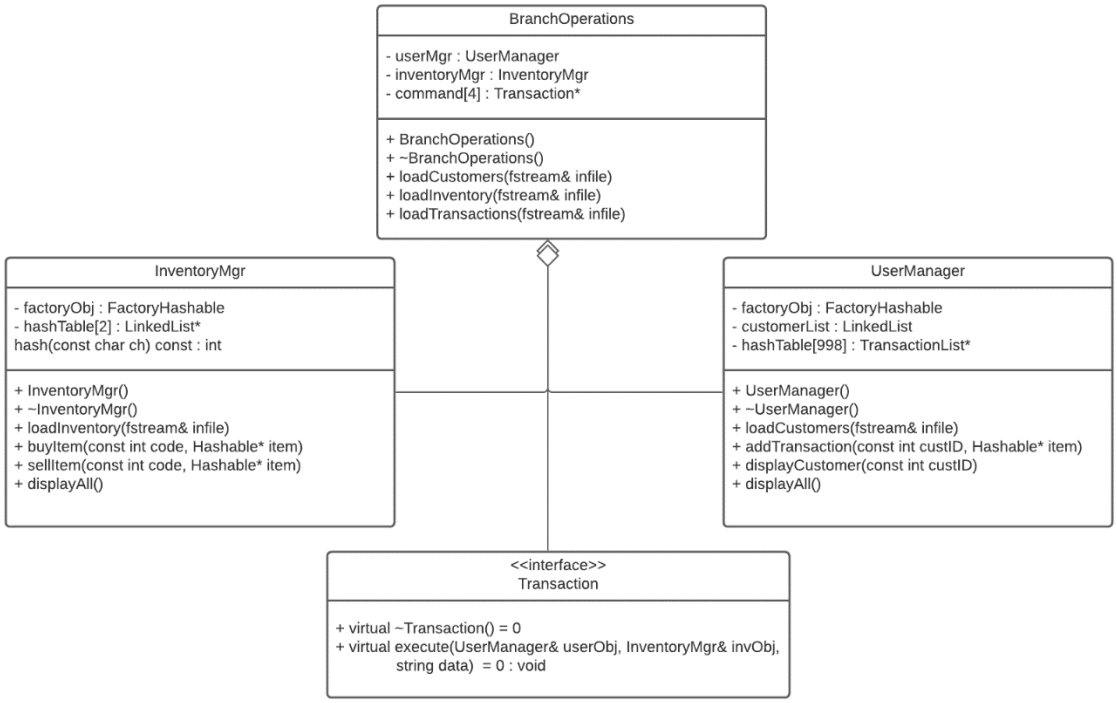
Class diagram:

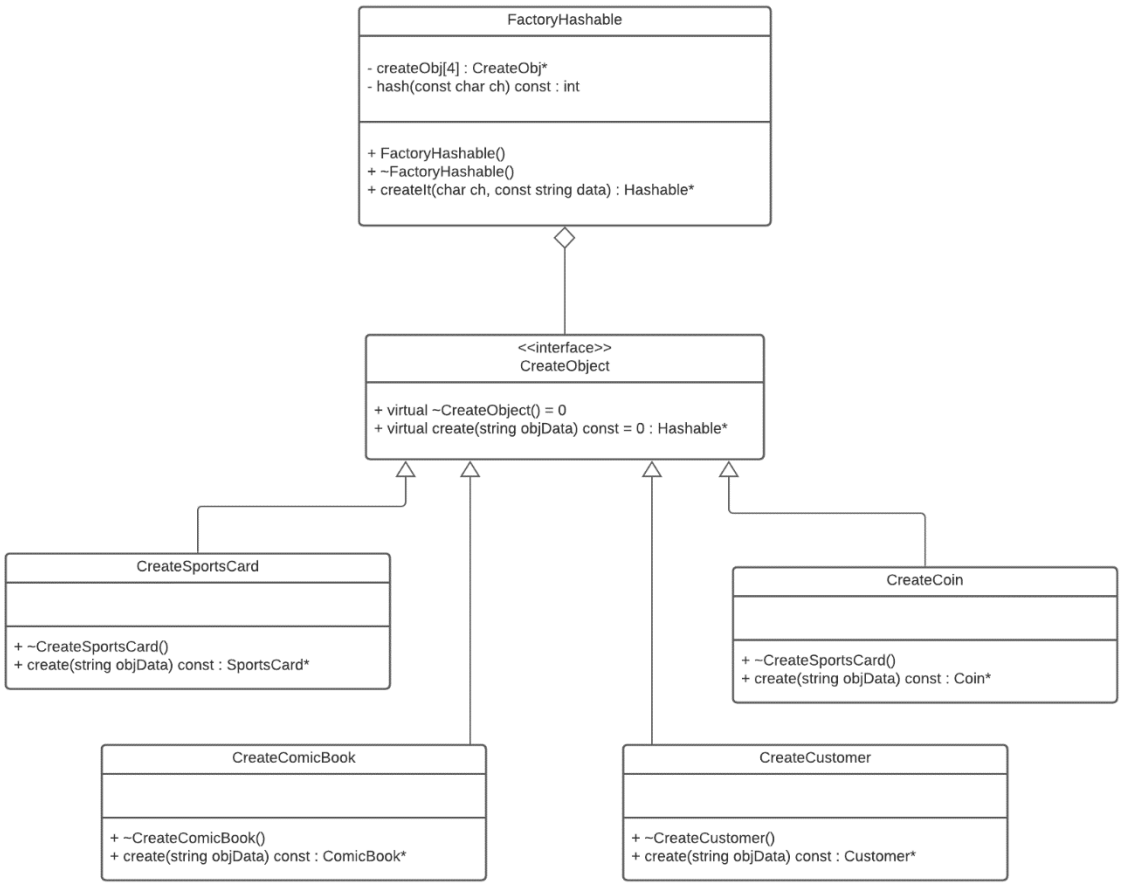
Overall UML Class Diagram.

A combination of command and factory design patterns were used to create a design that is extensible and adheres to the open-closed philosophy of OOP.

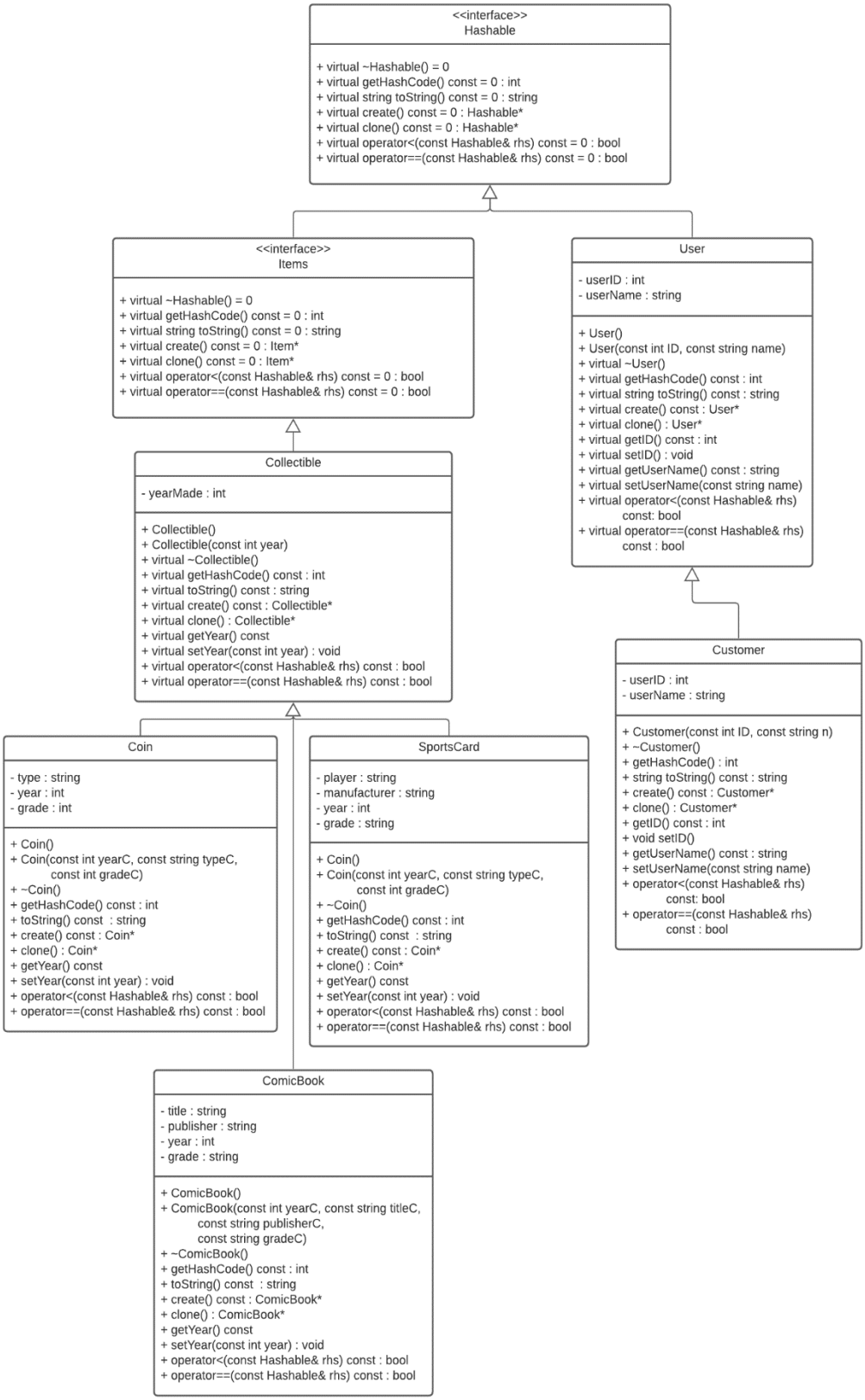


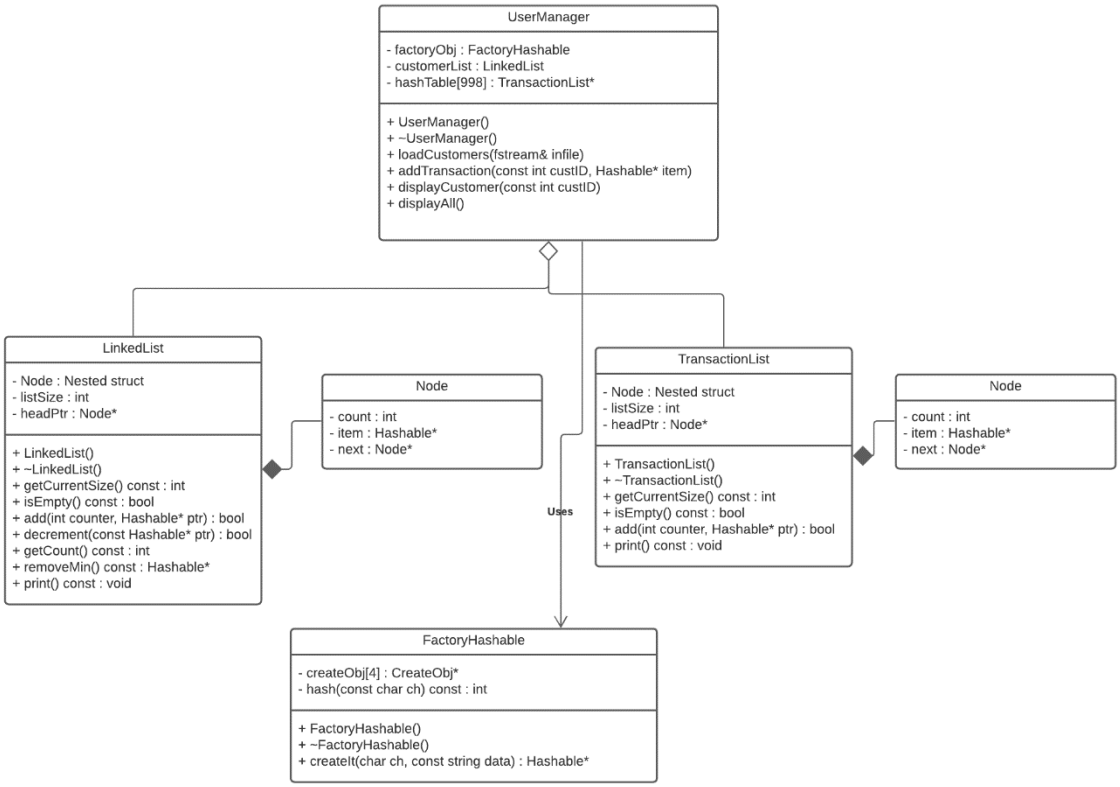
UML Diagrams with Methods:

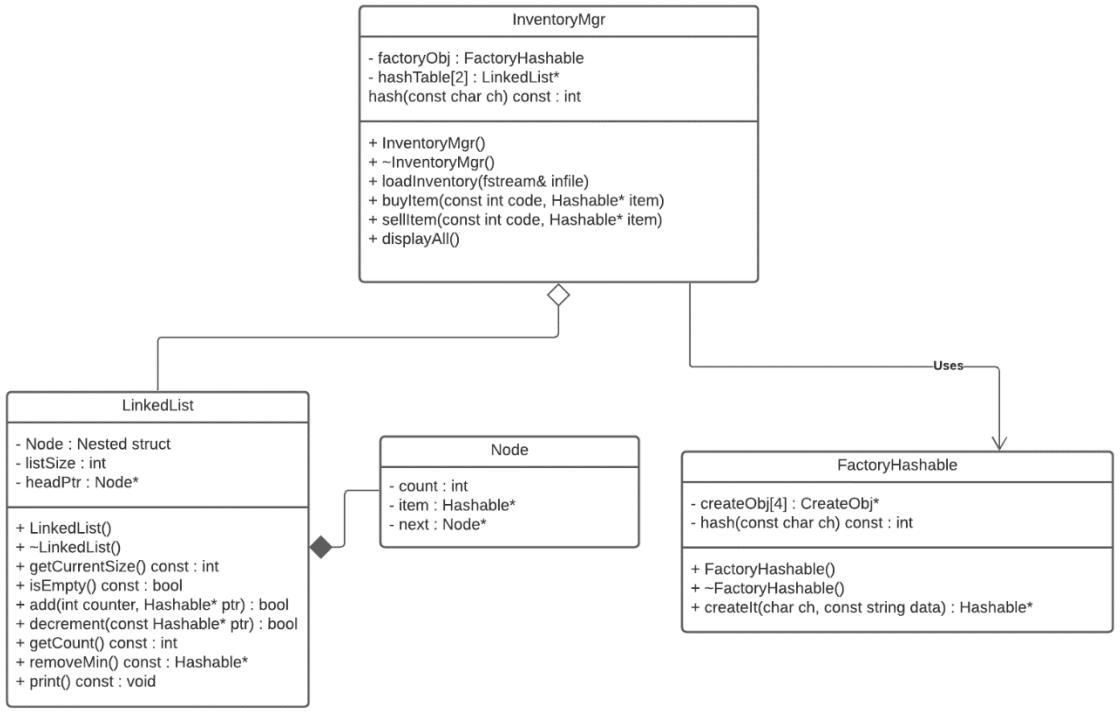


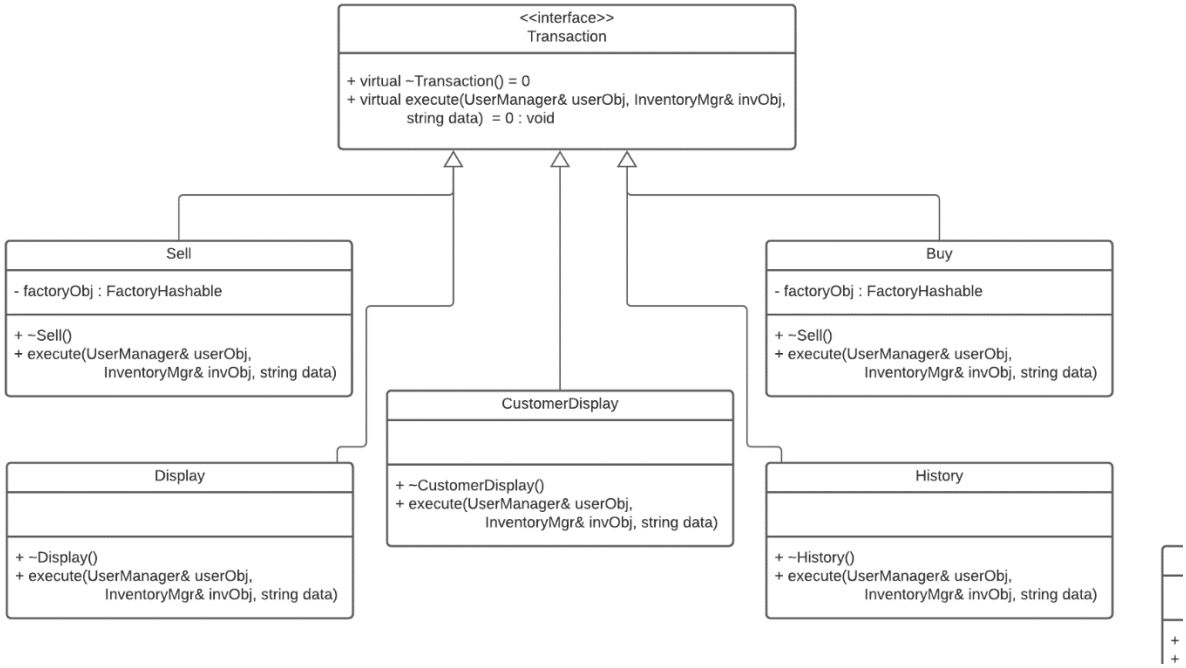

FactoryHashable
- createObj[4] : CreateObj*
- hash(const char ch) const : int
+ FactoryHashable()
+ ~FactoryHashable()
+ createIt(char ch, const string data) : Hashable*
<<interface>>
CreateObject
+ virtual ~CreateObject() = 0
+ virtual create(string objData) const = 0 : Hashable*
CreateSportsCard
+ ~CreateSportsCard()
+ create(string objData) const : SportsCard*
CreateComicBook
+ ~CreateComicBook()
+ create(string objData) const : ComicBook*
CreateCustomer
+ ~CreateCustomer()
+ create(string objData) const : Customer*
CreateCoin
+ ~CreateSportsCard()
+ create(string objData) const : Coin*

]


<<interface>>
Hashable
+ virtual ~Hashable() = 0
+ virtual getHashCode() const = 0 : int
+ virtual string toString() const = 0 : string
+ virtual create() const = 0 : Hashable*
+ virtual clone() const = 0 : Hashable*
+ virtual operator<(const Hashable& rhs) const = 0 : bool
+ virtual operator==(const Hashable& rhs) const = 0 : bool
<<interface>>
Items
+ virtual ~Hashable() = 0
+ virtual getHashCode() const = 0 : int
+ virtual string toString() const = 0 : string
+ virtual create() const = 0 : Item*
+ virtual clone() const = 0 : Item*
+ virtual operator<(const Hashable& rhs) const = 0 : bool
+ virtual operator==(const Hashable& rhs) const = 0 : bool
User
- userID : int
- userName : string
+ User()
+ User(const int ID, const string name)
+ virtual ~User()
+ virtual getHashCode() const : int
+ virtual string toString() const : string
+ virtual create() const : User*
+ virtual clone() : User*
+ virtual getID() const : int
+ virtual setID() : void
+ virtual getUserName() const : string
+ virtual setUserName(const string name)
+ virtual operator<(const Hashable& rhs)
const: bool
+ virtual operator==(const Hashable& rhs)
const : bool
Collectible
- yearMade : int
+ Collectible()
+ Collectible(const int year)
+ virtual ~Collectible()
+ virtual getHashCode() const : int
+ virtual toString() const : string
+ virtual create() const : Collectible*
+ virtual clone() : Collectible*
+ virtual getYear() const
+ virtual setYear(const int year) : void
+ virtual operator<(const Hashable& rhs) const : bool
+ virtual operator==(const Hashable& rhs) const : bool
Customer
- userID : int
- userName : string
+ Customer(const int ID, const string n)
+ ~Customer()
+ getHashCode() : int
+ string toString() const : string
+ create() const : Customer*
+ clone() : Customer*
+ getID() const : int
+ void setID()
+ getUserName() const : string
+ setUserName(const string name)
+ operator<(const Hashable& rhs)
const: bool
+ operator==(const Hashable& rhs)
const : bool
Coin
- type : string
- year : int
- grade : int
+ Coin()
+ Coin(const int yearC, const string typeC,
const int gradeC)
+ ~Coin()
+ getHashCode() const : int
+ toString() const : string
+ create() const : Coin*
+ clone() : Coin*
+ getYear() const
+ setYear(const int year) : void
+ operator<(const Hashable& rhs) const : bool
+ operator==(const Hashable& rhs) const : bool
SportsCard
- player : string
- manufacturer : string
- year : int
- grade : string
+ Coin()
+ Coin(const int yearC, const string typeC,
const int gradeC)
+ ~Coin()
+ getHashCode() const : int
+ toString() const : string
+ create() const : Coin*
+ clone() : Coin*
+ getYear() const
+ setYear(const int year) : void
+ operator<(const Hashable& rhs) const : bool
+ operator==(const Hashable& rhs) const : bool
ComicBook
- title : string
- publisher : string
- year : int
- grade : string
+ ComicBook()
+ ComicBook(const int yearC, const string titleC,
const string publisherC,
const string gradeC)
+ ~ComicBook()
+ getHashCode() const : int
+ toString() const : string
+ create() const : ComicBook*
+ clone() : ComicBook*
+ getYear() const
+ setYear(const int year) : void
+ operator<(const Hashable& rhs) const : bool
+ operator==(const Hashable& rhs) const : bool

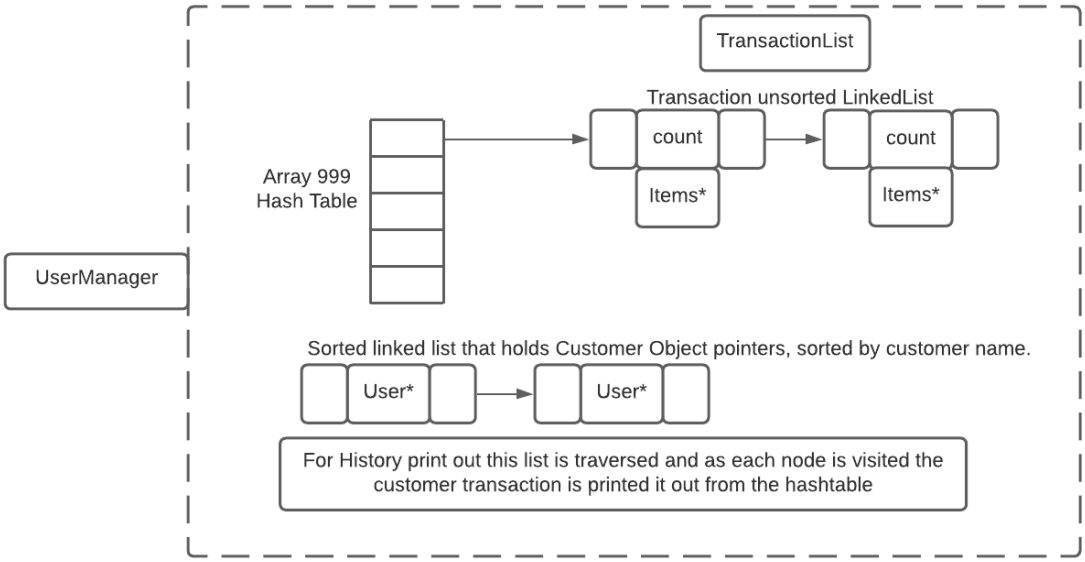

UserManager
- factoryObj : FactoryHashable
- customerList : LinkedList
- hashTable[998] : TransactionList*
+ UserManager()
+ ~UserManager()
+ loadCustomers(fstream& infile)
+ addTransaction(const int custID, Hashable* item)
+ displayCustomer(const int custID)
+ displayAll()
LinkedList
- Node : Nested struct
- listSize : int
- headPtr : Node*
+ LinkedList()
+ ~LinkedList()
+ getCurrentSize() const : int
+ isEmpty() const : bool
+ add(int counter, Hashable* ptr) : bool
+ decrement(const Hashable* ptr) : bool
+ getCount() const : int
+ removeMin() const : Hashable*
+ print() const : void
Node
- count : int
- item : Hashable*
- next : Node*
TransactionList
- Node : Nested struct
- listSize : int
- headPtr : Node*
+ TransactionList()
+ ~TransactionList()
+ getCurrentSize() const : int
+ isEmpty() const : bool
+ add(int counter, Hashable* ptr) : bool
+ print() const : void
Node
- count : int
- item : Hashable*
- next : Node*
Uses
FactoryHashable
- createObj[4] : CreateObj*
- hash(const char ch) const : int
+ FactoryHashable()
+ ~FactoryHashable()
+ createIt(char ch, const string data) : Hashable*


InventoryMgr
- factoryObj : FactoryHashable
- hashTable[2] : LinkedList*
hash(const char ch) const : int
+ InventoryMgr()
+ ~InventoryMgr()
+ loadInventory(fstream& infile)
+ buyItem(const int code, Hashable* item)
+ sellItem(const int code, Hashable* item)
+ displayAll()
LinkedList
- Node : Nested struct
- listSize : int
- headPtr : Node*
+ LinkedList()
+ ~LinkedList()
+ getCurrentSize() const : int
+ isEmpty() const : bool
+ add(int counter, Hashable* ptr) : bool
+ decrement(const Hashable* ptr) : bool
+ getCount() const : int
+ removeMin() const : Hashable*
+ print() const : void
Node
- count : int
- item : Hashable*
- next : Node*
Uses
FactoryHashable
- createObj[4] : CreateObj*
- hash(const char ch) const : int
+ FactoryHashable()
+ ~FactoryHashable()
+ createIt(char ch, const string data) : Hashable*

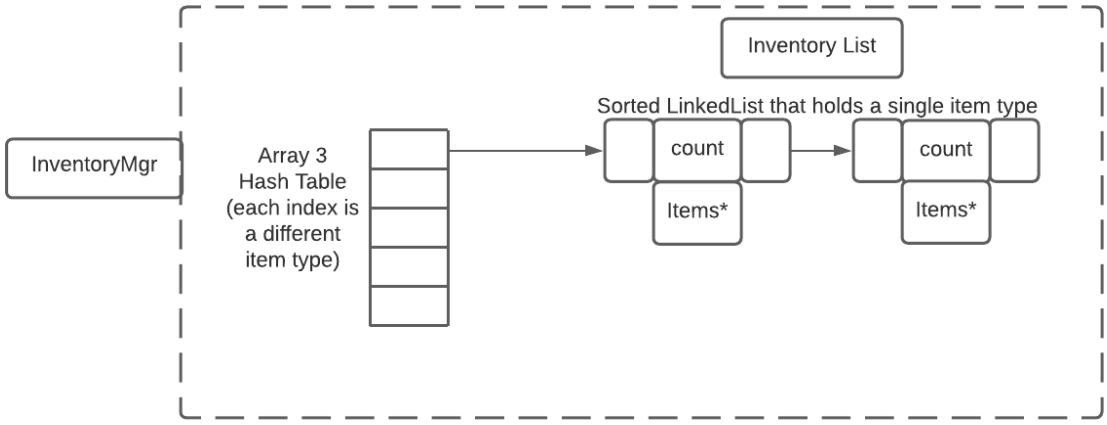

<<interface>>
Transaction
+ virtual ~Transaction() = 0
+ virtual execute(UserManager& userObj, InventoryMgr& invObj,
string data) = 0 : void
Sell
- factoryObj : FactoryHashable
+ ~Sell()
+ execute(UserManager& userObj,
InventoryMgr& invObj, string data)
Buy
- factoryObj : FactoryHashable
+ ~Sell()
+ execute(UserManager& userObj,
InventoryMgr& invObj, string data)
CustomerDisplay
+ ~CustomerDisplay()
+ execute(UserManager& userObj,
InventoryMgr& invObj, string data)
Display
+ ~Display()
+ execute(UserManager& userObj,
InventoryMgr& invObj, string data)
History
+ ~History()
+ execute(UserManager& userObj,
InventoryMgr& invObj, string data)

Memory diagram:

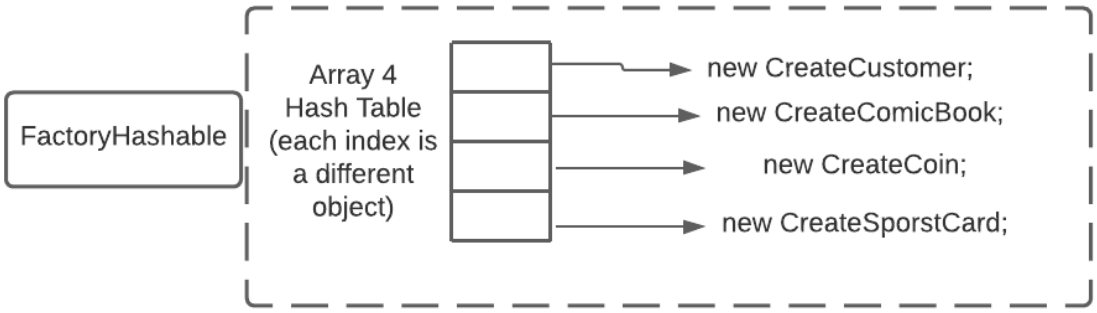
Customer Transactions:
Hash table is used to find customers quickly, a maximum of 999 customers is used. Each index of the hashtable has a unsorted linked list to keep track of transactions. A sorted linked list is kept to sort all customers.



Inventory:



Factory Class HashTable:



Branch Operations HashTable:

A similar table to that of the Factory class is used here.