Class Interaction

Use case 1: display library contents

Manager class:

```
void processTransaction (istream& ) {
        read one line from the file;
        create transaction object by calling the createIt method in transactionFactory class;
        check if returned transaction is NULL or not;
        if not NULL, call execute method on this transaction passing itemCollection and
        userCollection as parameter;
        use passed-in itemCollection to call its display method, to display all the items
        owned by the library;
}
```

DisplayTransaction class:

```
void execute (ItemCollection* , UserCollection*) {
        use passed-in itemCollection to call its display method;
}
```

ItemCollection class:

```
void display( ) const {
        call display method in ItemTree class to display all the ItemTrees that stored in
        ItemCollection;
}
```

ItemTree class:

```
void display( ) const {
        display all the Items that are stored in a tree, with how many hard copies are
        checked out and how many remain;
}
```

Use case 2: check out an Item

Manager class:

```
void processTransaction (istream& ) {
        read one line from the file;
        create transaction object by calling the createIt method in transactionFactory class;
        check if returned transaction is not a NULL and call execute method on
        this transaction passing itemCollection and userCollection ;
}
```

TransactionFactory class:

```
Transaction *createIt (string) {
        Read the first char from the string to check what type of transaction has to be
        performed;
        Hash the received char to get the right create method in Transaction class;
        Performed the create method in Transaction class and return the created
        transaction object;
}
```

CheckOutTransaction class:

```
void execute (ItemCollection* , UserCollection*) {
        Create a user object;
        Find the user object in UserCollection using retrieve method in UserCollection;
        Check if the user exist and if exist create the item object;
        Find the item object in ItemCollection using retrieve method in ItemCollection;
        Check if the item exists and if there are available copies of this book;
        If both conditions are true, increase the number of checked out books and add
        the transaction to user history;
}
```

Use case 3: return an Item

Manager class:

```
void processTransaction (istream& ) {
        read one line from the file;
        create transaction object by calling the createIt method in transactionFactory class;
        check if returned transaction is not a NULL and call execute method on
        this transaction passing itemCollection and userCollection ;
}
```

TransactionFactory class:

```
Transaction *createIt (string) {
        Read the first char from the string to check what type of transaction has to be
        performed;
        Hash the received char to get the right create method in Transaction class;
        Performed the create method in Transaction class and return the created
        transaction object;
}
```

ReturnTransaction class:

```
void execute (ItemCollection* , UserCollection*) {
        Create a user object;
        Find the user object in UserCollection using retrieve method in UserCollection;
        Check if the user exist and if exist create the item object;
        Find the item object in ItemCollection using retrieve method in ItemCollection;
        Check if the item exists;
        If both conditions are true, decrease the number of checked out books and add
        the transaction to user history;
```

```
    }
```

Use case 4: display a Patron’s transaction history

Manager class:

```
void processTransaction (istream& ) {
    read one line from the file;
    create transaction object by calling the createIt method in transactionFactory class;
    check if returned transaction is not a NULL and call execute method on
    this transaction passing itemCollection and userCollection ;
}
```

TransactionFactory class:

```
Transaction *createIt (string) {
    Read the first char from the string to check what type of transaction has to be
    performed;
    Hash the received char to get the right create method in Transaction class;
    Performed the create method in Transaction class and return the created
    transaction object;
}
```

HistoryTransaction class:

```
void execute (ItemCollection* , UserCollection*) {
    Create a user object;
    Find the user object in UserCollection using retrieve method;
    If the user exists, call display method using User’s History data member;
    Display iterates through the list of transactions;
}
```