UML Diagram for **MOVIE Store**

+ operator<< (ostream&, Drama&)

: return ostream&

StoreManager (logic and processing goes here) + movieDB: MovieDB + userDB: UserDB + buildDatabases () --- buildUserDB () abstract class: ItemFactory --- buildMovieDB () + originalItems : array containing emtpy instances of each Item type + executeTransactions () + processTransaction (Transaction) : Bool + createInstance (string/char objCode) : return Item* --- rent (Item*, Customer*) : return bool + hash (string/char itemTypeCode) : return int --- collect (Item*, Customer*): return bool -- displaySingleUser (Customer*) --- displayMovieDB() **MovieFactory** CustomerFactory + originalMovies : array containing emtpy + originalCustomers : array containing emtpy instances of each Movie type instances of each Customer type + createInstance (string/char movieCode): + createInstance (string/char userTypeCode): return Movie* return Customer* + hash (string/char userTypeCode) : return int + hash (string/char movieCode) : return int abstract class: Item **Tree** + id (unique among its own type) + root : Node* + nodeCount : int + pure virtual createEmptyInstance(): return Item* + insert (Item*) : return bool + pure virtual setData(string) : bool + remove (Item*): return bool Movie **Customer (open for extension)** + medium : string/char + personal_info : struct + director: string + rentedItems : linked list + title: string + borrow (Item*) : return bool + date: string + return (Item*) : return bool + borrowCustomers : list containing record of + operator== (Customer&) : return bool customers who have borrowed this movie. + operator< (Customer&) : return bool + pure virtual isRentedBy(Customer*) Classic + major_actor_fname : string **Drama** Comedy + major actor Iname : string + operator< (Drama&) : return bool

+ operator< (Comedy&) : return bool

+ operator<< (ostream&, Comedy&)

: return ostream&

+ operator< (Classic&) : return bool

+ operator<< (ostream&, Classic&)

: return ostream&

Transaction

- + Instruction: Char + UserID : Int
- + MediaType : Char + MediaCategory: Char
- + MediaDetails : String

Media details are formatted differently for different media type. We thus store it as a string to be parsed by the appropriate logic inside the appropriate objects themselves.

InputManager

- + transactionList : Transaction []
- + movieDetails : string []
- + parseTransaction ()
- + parseMovieDetails ()
- + getNextTransaction (): return Transaction
- + getNextMovie () : return string

abstract class: GenericDatabase

- + itemTrees : AVL Trees containing different types of a generic item
- + pure virtual findByID (int) : return Item
- + pure virtual displayAll ()
- + pure virtual addItem (Item*): return bool
- + hash (string itemTypeCode) : return int

UserDB

- + userTree : AVL tree containing customers
- + findByID (int): return Customer*
- + findByLastName (string) : Customer**
- + insertUser (Customer*) : return bool

MovieDB

- + treeList : list containing Movie Trees
- + movieStock : hash table
- + printRentedMovies(Customer*)
- + insertMovie (Movie*) : return bool
- + findByID (int) : return Movie*
- + findByTitle (string) : return Movie**
- + increaseStock (Movie*, int)