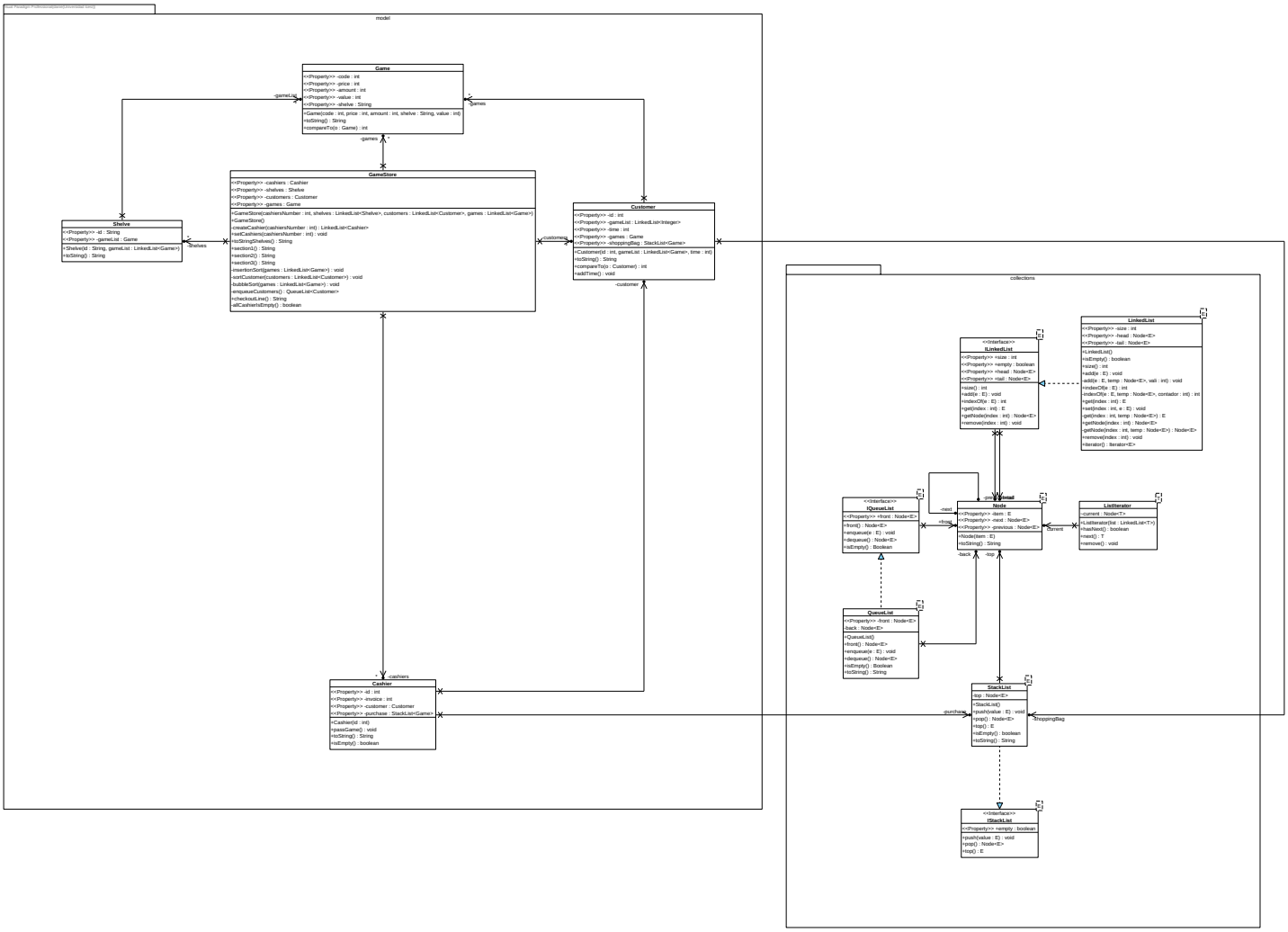
model
Game
<<Property>> -code : int
<<Property>> -price : int
<<Property>> -amount : int
<<Property>> -value : int
<<Property>> -shelve : String
+Game(code : int, price : int, amount : int, shelve : String, value : int)
+toString() : String
+compareTo(o : Game) : int
-gameList
-games
*
*
-games
GameStore
<<Property>> -cashiers : Cashier
<<Property>> -shelves : Shelve
<<Property>> -customers : Customer
<<Property>> -games : Game
+GameStore(cashiersNumber : int, shelves : LinkedList<Shelve>, customers : LinkedList<Customer>, games : LinkedList<Game>)
+GameStore()
-createCashier(cashiersNumber : int) : LinkedList<Cashier>
-setCashiers(cashiersNumber : int) : void
+toStringShelves() : String
+section1() : String
+section2() : String
+section3() : String
-insertionSort(games : LinkedList<Game>) : void
-sortCustomer(customers : LinkedList<Customer>) : void
-bubbleSort(games : LinkedList<Game>) : void
-enqueueCustomers() : QueueList<Customer>
+checkoutLine() : String
-allCashiersIsEmpty() : boolean
Shelve
<<Property>> -id : String
<<Property>> -gameList : Game
+Shelve(id : String, gameList : LinkedList<Game>)
+toString() : String
-shelves
-customers
Customer
<<Property>> -id : int
<<Property>> -gameList : LinkedList<Integer>
<<Property>> -time : int
<<Property>> -games : Game
<<Property>> -shoppingBag : StackList<Game>
+Customer(id : int, gameList : LinkedList<Game>, time : int)
+toString() : String
+compareTo(o : Customer) : int
+addTime() : void
-customer
-cashiers
Cashier
<<Property>> -id : int
<<Property>> -invoice : int
<<Property>> -customer : Customer
<<Property>> -purchase : StackList<Game>
+Cashier(id : int)
+passGame() : void
+toString() : String
+isEmpty() : boolean
collections
LinkedList
<<Property>> -size : int
<<Property>> -head : Node<E>
<<Property>> -tail : Node<E>
+LinkedList()
+isEmpty() : boolean
+size() : int
+add(e : E) : void
+add(e : E, temp : Node<E>, vali : int) : void
+indexOf(e : E) : int
+indexOf(e : E, temp : Node<E>, contador : int) : int
+get(index : int) : E
+set(index : int, e : E) : void
-get(index : int, temp : Node<E>) : E
+getNode(index : int) : Node<E>
-getNode(index : int, temp : Node<E>) : Node<E>
+remove(index : int) : void
+Iterator() : Iterator<E>
<<Interface>>
ILinkedList
<<Property>> +size : int
<<Property>> +empty : boolean
<<Property>> +head : Node<E>
<<Property>> +tail : Node<E>
+size() : int
+add(e : E) : void
+indexOf(e : E) : int
+get(index : int) : E
+getNode(index : int) : Node<E>
+remove(index : int) : void
getNext()
Node
<<Property>> -item : E
<<Property>> -next : Node<E>
<<Property>> -previous : Node<E>
+Node(item : E)
+toString() : String
-next
-prev
-current
ListIterator
-current : Node<T>
+ListIterator(list : LinkedList<T>)
+hasNext() : boolean
+next() : T
+remove() : void
<<Interface>>
IQueueList
<<Property>> +front : Node<E>
+front() : Node<E>
+enqueue(e : E) : void
+dequeue() : Node<E>
+isEmpty() : Boolean
-front
-back
QueueList
<<Property>> -front : Node<E>
-back : Node<E>
+QueueList()
+front() : Node<E>
+enqueue(e : E) : void
+dequeue() : Node<E>
+isEmpty() : Boolean
+toString() : String
StackList
-top : Node<E>
+StackList()
+push(value : E) : void
+pop() : Node<E>
+top() : E
+isEmpty() : boolean
+toString() : String
-top
-purchase
-shoppingBag
<<Interface>>
IStackList
<<Property>> +empty : boolean
+push(value : E) : void
+pop() : Node<E>
+top() : E