FN

Pr

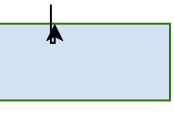
C

NMS

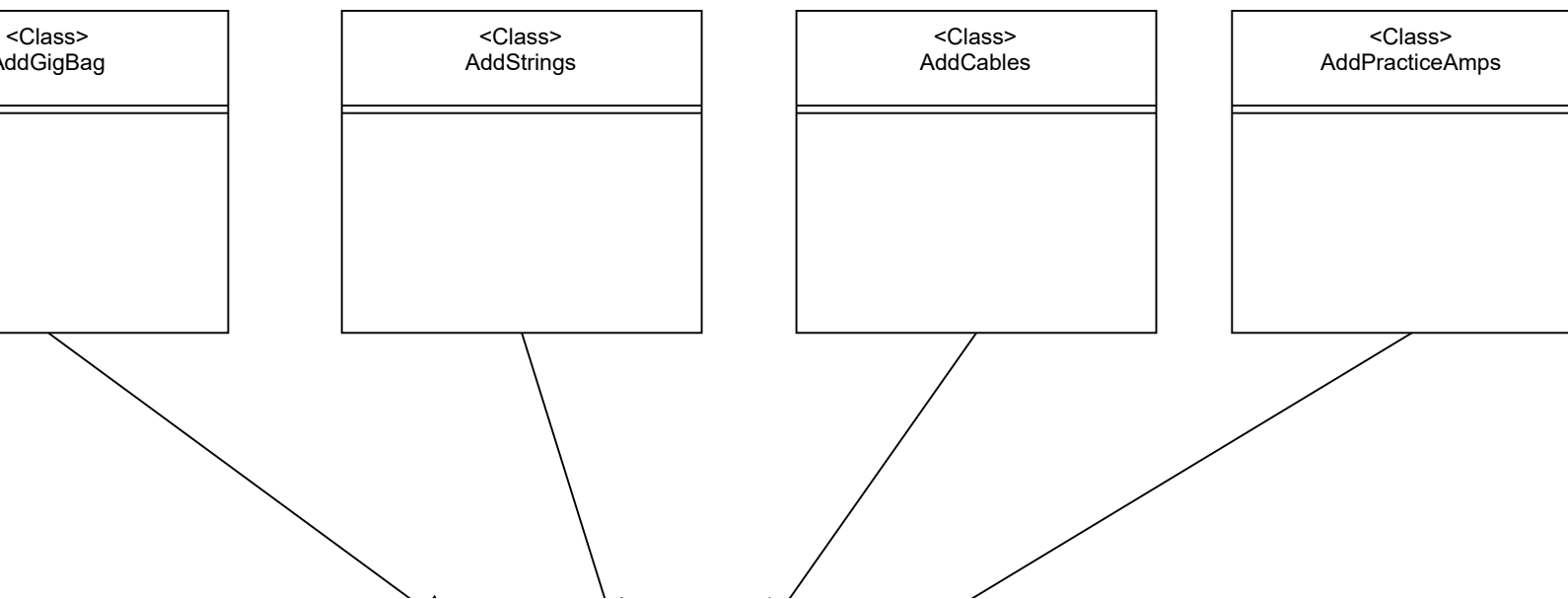
oject 4 UML

Team member: Srinivas Akhil Mallela, Likhitha Katakam, Vishal Prabhachandar

olor code

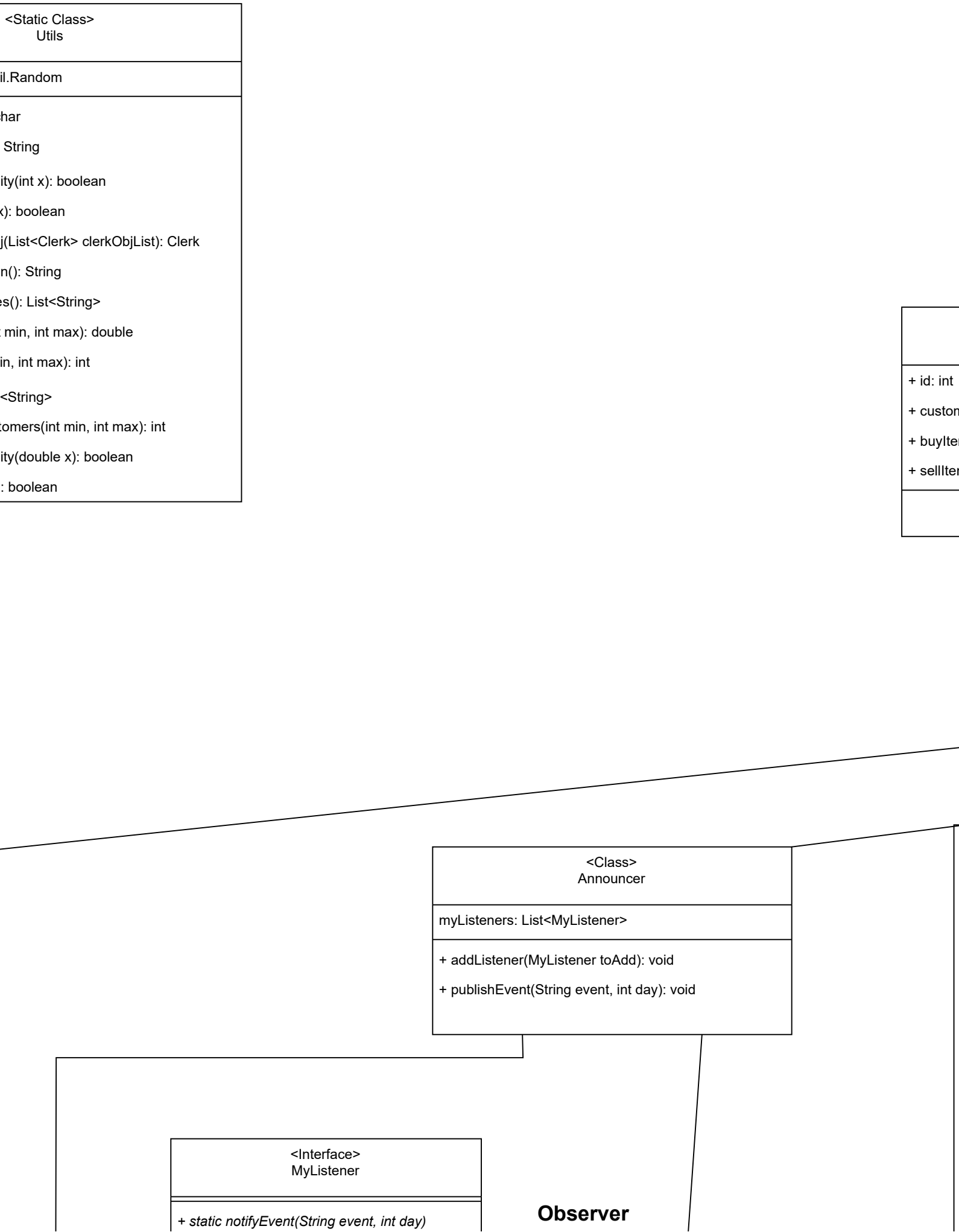


New components

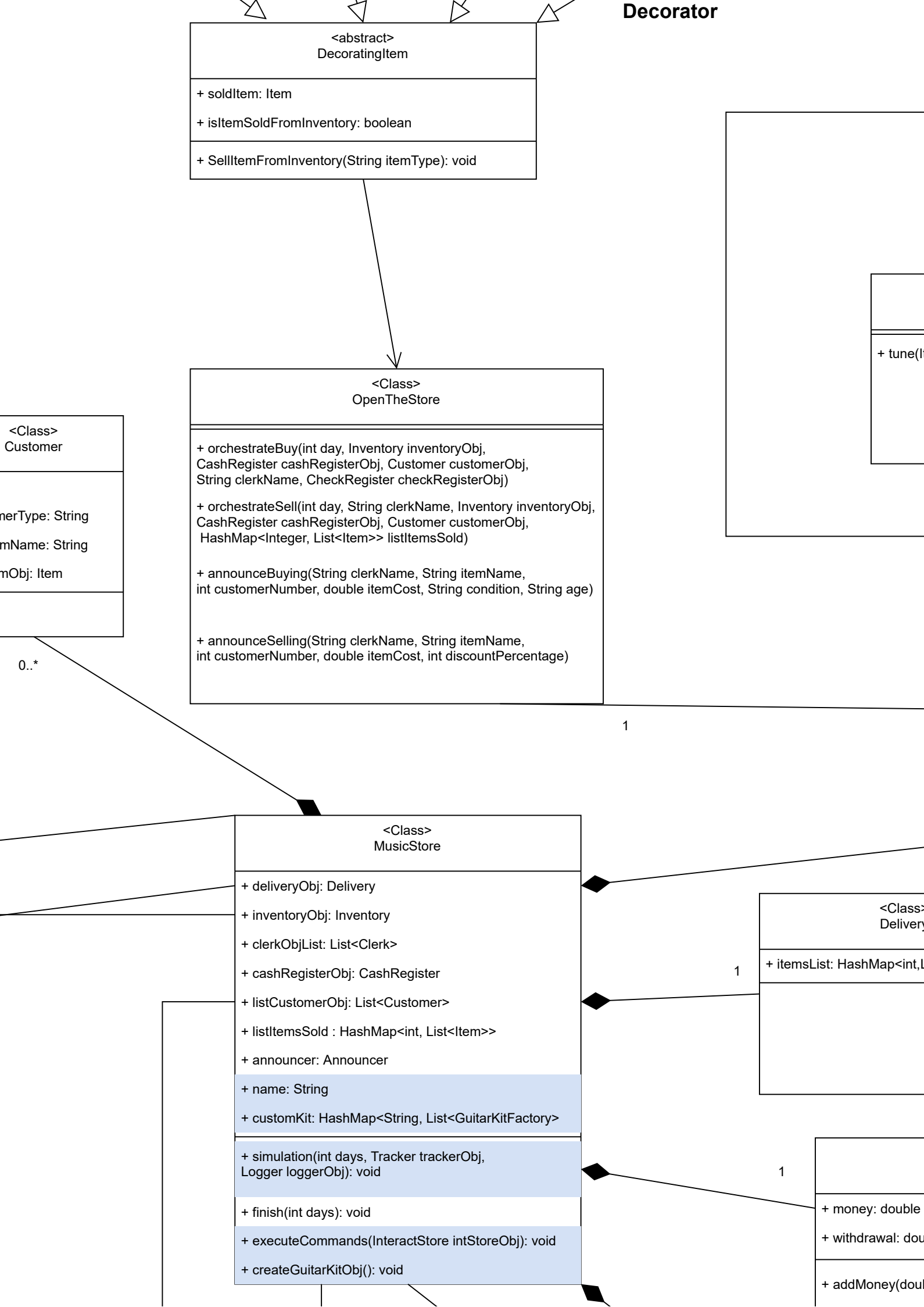

<Class>
AddGigBag
<Class>
AddStrings
<Class>
AddCables
<Class>
AddPracticeAmps

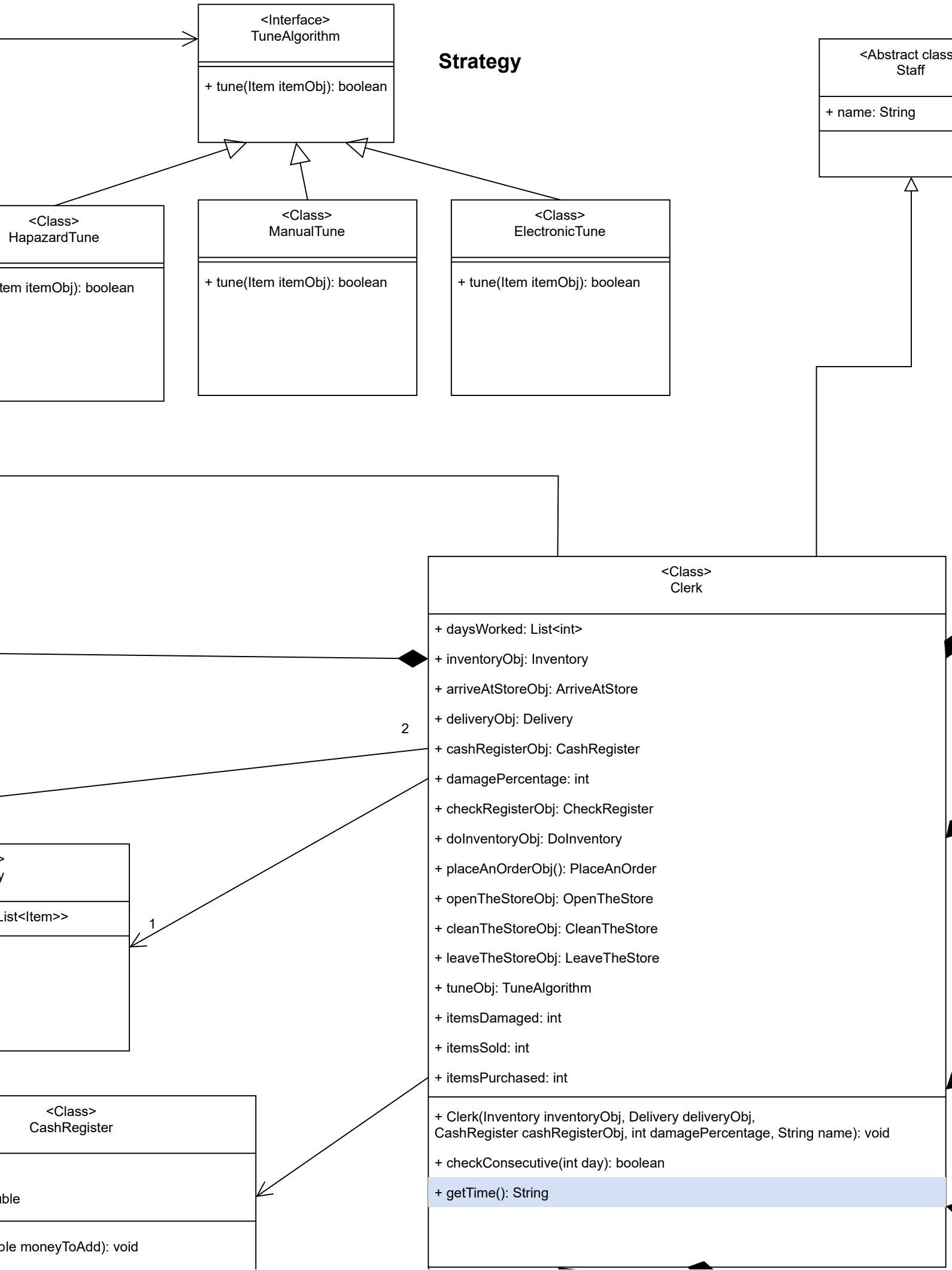
<Abstract Class>
GuitarKitFactory
+ createBridge(): Bridge
+ createKnobSet(): KnobSet
+ createCovers(): Covers
+ createNeck(): Neck
+ createPickGuard(): PickGuard

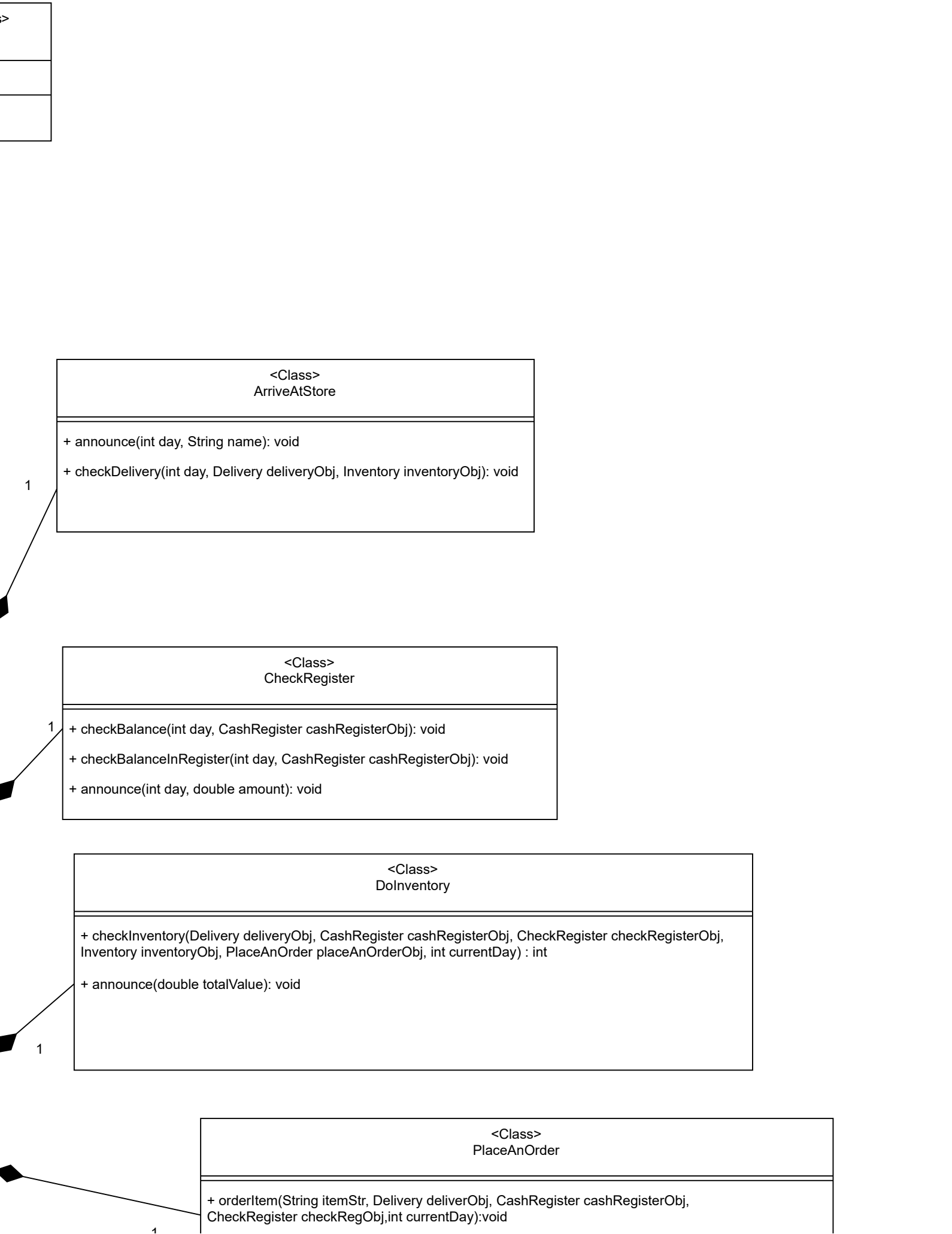
+ randomGen: java.ut
+ getRandomSize(): c
+ getRandomName():
+ getRandomProbabil
+ getRandomBuy(int x
+ getRandomClerkOb
+ getRandomConditio
+ getRandomItemtype
+ getRandomPrice(int
+ getRandomInt(int m
+ getItemTypes(): List
+ getRandomBuyCust
+ getRandomProbabil
+ getRandomBoolean

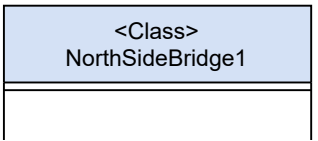

<Static Class>
Utils
il.Random
har
String
ity(int x): boolean
x): boolean
j(List<Clerk> clerkObjList): Clerk
n(): String
es(): List<String>
min, int max): double
in, int max): int
<String>
tomers(int min, int max): int
ity(double x): boolean
: boolean
+ id: int
+ custon
+ buyIte
+ sellIte
<Class>
Announcer
myListeners: List<MyListener>
+ addListener(MyListener toAdd): void
+ publishEvent(String event, int day): void
<Interface>
MyListener
+ static notifyEvent(String event, int day)
Observer

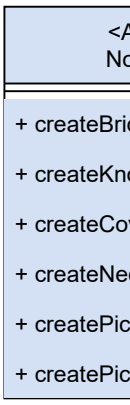
Decorator

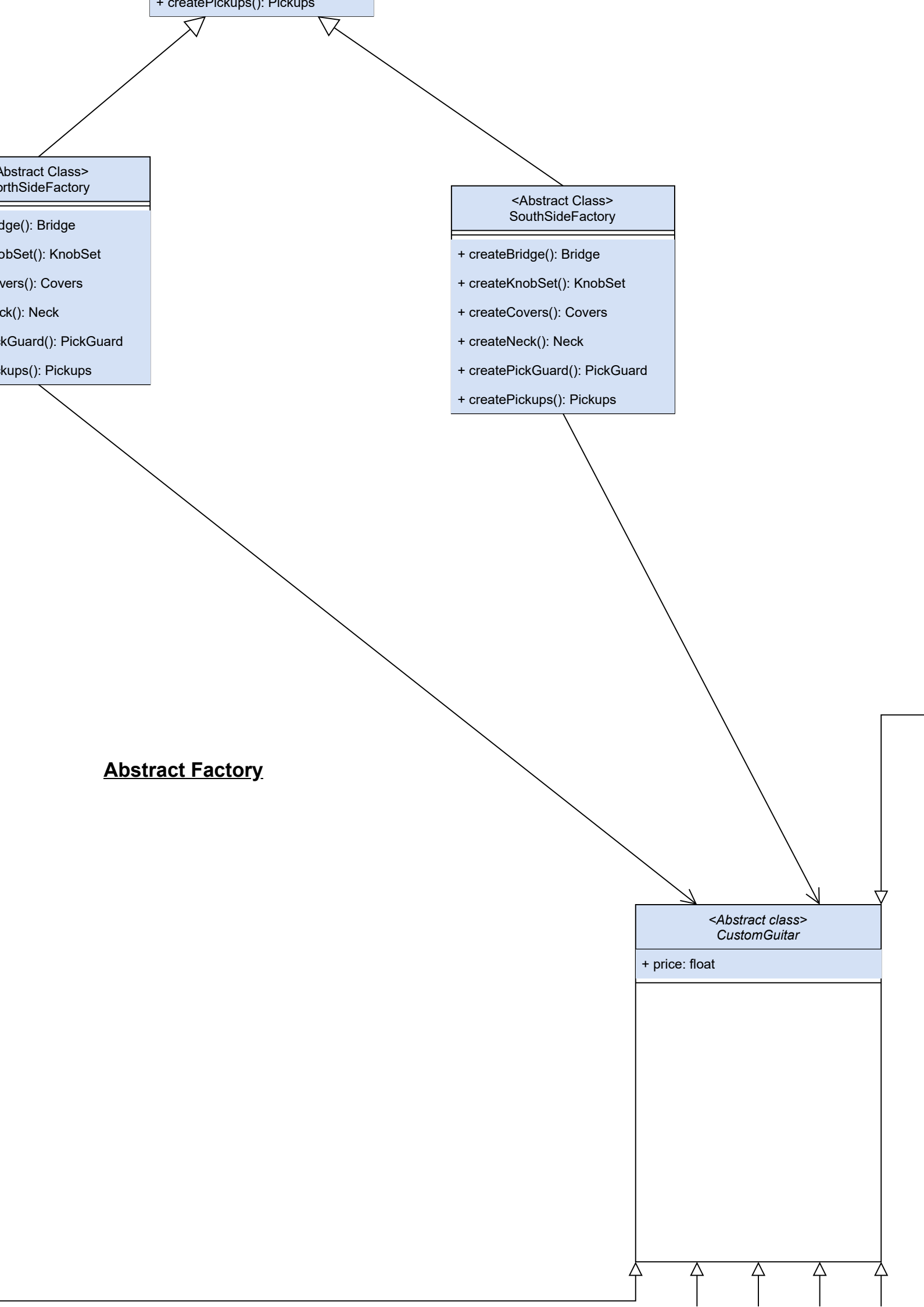

<abstract>
DecoratingItem
+ soldItem: Item
+ isItemSoldFromInventory: boolean
+ SellItemFromInventory(String itemType): void
<Class>
OpenTheStore
+ orchestrateBuy(int day, Inventory inventoryObj, CashRegister cashRegisterObj, Customer customerObj, String clerkName, CheckRegister checkRegisterObj)
+ orchestrateSell(int day, String clerkName, Inventory inventoryObj, CashRegister cashRegisterObj, Customer customerObj, HashMap<Integer, List<Item>> listItemsSold)
+ announceBuying(String clerkName, String itemName, int customerNumber, double itemCost, String condition, String age)
+ announceSelling(String clerkName, String itemName, int customerNumber, double itemCost, int discountPercentage)
<Class>
Customer
merType: String
mName: String
mObj: Item
0..*
1
<Class>
MusicStore
+ deliveryObj: Delivery
+ inventoryObj: Inventory
+ clerkObjList: List<Clerk>
+ cashRegisterObj: CashRegister
+ listCustomerObj: List<Customer>
+ listItemsSold : HashMap<int, List<Item>>
+ announcer: Announcer
+ name: String
+ customKit: HashMap<String, List<GuitarKitFactory>
+ simulation(int days, Tracker trackerObj, Logger loggerObj): void
+ finish(int days): void
+ executeCommands(InteractStore intStoreObj): void
+ createGuitarKitObj(): void
1
+ itemsList: HashMap<int,
1
+ money: double
+ withdrawal: dou
+ addMoney(doul
+ tune(I

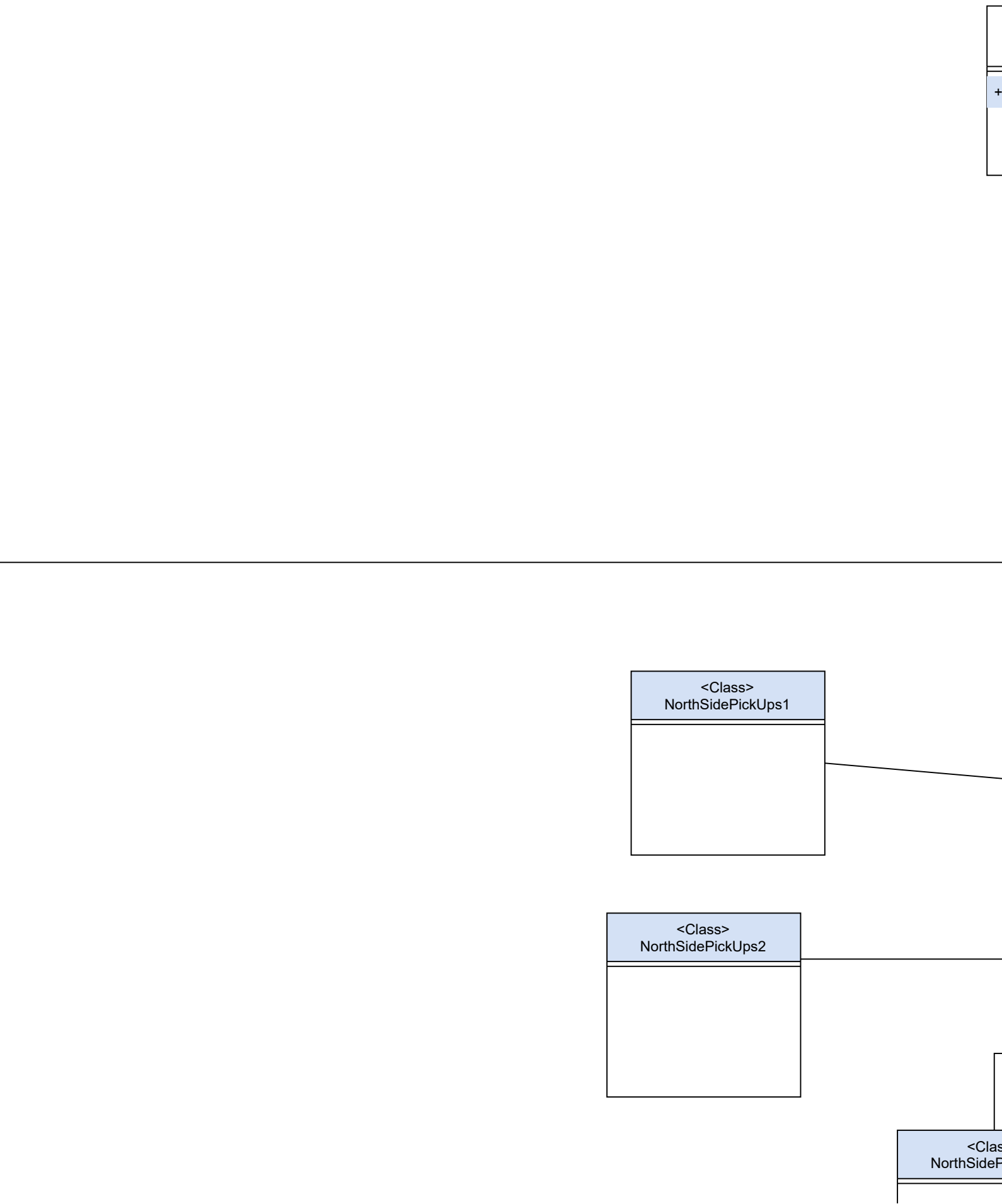

<Interface>
TuneAlgorithm
+ tune(Item itemObj): boolean
Strategy
<Abstract class
Staff
+ name: String
<Class>
HapazardTune
tem itemObj): boolean
<Class>
ManualTune
+ tune(Item itemObj): boolean
<Class>
ElectronicTune
+ tune(Item itemObj): boolean
<Class>
Clerk
+ daysWorked: List<int>
+ inventoryObj: Inventory
+ arriveAtStoreObj: ArriveAtStore
+ deliveryObj: Delivery
+ cashRegisterObj: CashRegister
+ damagePercentage: int
+ checkRegisterObj: CheckRegister
+ doInventoryObj: DoInventory
+ placeAnOrderObj(): PlaceAnOrder
+ openTheStoreObj: OpenTheStore
+ cleanTheStoreObj: CleanTheStore
+ leaveTheStoreObj: LeaveTheStore
+ tuneObj: TuneAlgorithm
+ itemsDamaged: int
+ itemsSold: int
+ itemsPurchased: int
+ Clerk(Inventory inventoryObj, Delivery deliveryObj,
CashRegister cashRegisterObj, int damagePercentage, String name): void
+ checkConsecutive(int day): boolean
+ getTime(): String
2
1
List<Item>>
<Class>
CashRegister
ble
ble moneyToAdd): void

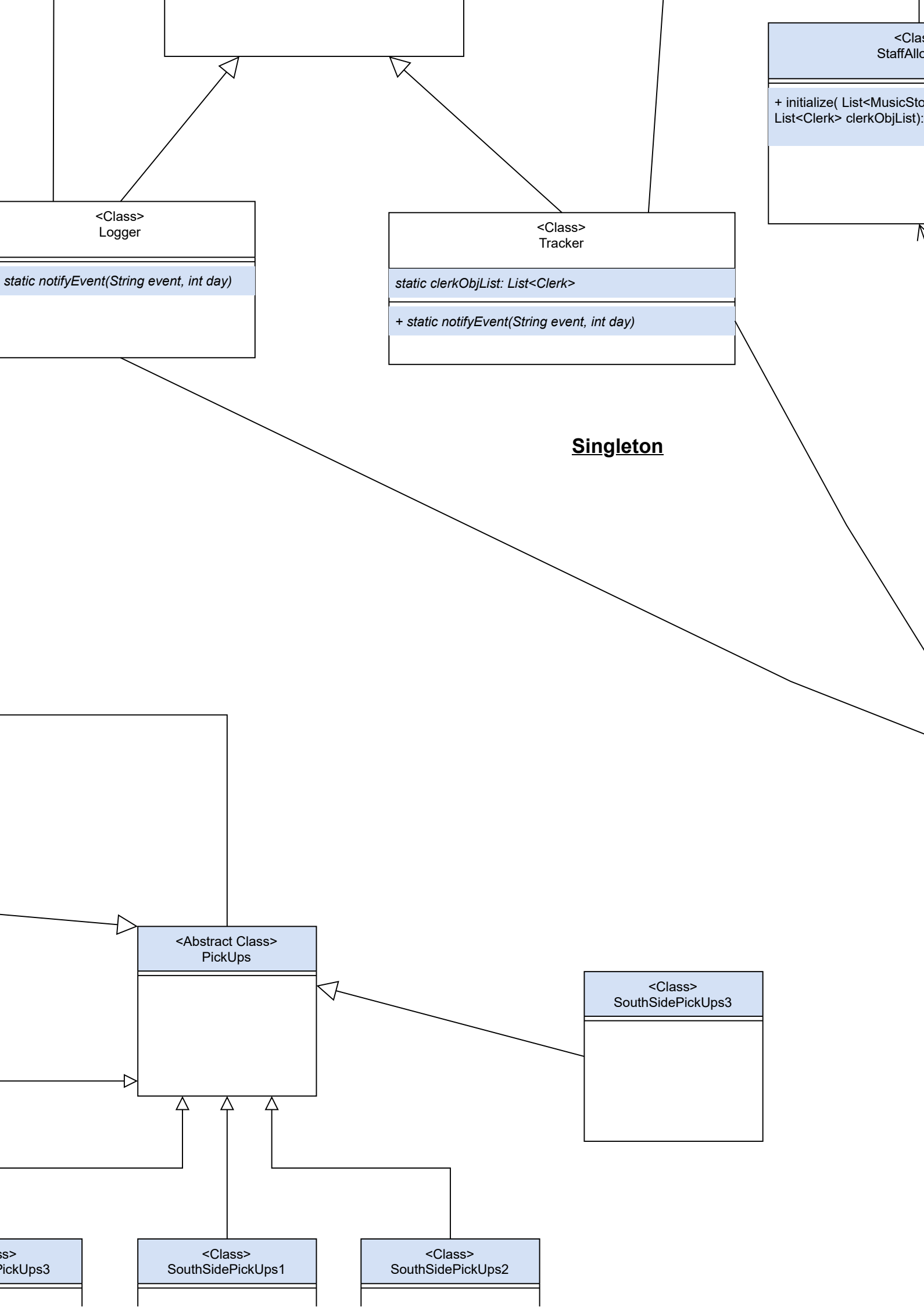

<Class>
ArriveAtStore
+ announce(int day, String name): void
+ checkDelivery(int day, Delivery deliveryObj, Inventory inventoryObj): void
1
<Class>
CheckRegister
1
+ checkBalance(int day, CashRegister cashRegisterObj): void
+ checkBalanceInRegister(int day, CashRegister cashRegisterObj): void
+ announce(int day, double amount): void
<Class>
DoInventory
+ checkInventory(Delivery deliveryObj, CashRegister cashRegisterObj, CheckRegister checkRegisterObj, Inventory inventoryObj, PlaceAnOrder placeAnOrderObj, int currentDay) : int
+ announce(double totalValue): void
1
<Class>
PlaceAnOrder
+ orderItem(String itemStr, Delivery deliverObj, CashRegister cashRegisterObj, CheckRegister checkRegObj,int currentDay):void
1

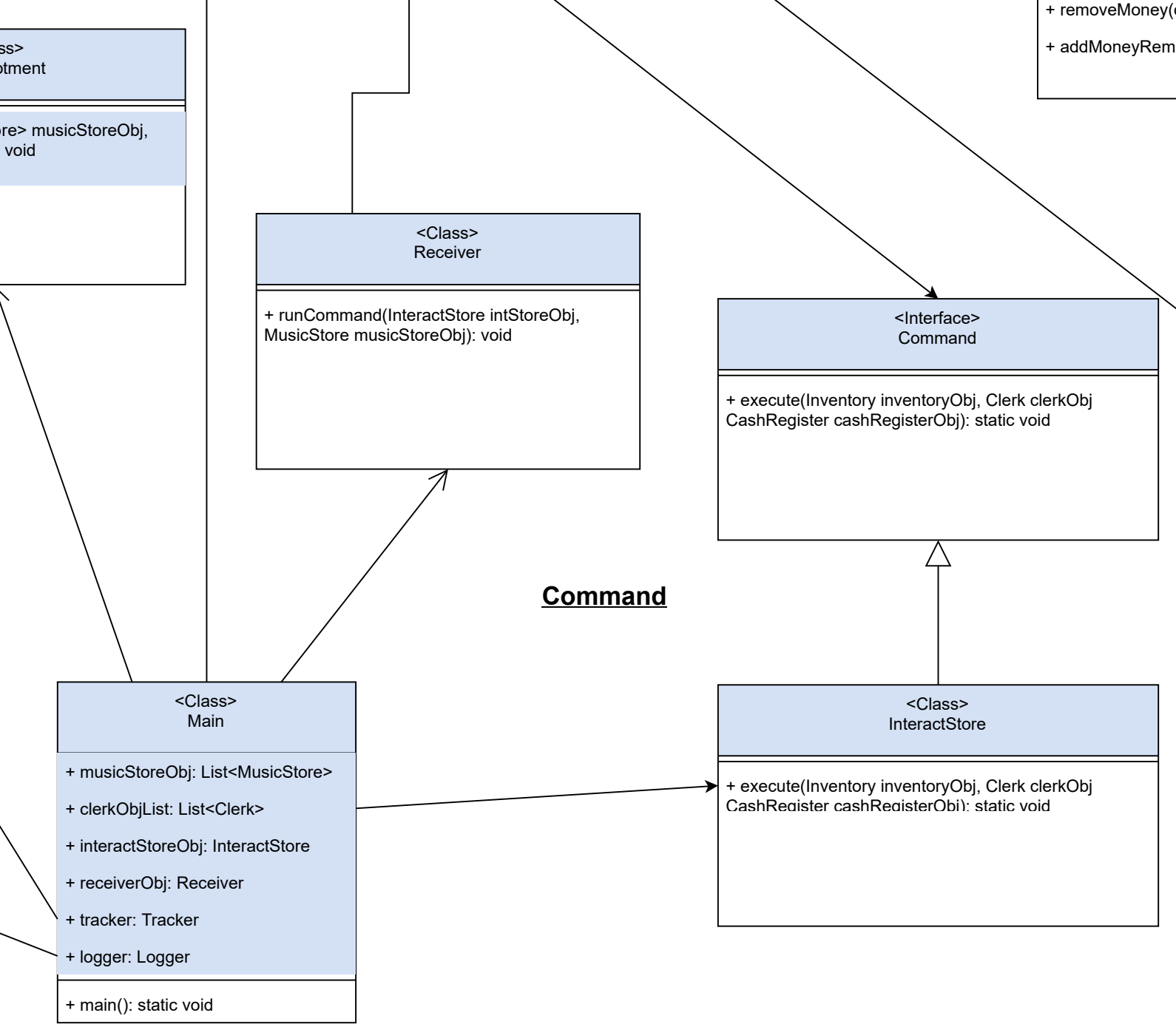

<Class>
NorthSideBridge1

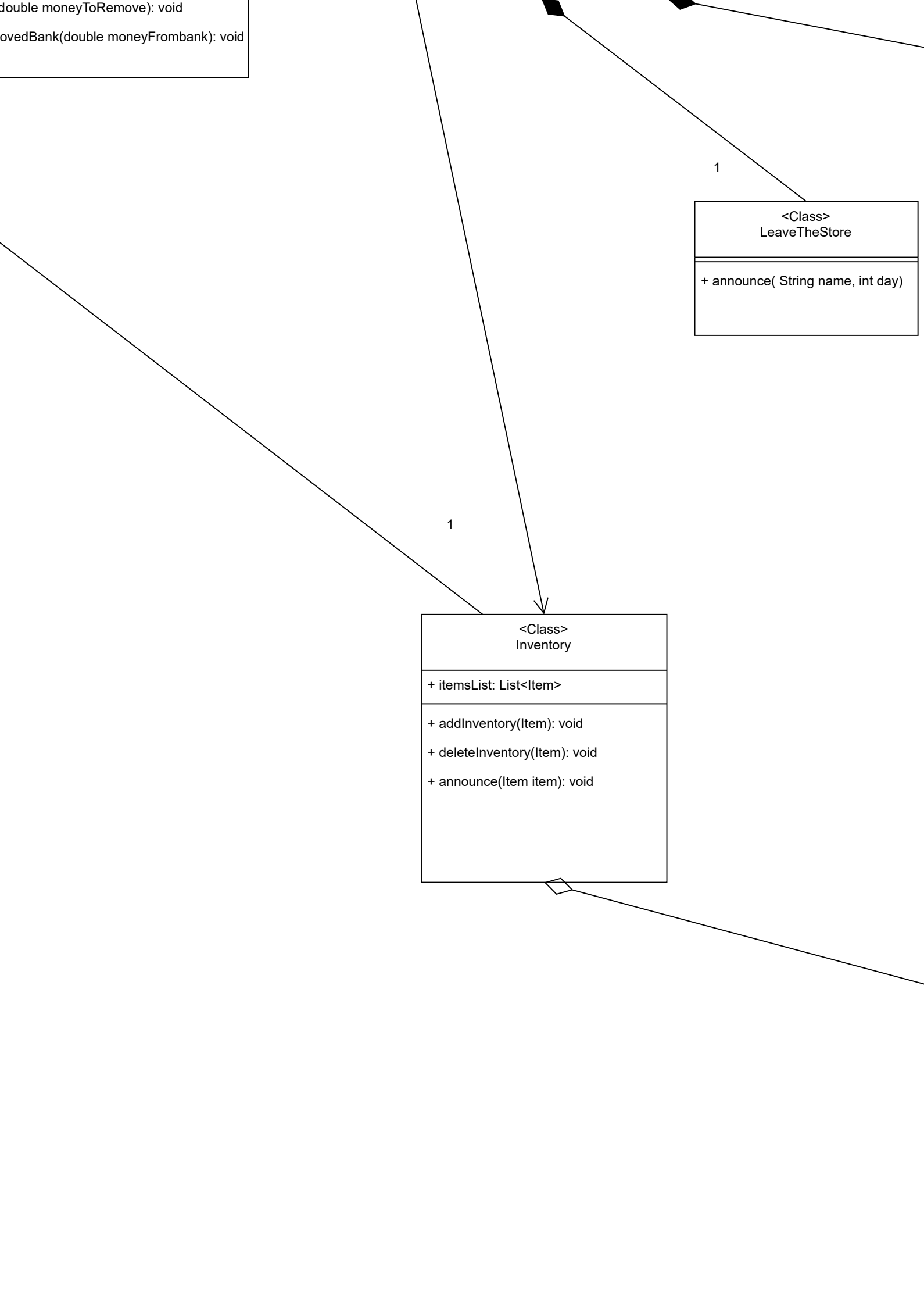

+ createBri
+ createKn
+ createCo
+ createNe
+ createPic
+ createPic

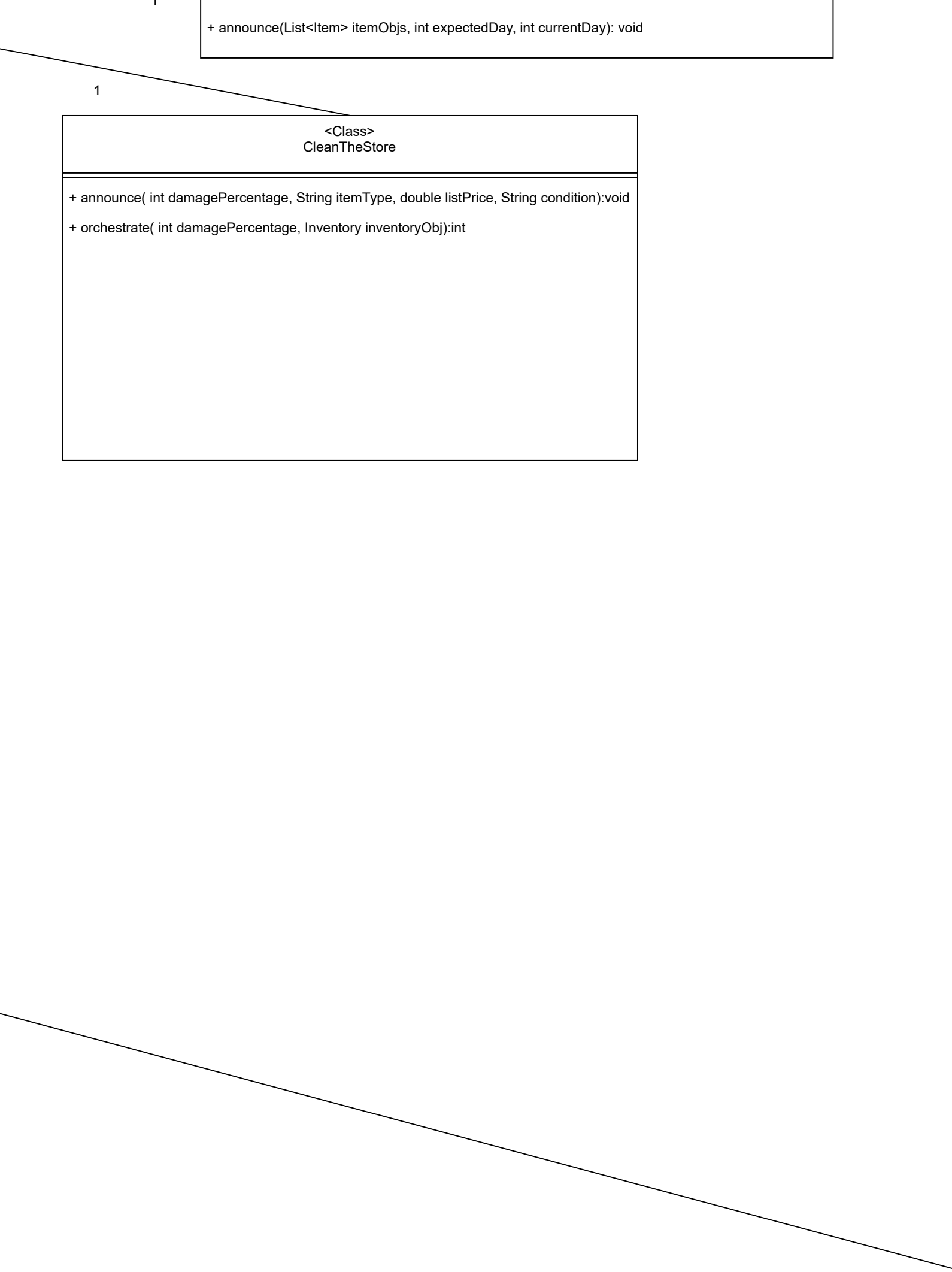

+ createPickups(): Pickups
Abstract Class>
orthSideFactory
dge(): Bridge
obSet(): KnobSet
vers(): Covers
ck(): Neck
kGuard(): PickGuard
kups(): Pickups
<Abstract Class>
SouthSideFactory
+ createBridge(): Bridge
+ createKnobSet(): KnobSet
+ createCovers(): Covers
+ createNeck(): Neck
+ createPickGuard(): PickGuard
+ createPickups(): Pickups
Abstract Factory
<Abstract class>
CustomGuitar
+ price: float


<Class>
NorthSidePickUps1
<Class>
NorthSidePickUps2

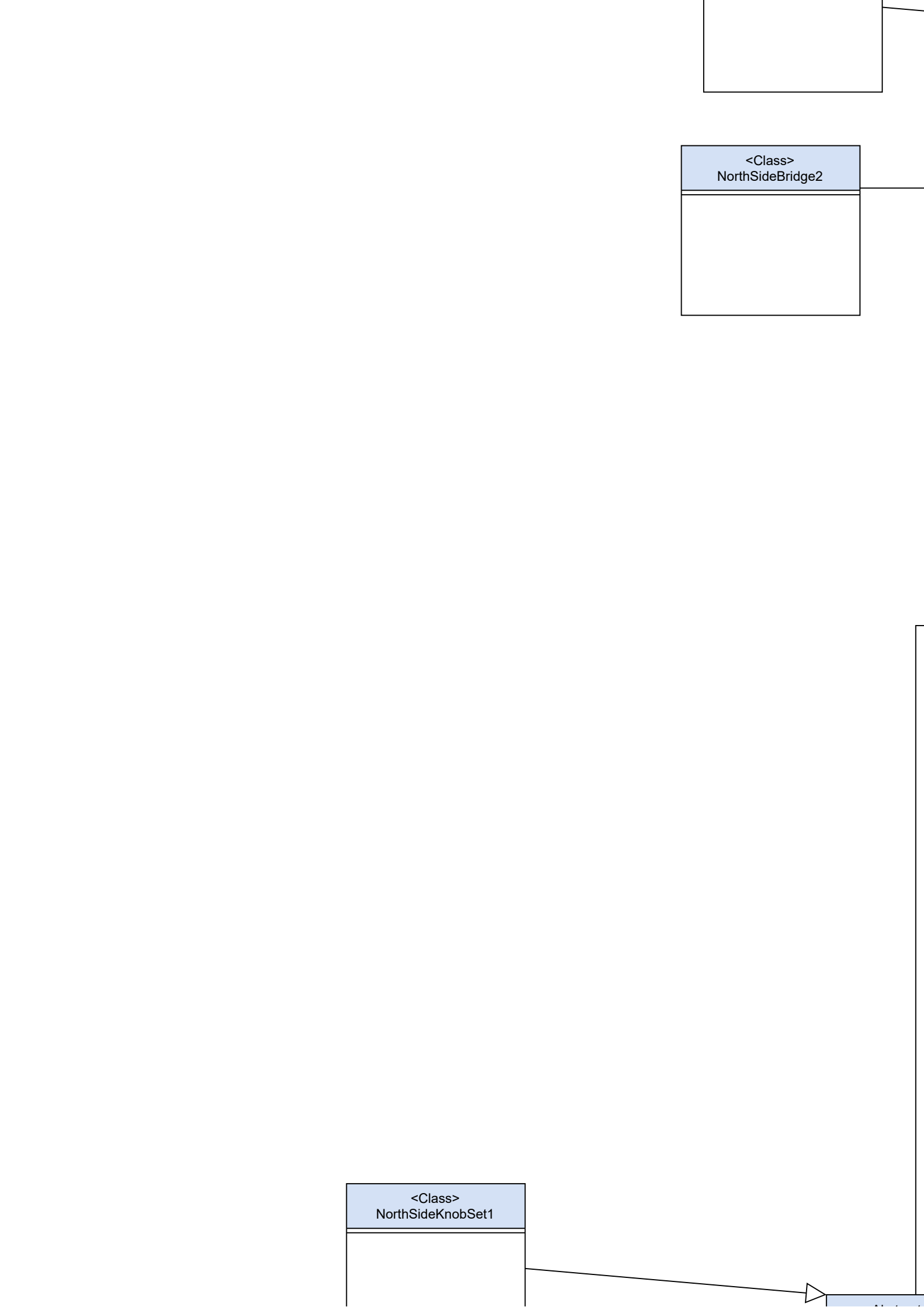

<Class>
Logger
static notifyEvent(String event, int day)
<Class>
Tracker
static clerkObjList: List<Clerk>
+ static notifyEvent(String event, int day)
+ initialize(List<MusicSto
List<Clerk> clerkObjList):
Singleton
<Abstract Class>
PickUps
<Class>
SouthSidePickUps3
<Class>
SouthSidePickUps1
<Class>
SouthSidePickUps2

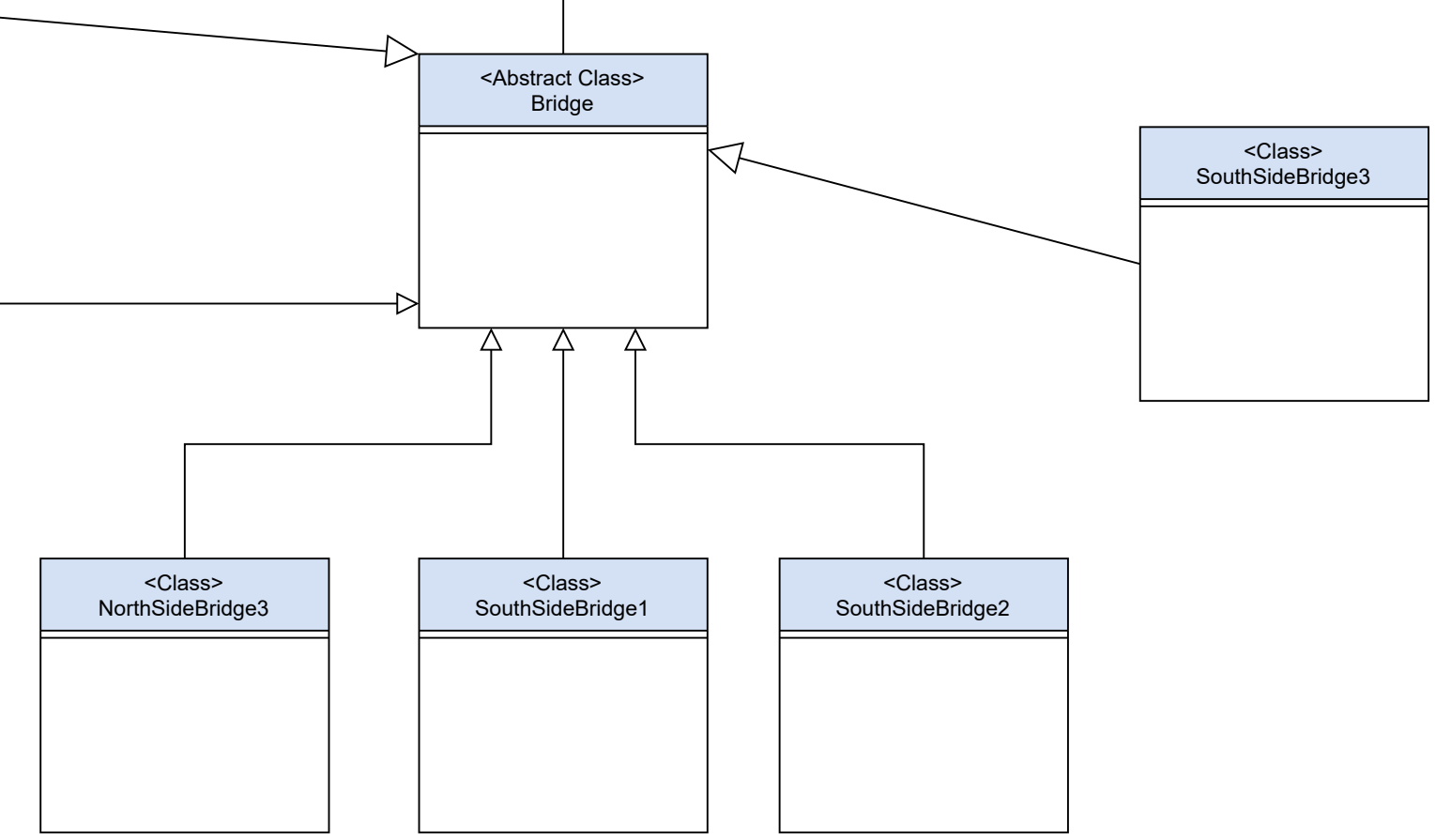

ss>
otment
re> musicStoreObj,
void
+ removeMoney(
+ addMoneyRem
<Class>
Receiver
+ runCommand(InteractStore intStoreObj,
MusicStore musicStoreObj): void
<Interface>
Command
+ execute(Inventory inventoryObj, Clerk clerkObj
CashRegister cashRegisterObj): static void
Command
<Class>
Main
+ musicStoreObj: List<MusicStore>
+ clerkObjList: List<Clerk>
+ interactStoreObj: InteractStore
+ receiverObj: Receiver
+ tracker: Tracker
+ logger: Logger
+ main(): static void
<Class>
InteractStore
+ execute(Inventory inventoryObj, Clerk clerkObj
CashRegister cashRegisterObj): static void

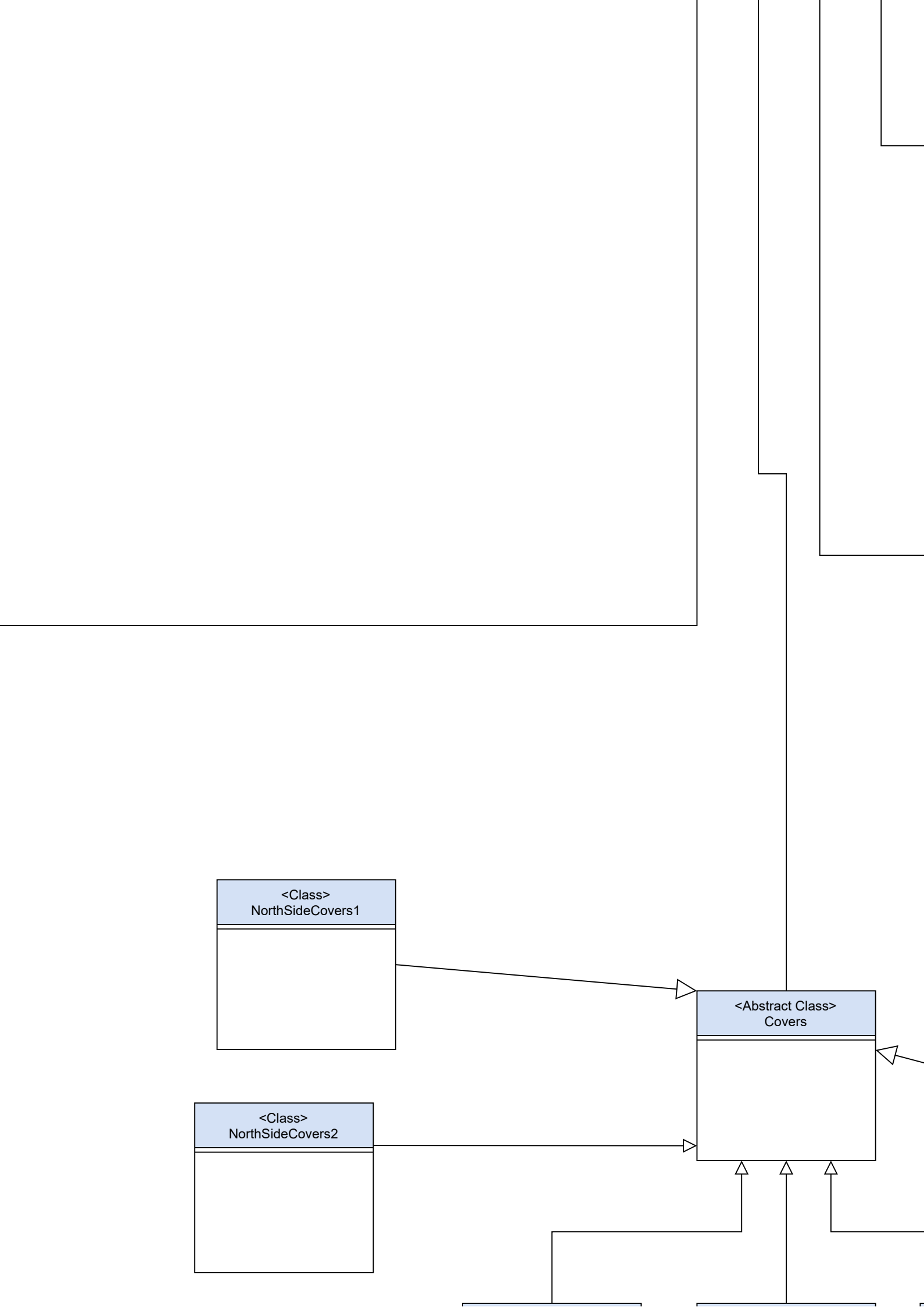

double moneyToRemove): void
ovedBank(double moneyFrombank): void
1
<Class>
LeaveTheStore
+ announce(String name, int day)
1
<Class>
Inventory
+ itemsList: List<Item>
+ addInventory(Item): void
+ deleteInventory(Item): void
+ announce(Item item): void

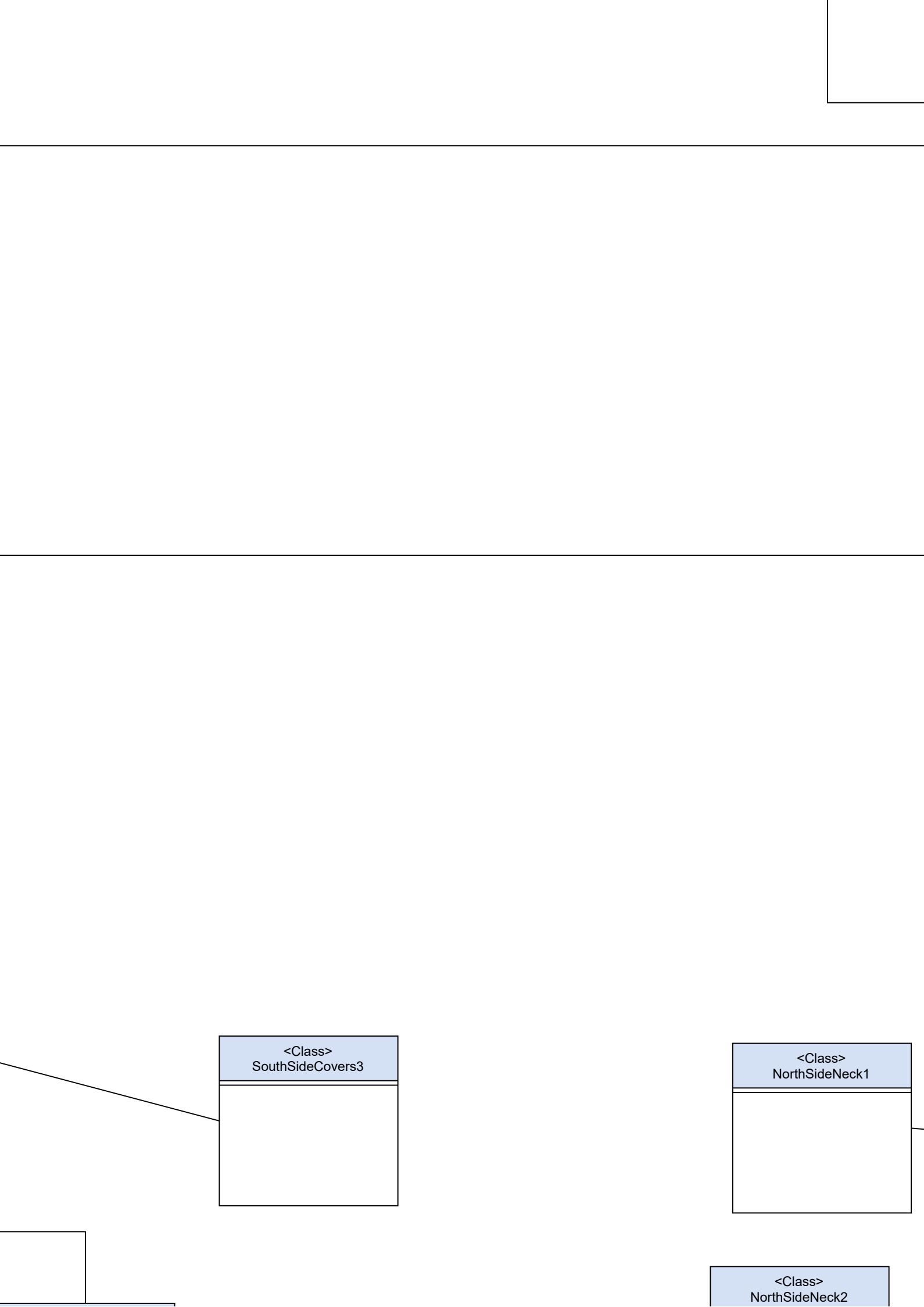

1
+ announce(List<Item> itemObjs, int expectedDay, int currentDay): void
1
<Class>
CleanTheStore
+ announce(int damagePercentage, String itemType, double listPrice, String condition):void
+ orchestrate(int damagePercentage, Inventory inventoryObj):int

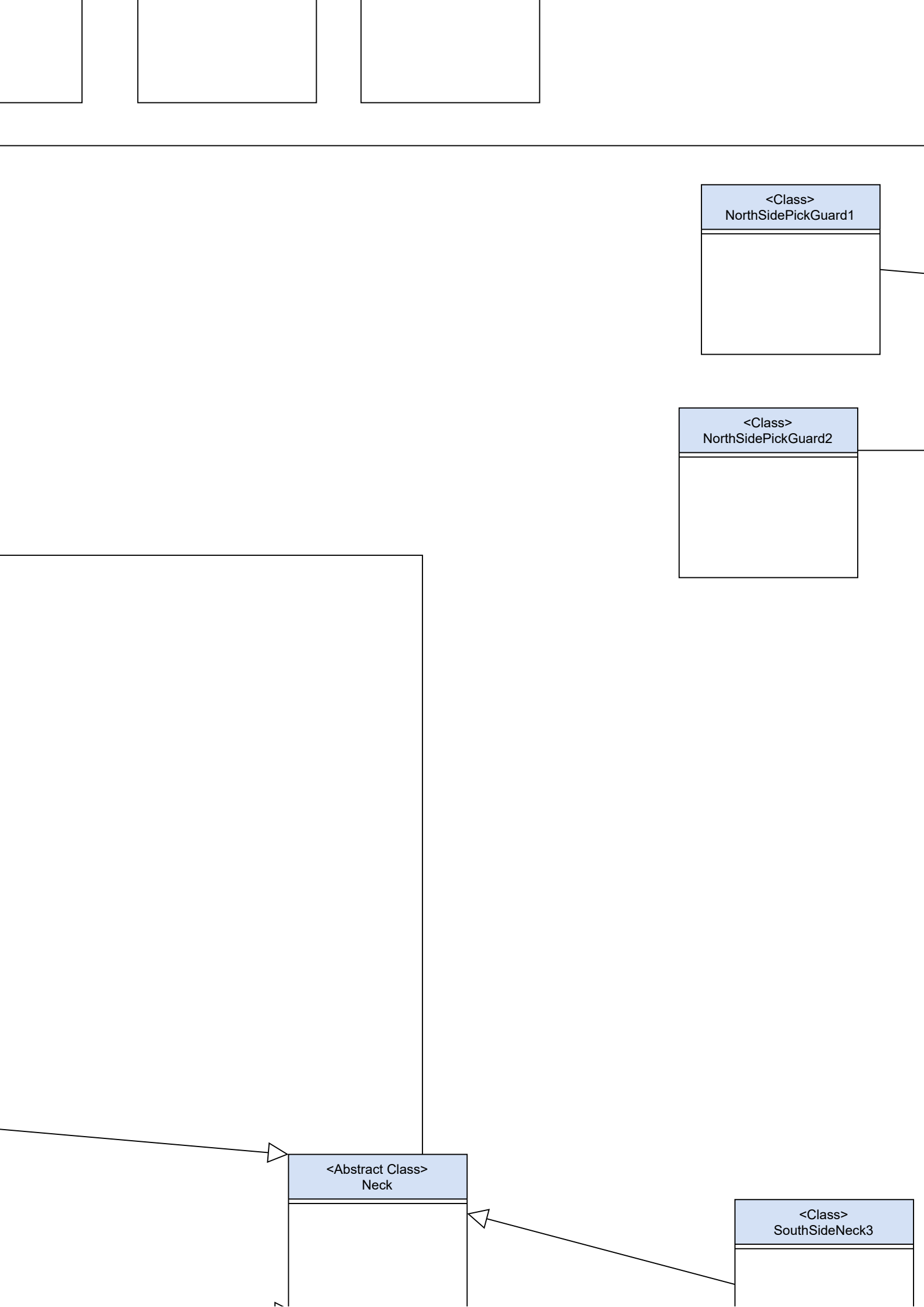
0..*
<Abstract class>

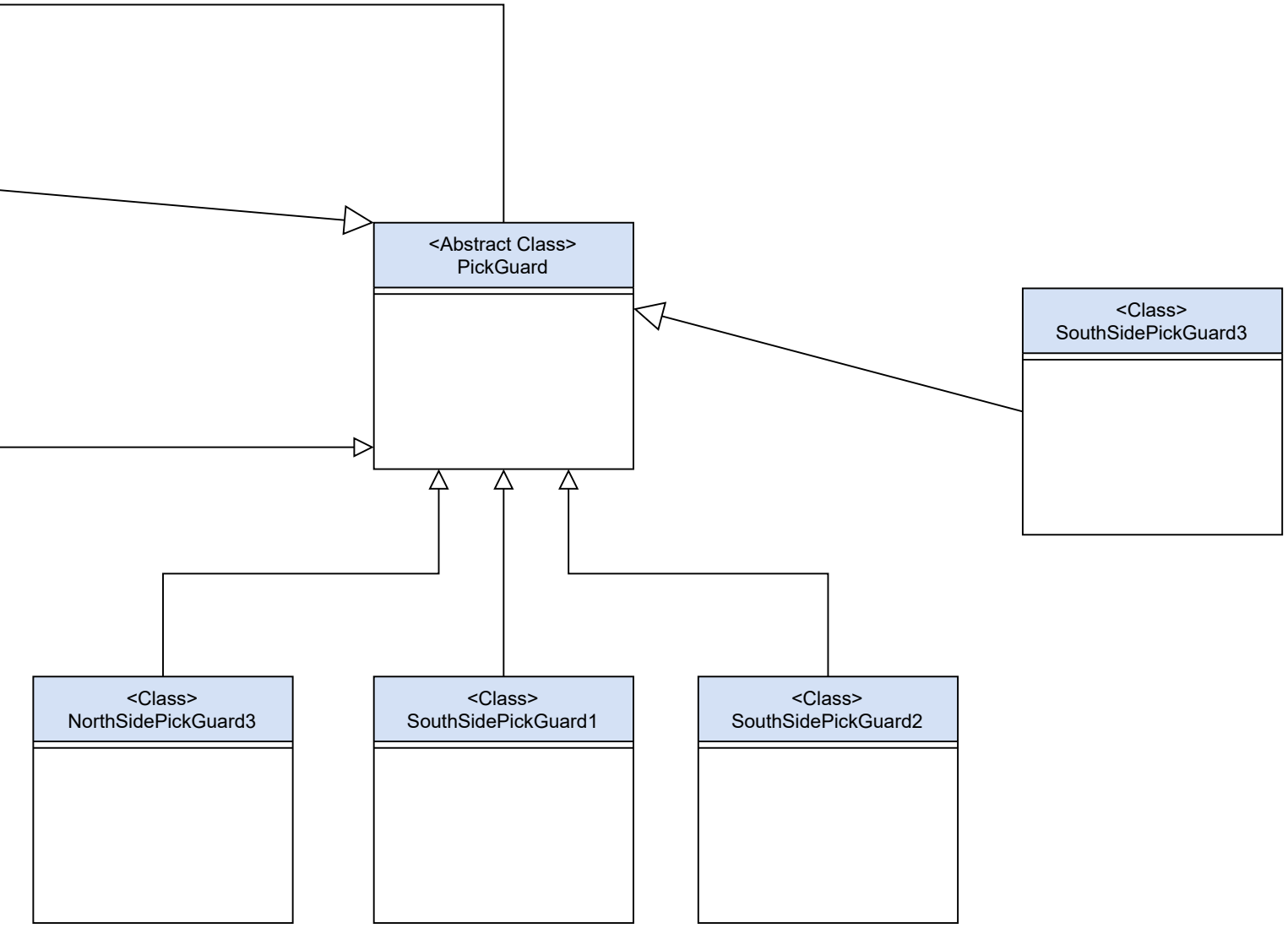

<Class>
NorthSideBridge2
<Class>
NorthSideKnobSet1

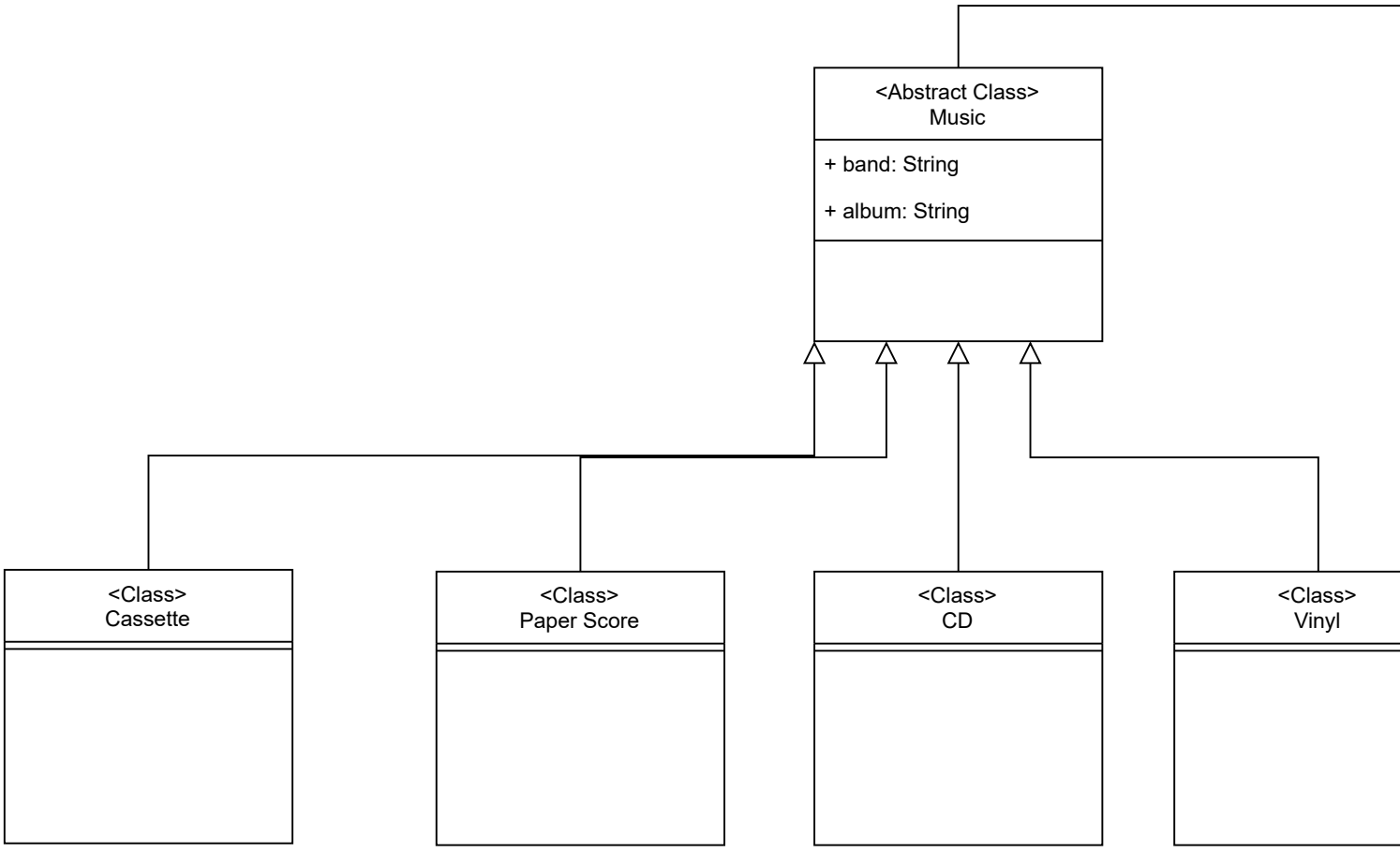

<Abstract Class>
Bridge
<Class>
SouthSideBridge3
<Class>
NorthSideBridge3
<Class>
SouthSideBridge1
<Class>
SouthSideBridge2

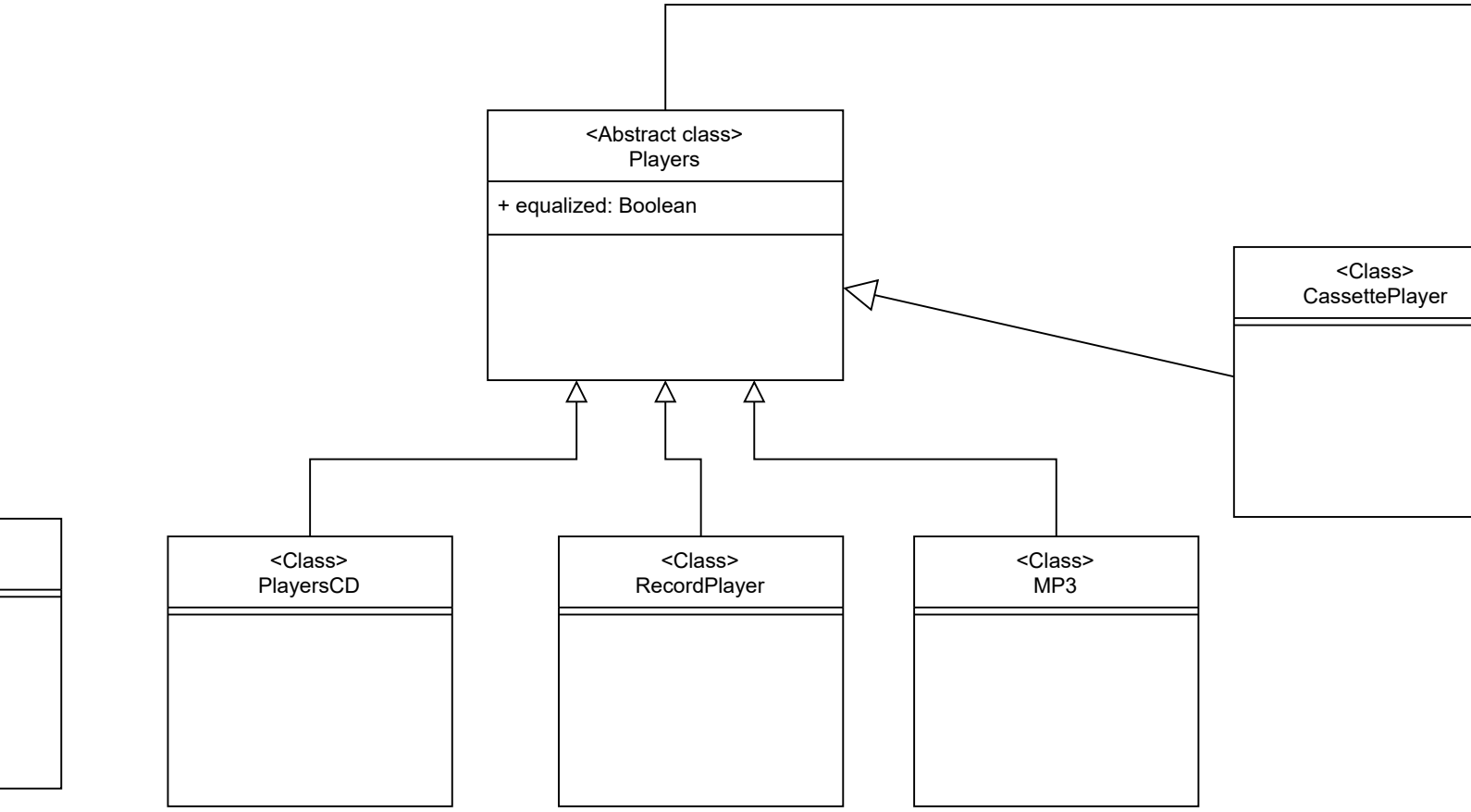

<Class>
NorthSideCovers1
<Class>
NorthSideCovers2
<Abstract Class>
Covers


<Class>
SouthSideCovers3
<Class>
NorthSideNeck1
<Class>
NorthSideNeck2

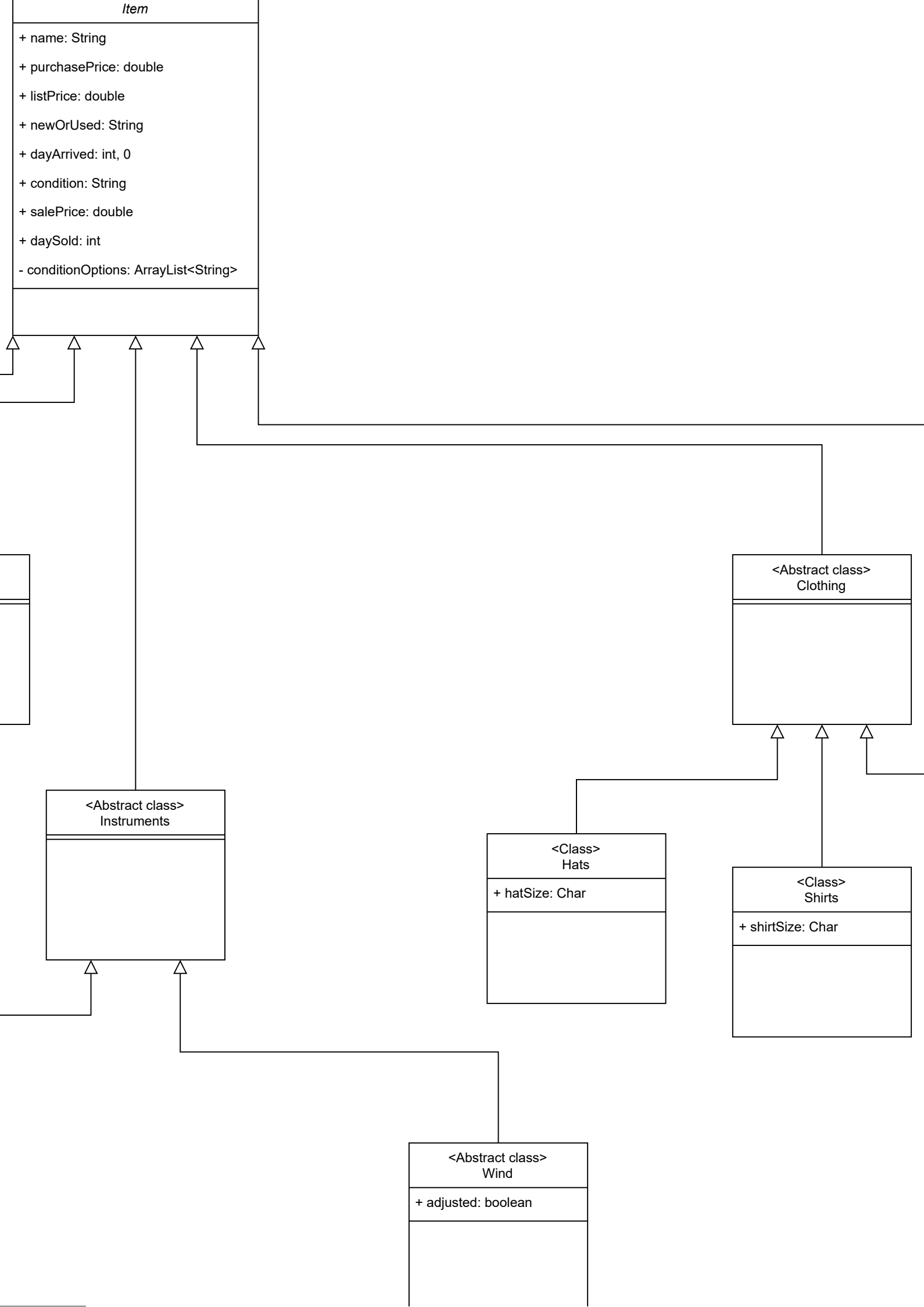

<Class>
NorthSidePickGuard1
<Class>
NorthSidePickGuard2
<Abstract Class>
Neck
<Class>
SouthSideNeck3

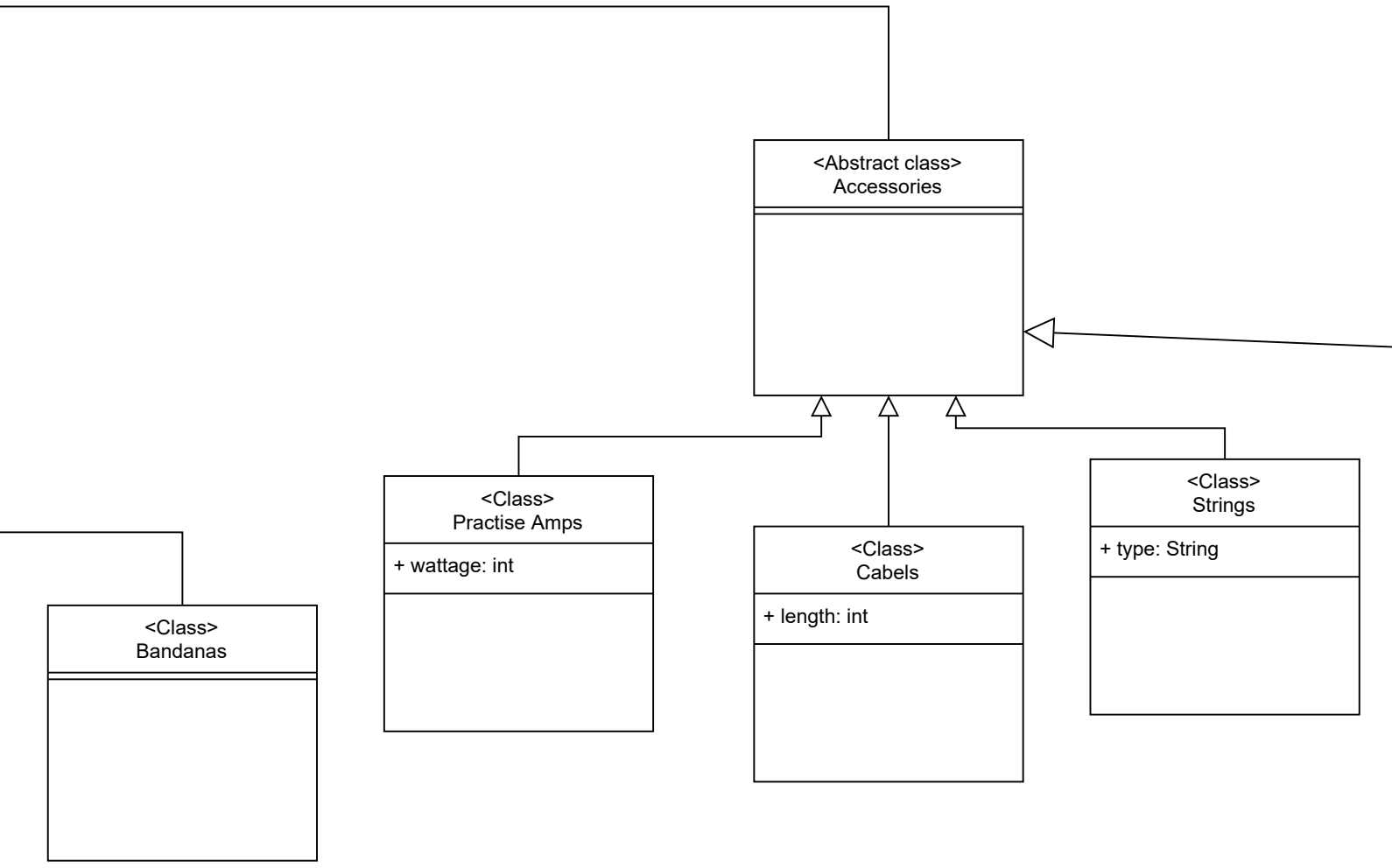

<Abstract Class>
PickGuard
<Class>
SouthSidePickGuard3
<Class>
NorthSidePickGuard3
<Class>
SouthSidePickGuard1
<Class>
SouthSidePickGuard2

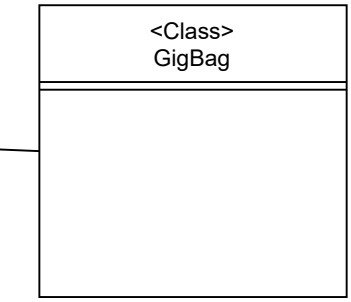

<Abstract Class>
Music
+ band: String
+ album: String
<Class>
Cassette
<Class>
Paper Score
<Class>
CD
<Class>
Vinyl

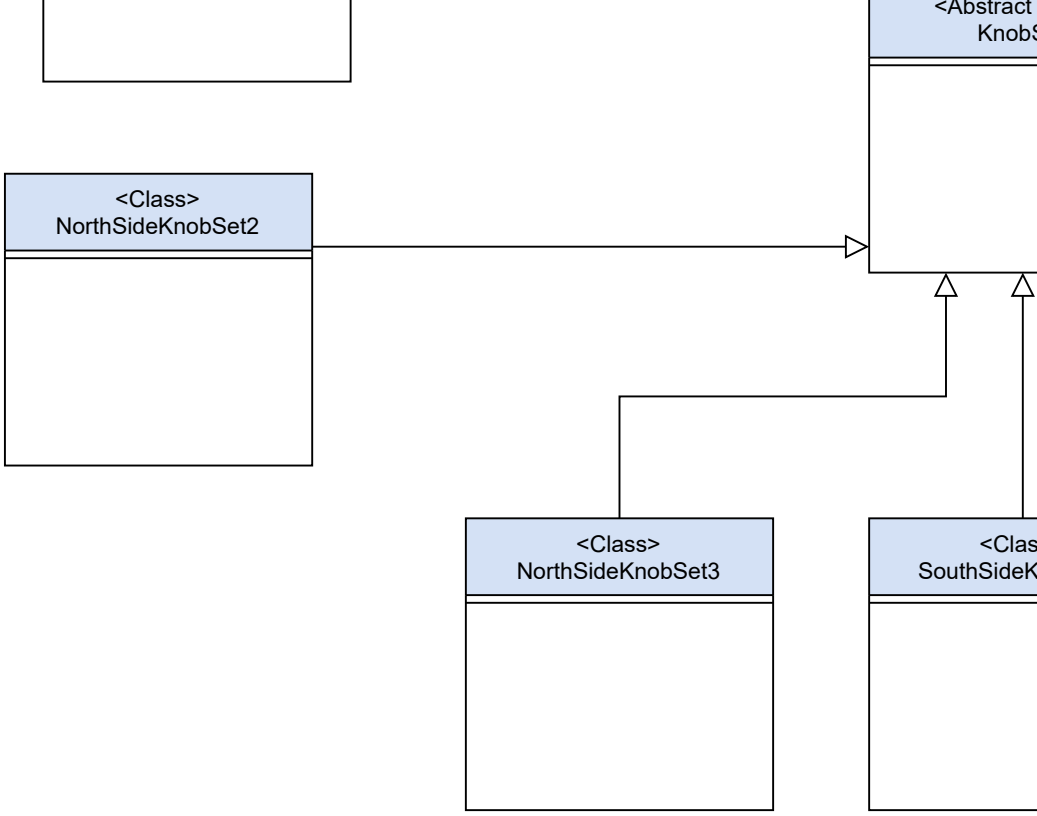

<Abstract class>
Players
+ equalized: Boolean
<Class>
CassettePlayer
<Class>
PlayersCD
<Class>
RecordPlayer
<Class>
MP3

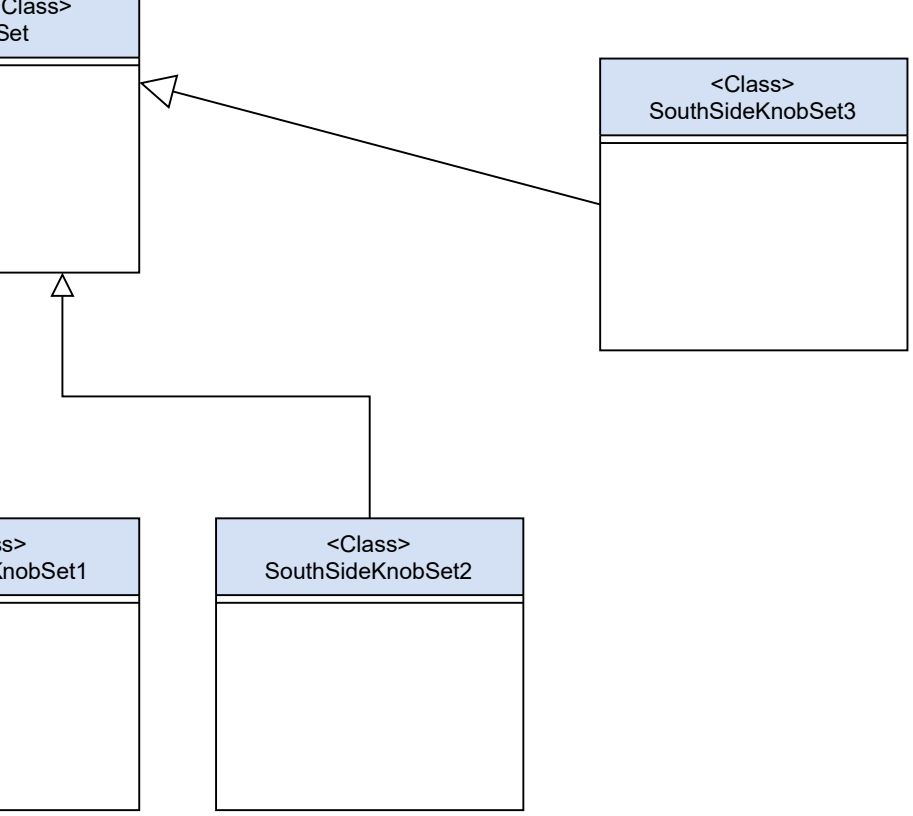
<Abstract class>
Stringed
+ electric: boolean
+ tuned: boolean

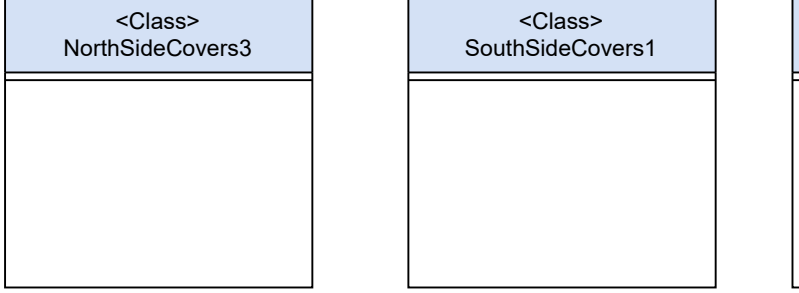

Item
+ name: String
+ purchasePrice: double
+ listPrice: double
+ newOrUsed: String
+ dayArrived: int, 0
+ condition: String
+ salePrice: double
+ daySold: int
- conditionOptions: ArrayList<String>
<Abstract class>
Clothing
<Abstract class>
Instruments
<Class>
Hats
+ hatSize: Char
<Class>
Shirts
+ shirtSize: Char
<Abstract class>
Wind
+ adjusted: boolean

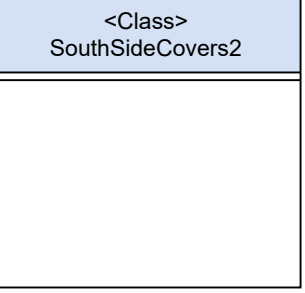

<Abstract class>
Accessories
<Class>
Practise Amps
+ wattage: int
<Class>
Cabels
+ length: int
<Class>
Strings
+ type: String
<Class>
Bandanas

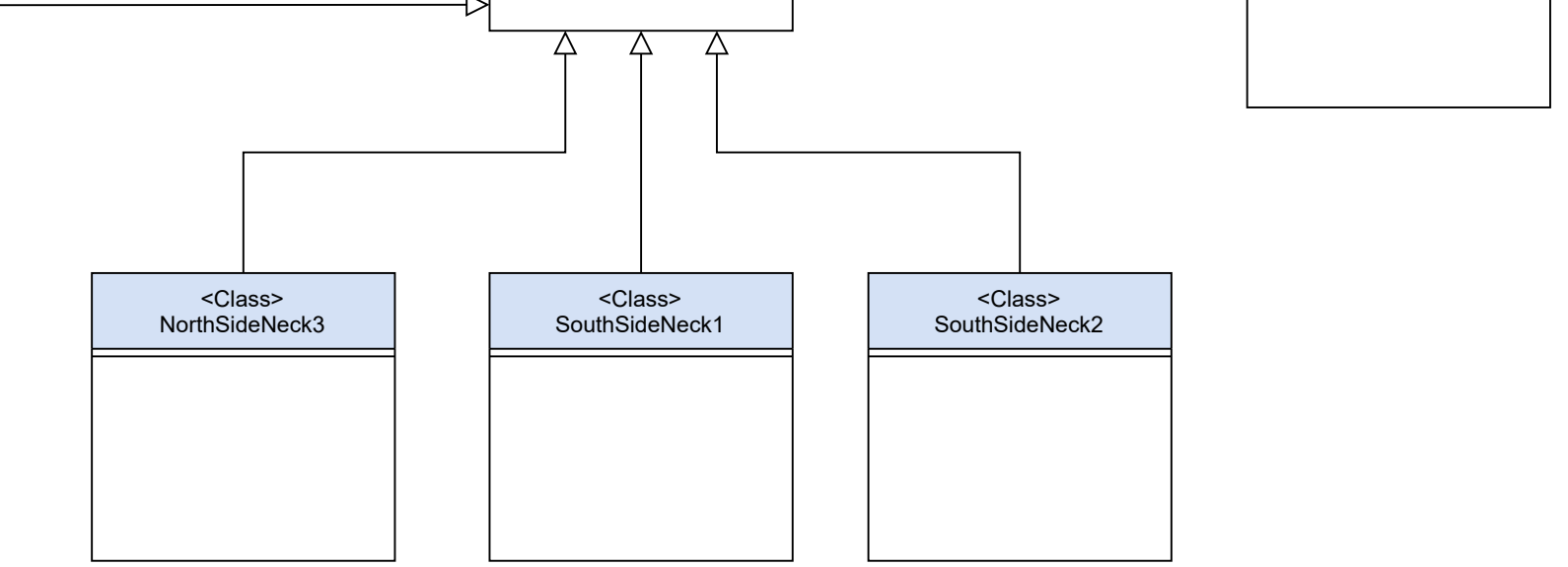

<Class>
GigBag

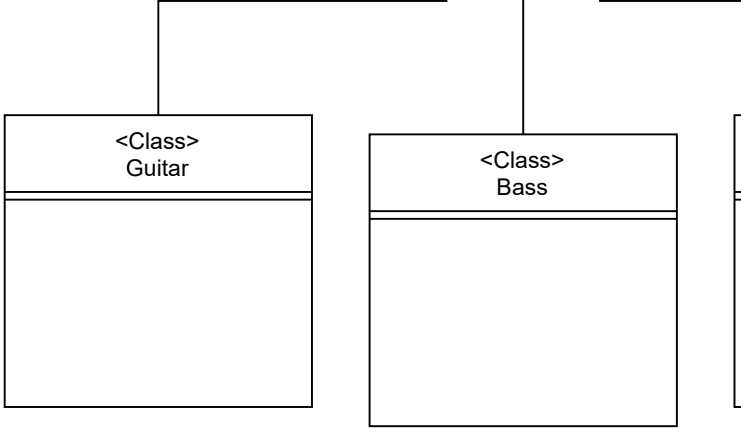

<Class>
NorthSideKnobSet2
<Class>
NorthSideKnobSet3

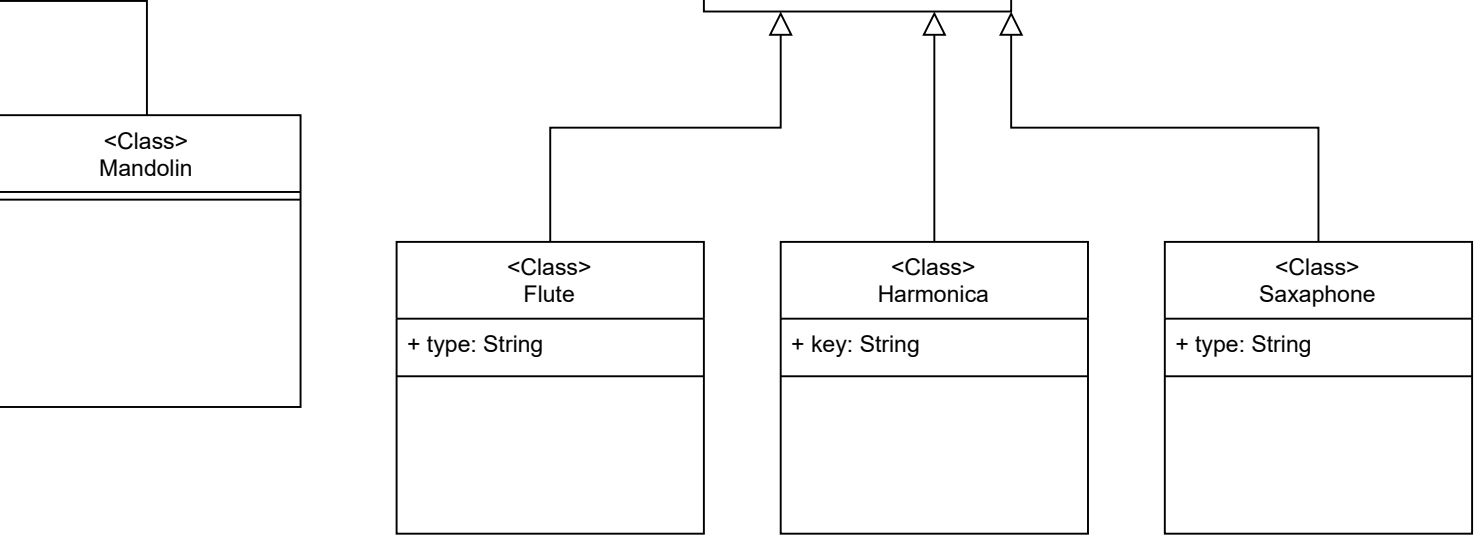

Class>
Set
<Class>
SouthSideKnobSet3
ss>
KnobSet1
<Class>
SouthSideKnobSet2


<Class>
NorthSideCovers3
<Class>
SouthSideCovers1


<Class>
SouthSideCovers2


<Class>
NorthSideNeck3
<Class>
SouthSideNeck1
<Class>
SouthSideNeck2


<Class>
Guitar
<Class>
Bass


<Class>
Mandolin
<Class>
Flute
+ type: String
<Class>
Harmonica
+ key: String
<Class>
Saxaphone
+ type: String