[[1]](#footnote-1)

# Data Structures Essentials

## Assignment 2: GameStonk Share Trading

# Background knowledge

In this assignment you will develop a *very* basic stock market with a stock exchange, brokers that process trades, and trades for different companies’ stocks. In this assignment we have:

* A **listed company** is a company whose shares are bought and sold. Each company has a name, a company code that is an abbreviation of their name, and the current price of their shares.
* A **trade** object contains the number of shares to buy/sell, which company those shares need to be from, and which broker will process that trade.
* A **broker** takes trades from individual users, and puts them in single a queue to process. Because brokers are experts at what they do, they also have a “watchlist” of which companies they recommend people buy shares for. Whilst trades should be processed on a first-in, first-out queue, some unethical brokers might decide to delay when certain trades are processed by putting them at the back of the queue so they could process their own trades first. This is what we’ll be looking at in this assignment.
* A **stock exchange** has a collection of brokers that can buy and sell stocks on it. Each time the stock exchange processes trades, it asks each broker for the next trade in their queue to process. The exchange then processes that trade which causes the company on the trade’s share price to go up or down. This process is then repeated to processes additional trades.

この課題では、証券取引所、取引を処理するブローカー、さまざまな企業の株式の取引など、非常に基本的な株式市場を開発することになります。この課題では、以下のようなものがあります：

- 上場企業は、株式が売買される企業である。各企業は、名前、名前の略称である企業コード、現在の株価を持ちます。

- トレードオブジェクトには、売買する株の数、その株がどの会社から必要なのか、どのブローカーがそのトレードを処理するのかが含まれています。

- ブローカーは、個々のユーザーからトレードを受け取り、それを処理するために単一のキューに入れます。ブローカーは専門家であるため、どの企業の株を買うことを推奨するかという「ウォッチリスト」も持っています。取引は先入れ先出しで処理されるべきですが、一部の非倫理的なブローカーは、自分の取引を先に処理するために、ある取引が処理されるタイミングを遅らせるために、その取引をキューの後方に置くことを決定するかもしれません。この課題では、このようなことを考えることにします。

- 証券取引所には、株式を売買できるブローカーが集まっています。証券取引所が取引を処理するたびに、各ブローカーに処理するキューにある次の取引を問い合わせます。そして、取引所はその取引を処理することで、その取引の対象となった企業の株価を上下させる。このプロセスを繰り返して、さらに取引を処理する。

## What you will learn in this assignment

This assignment will help you understand the following concepts:

* Creating your own linked lists
* Basic use of Priority queues
* Basic use of maps/hash maps
* compareTo() for comparing objects
* Basic exceptions
* Unit testing and Junit

この課題では、以下の概念を理解することができます：

- 独自のリンクリストを作成する

- プライオリティキューの基本的な使い方

- マップ/ハッシュマップの基本的な使い方

- オブジェクトを比較するためのcompareTo()

- 基本的な例外処理

- ユニットテストとJunit

# Academic Integrity

Any work submitted must represent your own knowledge and efforts. This means that whilst you can use Eclipse’s auto-generation functions for making functions, you **cannot** use Chat GPT or other larger code-generation tools, or get direct assistance from others to write your code.

As part of your git history, it should show your knowledge and refactoring of the code to improve it over time, showing that the project submitted was your own work, and was developed over a period of time.

If we have concerns regarding if your submission is your own, we may interview you about your submission and knowledge of the code and how it works, and how to generate similar code.

Whilst you can look at other materials and resources online to gain understanding and knowledge, the code written must be your own and represent how you think the code should function. Whilst you can talk to other students about your submission at a high level (functions, ideas, class relationships, etc), you are not allowed to share code.

提出される作品は、あなた自身の知識と努力によるものでなければなりません。つまり、Eclipseの自動生成機能を使って機能を作ることはできますが、Chat GPTやその他の大規模なコード生成ツールを使ったり、他の人から直接援助を受けてコードを書いたりすることはできません。

git履歴の一部として、提出されたプロジェクトがあなた自身の作品であり、一定期間かけて開発されたことを示すために、時間の経過とともにコードを改善するためのあなたの知識とリファクタリングを示す必要があります。

提出されたものがあなた自身のものであるかどうかについて懸念がある場合、提出されたコードとその動作方法、および同様のコードを生成する方法に関するあなたの知識についてインタビューさせていただくことがあります。

理解や知識を得るためにオンラインで他の資料やリソースを見ることはできますが、書かれたコードはあなた自身のものであり、コードがどのように機能すべきかを表している必要があります。提出したコードについて、他の生徒と高いレベル（関数、アイデア、クラスの関係など）で話すことはできますが、コードを共有することは許可されていません。

# Getting Started

This assignment is an Eclipse project with existing class files that you will be required to complete in varying forms by changing method bodies, return types, creating getters, and setters. In addition to the classes you will be editing, there are a number of Junit tests that are used by a marking program to give you and indication of how well your code is working. Any changes you make to the class files listed below should be in-line with this requirement specification, any documentation in the code itself, and passing the tests. You do not need to edit any of the test files, or the marker, we will be using our own version of them for marking. However, you may find it useful to edit the test files for debugging purposes.

この課題はEclipseのプロジェクトで、既存のクラスファイルがあり、メソッド本体、戻り値の型、ゲッター、セッターの作成など、様々な形で完成させる必要があります。編集するクラスに加えて、あなたのコードがどの程度うまく機能しているかを示すために、採点プログラムで使用されるいくつかのJunitテストがあります。 以下のクラスファイルに加える変更は、この要求仕様、コード自体のドキュメント、およびテストに合格することに沿ったものでなければなりません。テストファイルやマーカーを編集する必要はありません。ただし、デバッグのためにテストファイルを編集しておくと便利かもしれません。

## Getting it running

1. Download the ZIP from the course web page and unzip.
2. In Eclipse > File > Open Projects from File System and open the project
3. There are two ways you can run the tests:
   1. The first will run the entire marking program and give you an overall score and marks. Go into the AssignmentMarker.java file in the Junit package and run the file. The console will show you the output from the test marker. Note that you’ll see lots of things going wrong and it doesn’t run completely! That’s ok, we’ll work on that. As you complete more of the code, more of the assignment marker will be able to run.
   2. The second way is to go into one of the Junit test files specifically, e.g. open up “ListTest.java” and hit run. You should see Junit tests appear in your window and console, showing the output of each test run:

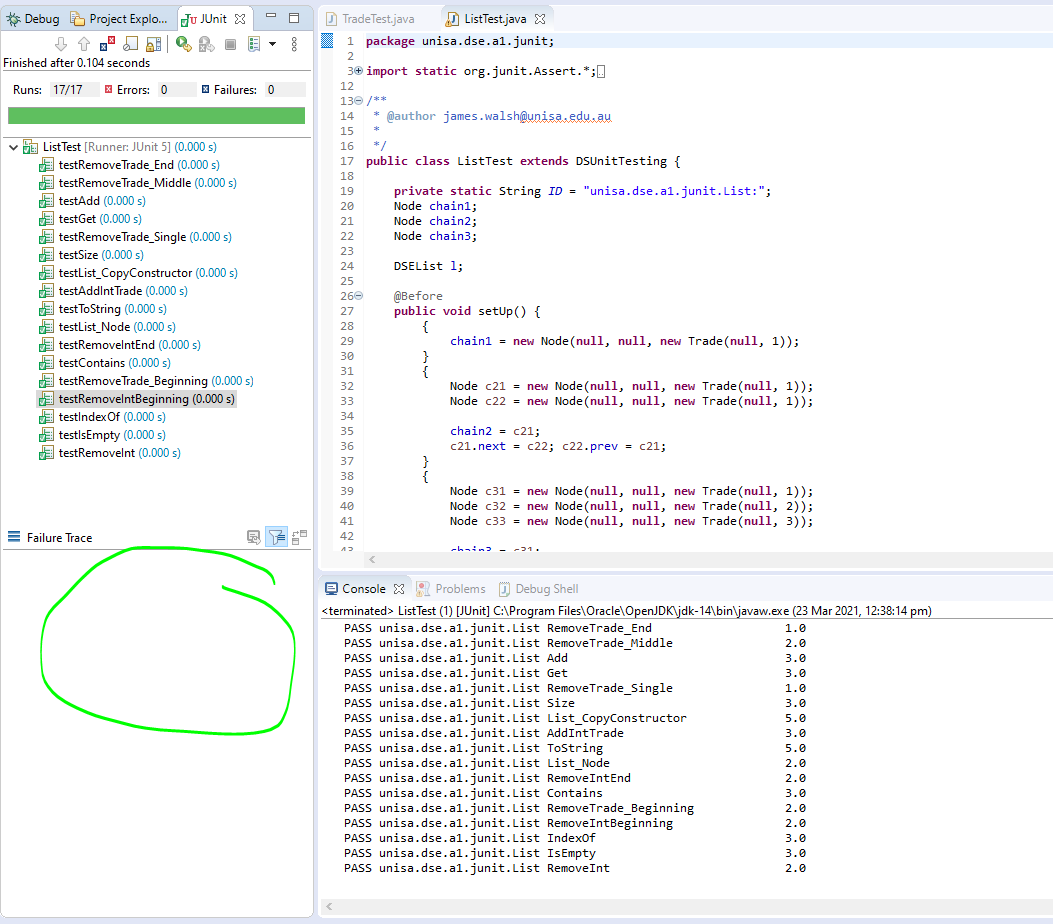
1. 講座のWebページからZIPをダウンロードし、解凍してください。

2. Eclipseの「ファイル」→「ファイルシステムからプロジェクトを開く」で、プロジェクトを開く

3. テストを実行する方法は2つあります：

3.1. 1つ目は、採点プログラム全体を実行し、総合得点と点数を出します。JunitパッケージのAssignmentMarker.javaに入り、このファイルを実行します。コンソールには、テストマーカーからの出力が表示されます。たくさんのことがうまくいかず、完全に実行されないことがわかりますので注意してください！それは大丈夫です、それに取り組みましょう。より多くのコードを完成させると、より多くの課題マーカーが実行できるようになります。

3.2. 2つ目の方法は、Junitのテストファイルの1つに特別に入ることです。例えば、「ListTest.java」を開いて、実行を押してみてください。ウィンドウとコンソールにJunitテストが表示され、各テスト実行の出力が表示されるはずです：



In the bottom left section of the screenshot above it will show a stack trace for any test you click on that failed. This will be what you can use to start debugging what’s happening.

Using the above, you can run the whole marker, or just an individual set of test for a specific class, which makes it easier to focus on one class at a time. As you go through and complete functionality in each class, run the tests for that class to see how successful you were in passing those tests.

Please note that whilst we have provided SOME tests, it does not mean the tests cover everything. Additional tests will be used to mark you assignment, so please review your code and make sure it’s not just passing the tests you have, but also handling any other scenarios it should be.

上のスクリーンショットの左下には、クリックしたテストが失敗した場合のスタックトレースが表示されます。これは、何が起きているのかデバッグを開始するために使用することができるものです。

上記の方法で、マーカー全体を実行することもできますし、特定のクラスのテストだけを個別に実行することもできるので、一度に1つのクラスに集中することが容易になります。各クラスの機能を完成させたら、そのクラスのテストを実行し、テストに合格できたかどうかを確認します。

一部のテストは用意されていますが、そのテストがすべてをカバーしているわけではないことに注意してください。追加テストはあなたの課題を採点するために使用されます。したがって、あなたのコードを見直し、あなたが持っているテストに合格するだけでなく、他のシナリオを処理する必要があることを確認してください。

## Getting it into GitHub Desktop

GitHub Desktop is GitHub’s tool for managing code using [Git](https://en.wikipedia.org/wiki/Git), which is a source control tool that, in short, allows you to take snapshots of your code as you, and easily merge it with what others are working on. For this assignment, you’ll just be using git locally on your own machine to take commits (i.e. a snapshot) of your code as you go through development.

1. Download and install GitHub Desktop from <https://desktop.github.com/>
2. One installed and open, go File menu > “Open Local Repository”
3. Open the root of the A2 folder you downloaded. You might see a “.git” folder in your download, you want to open the folder that contains this “.git” folder.
   1. If you get a warning about “This directory does not appear to be a Git repository” you’ve selected the wrong folder.
4. In the top left you can now switch into the “History” tab and see the initial commit, what files it contained, and what changes were in those files.
5. You can, if needed, revert back to any of your prior commits if you make a mistake and want to “undo” it, however BEWARE! If you do rollback, make sure you don’t lose work you want to keep!

You MUST commit your code at regular intervals. This will teach you to commit your code as you get things working. What you commit doesn’t have to be perfect, and you can refine your code later on, but it is important to commit your work as you go so that you have snapshots showing it’s development over time, and your trial-and-error as you learn and improve on it.

**Your git repository and its history are part of your marks!** Make sure to commit each time you get things to a good point.

There are lots of tutorials online for using GitHub Desktop, with their official documentation for making commits available at <https://docs.github.com/en/desktop/contributing-and-collaborating-using-github-desktop/making-changes-in-a-branch/committing-and-reviewing-changes-to-your-project>. \\

GitHub Desktopは、GitHubが提供するGitを使ったコード管理ツールです。Gitはソース管理ツールで、簡単に言うと、自分のコードのスナップショットを取りながら、他の人が作っているコードと簡単にマージできるようにするツールです。この課題では、自分のマシン上でローカルにgitを使い、開発を進めながらコードのコミット（つまりスナップショット）を取るだけです。

1. https://desktop.github.com/ から GitHub Desktop をダウンロードし、インストールします。

2. インストール後、ファイルメニュー > "ローカルリポジトリを開く "で開きます。

3. ダウンロードしたA2フォルダのルートを開く。ダウンロードしたフォルダの中に「.git」フォルダがあるかもしれませんが、この「.git」フォルダが含まれるフォルダを開きます。

3.1. このディレクトリはGitリポジトリではないようです」という警告が表示された場合は、フォルダの選択を間違えています。

4. 左上の「History」タブで、最初のコミット、その中に含まれるファイル、そしてそのファイルに加えられた変更を確認できます。

5. 5.間違いがあって「元に戻したい」場合、必要に応じて以前のコミットに戻すことができますが、注意してください！ロールバックをする場合は、残しておきたい作業を失わないように注意してください！

定期的にコードをコミットする必要があります。これは、物事がうまくいくようになったときにコードをコミットすることを学ぶものです。コミットしたものが完璧である必要はありませんし、後でコードを改良することもできます。しかし、作業を進めながらコミットすることは重要で、時間の経過に伴う開発のスナップショットや、学習と改良のための試行錯誤を示すことができるようになります。

gitリポジトリとその履歴は、あなたの印の一部です！物事が良い方向に進んだら、必ずコミットしてください。

GitHub Desktopの使い方については、オンライン上にたくさんのチュートリアルがあります。コミットを行うための公式ドキュメントは、https://docs.github.com/en/desktop/contributing-and-collaborating-using-github-desktop/making-changes-in-a-branch/committing-and-reviewing-changes-to-your-project にあります。

# Completing the assignment

Please review the below, along with the actual source files comments, and the tests themselves for additional information.

You may be required to change the parameter types or return types of function calls as part of the assignment. You’ll need to use your understanding of how the code should operate to make the required changes.

以下の内容、実際のソースファイルのコメント、およびテストそのものを確認して、追加情報を確認してください。

課題の一環として、関数呼び出しのパラメータ型や戻り値の型を変更する必要がある場合があります。その際、コードの操作方法について理解した上で、必要な変更を行う必要があります。

## Step One – DSEList.java

In part 1, you will create your own implementation of the Java LinkedList Collections class. There is an «interface» provided for List. In Step Two, you will then use your Linked List implementation as the basis for making a generic version that can store any type of object.

DSEList will extend the List class defined in List.java. The implementation will be a double-linked list and must implement the abstract methods from List.java.

DSEList should have one data member: public Node head. Others can be added if you require them.

The Node objects used by the list store basic String objects.

第1部では、JavaのLinkedList Collectionsクラスの実装を自分で作成します。Listには「インターフェース」が用意されています。ステップ2では、作成したLinked Listの実装をもとに、あらゆるタイプのオブジェクトを格納できる汎用バージョンを作成します。

DSEListは、List.javaで定義されたListクラスを拡張します。実装は二重リンクリストとし、List.javaの抽象メソッドを実装する必要があります。

DSEListは1つのデータメンバを持つべきである：public Node head. 必要であれば、他のものを追加することができます。

リストが使用するNodeオブジェクトは、基本的なStringオブジェクトを格納します。

Implement the following methods in the List class:

* Constructor: implement a blank constructor which takes no parameters.
* Constructor: implement a constructor accepting one Node (containing a String object). The constructor should set head to the given Node.
* Copy constructor: implement a copy constructor accepting a DSEList object. The copy constructor should perform a deep copy of the DSEList passed to the constructor: the new DSEList should not contain references to the Node objects in the second DSEList. (The two DSELists should be independent: changing the contents of Node objects in one DSEList should not affect the other).
* public boolean add(String obj): The add method should append the specified object to the end of the List.
* public boolean isEmpty()
* public int size()
* public String toString(): this should return a String created by concatenating each Nodes toString(). A single space: ‘ ’ should be inserted between each Nodes toString(). No trailing space should be inserted. For example, if the list contains 3 Node objects, an appropriate toString() return value could be ‘1 2 3’, but not ‘123’ or ‘1 2 3 ’ [note the trailing whitespace]. For further details, refer to the unit tests supplied with the assignment.
* public boolean equals(Object other): two DSEList objects are equal if they contain the same Strings in the same order.

Listクラスで以下のメソッドを実装する：

- コンストラクタ：パラメータを受け取らない空白のコンストラクタを実装する。

- コンストラクタ：1つの Node（Stringオブジェクトを含む）を受け入れるコンストラクタを実装する。コンストラクタは、head を与えられた Node に設定する。

- コピーコンストラクタ：DSEListオブジェクトを受け入れるコピーコンストラクタを実装する。コピーコンストラクタはコンストラクタに渡されたDSEListのディープコピーを実行する必要がある。(2つのDSEListは独立であるべきで、一方のDSEListのNodeオブジェクトの内容を変更しても、他方のDSEListには影響しないはずです)。

- public boolean add(String obj)： addメソッドは、指定されたオブジェクトをListの末尾に追加する必要があります。

- public boolean isEmpty()

- public int size()

- public String toString(): 各ノードのtoString()を連結した文字列を返す。各Nodes toString()の間には、1つのスペース: ' を挿入する必要があります。末尾のスペースは挿入してはいけません。例えば、リストに 3 つの Node オブジェクトが含まれている場合、適切な toString() の返り値は '1 2 3' ですが、'123' や '1 2 3 ' ではありません [後続の空白に注意]。詳細については、課題とともに提供されるユニットテストを参照してください。

- public boolean equals(Object other): 2つのDSEListオブジェクトは、同じ順番で同じ文字列を含む場合、等しいと言えます。

## Step Two – DSEListGeneric.java

The second part of the assignment is to take the code you have written for your list and make it generic. This will allow the list’s nodes to store ANY objects, not just Strings. **You should have to write almost no code for this step,** instead you should only need **copy and refactor your existing code** from the functions in DSEList.java into DSEListGeneric.java, and make very small changes to it to enable generics. The generic list should use the NodeGeneric class for its nodes.

DSEListGeneric should support the same functions as above for DSEList, however any references to the String type that the list stores should be replaced with the generic type that’s passed in when the generic list is created at runtime. Again, you should have to add no additional logic for this step, rather you are just copying and refactoring your existing DSEList functions into DSEListGeneric with very minor changes to the signature of methods and their contents. As a hint for converting a class from a non-generic class into a generic class, compare the Node and NodeGeneric classes. They achieve the same thing, however the second one supports generics.

課題の2つ目は、リスト用に書いたコードを汎用化することです。これにより、リストのノードに文字列だけでなく、あらゆるオブジェクトを格納することができるようになります。DSEList.javaの関数からDSEListGeneric.javaに既存のコードをコピーしてリファクタリングし、ジェネリックを有効にするために非常に小さな変更を加えるだけでよいのです。 ジェネリックリストは、そのノードにNodeGenericクラスを使用する必要があります。

DSEListGenericは、上記のDSEListと同じ関数をサポートする必要がありますが、リストが格納するString型への参照は、実行時にジェネリックリストが作成されるときに渡されるジェネリック型に置き換えられる必要があります。このステップでも追加のロジックは必要ありません。むしろ、メソッドのシグネチャとその内容に非常に小さな変更を加えるだけで、既存のDSEList関数をDSEListGenericにコピーしてリファクタリングするだけです。非ジェネリッククラスからジェネリッククラスに変換するヒントとして、NodeクラスとNodeGenericクラスを比較してみてください。これらは同じことを実現しますが、2番目のクラスはジェネリックをサポートしています。

## Step Three

Now that we’ve got the ability to store a list of things, we can start to build out the rest of the trading simulator. Generally you should be able to go through this list top-to-bottom in order when implementing things, however occasionally you may need to complete or at least start a function listed later in the document if it’s required by a function you’re trying to complete.

これでリストを保存できるようになったので、トレーディングシミュレータの残りの部分を構築し始めることができます。一般的には、このリストを上から下へ順番に見ていきながら実装していけばよいのですが、時には、この文書の後半に記載されている機能が、完成させようとしている機能で必要である場合、完成させたり、少なくとも開始したりする必要がある場合があります。

### ListedCompany.java

A listed company is a company that can have its shares bought and sold on a securities exchange.

* public String getName(): public getter for "name";
* public String getCode(): public getting for "code"
* public int getCurrentPrice(): public getter for "currentPrice"
* public ListedCompany(String code, String name, int currentPrice): Should store the three parameters into the instance variables
* public int processTrade(int quantity): should increase or decrease the value of the currentPrice variable depending on the quantity of stock as the parameter. The price should increase by "quantity / 100" amount, and never drop below 1 in price. For a “sell” the quantity will be negative (price goes down), and positive for a buy (price goes up).

上場企業とは、証券取引所で株式の売買ができる企業のことです。

- public String getName() ： "name "のパブリックなゲッター；

- public String getCode()："code "のパブリックゲッター。

- public int getCurrentPrice() ： "currentPrice "のpublicゲッター。

- public ListedCompany(String code, String name, int currentPrice)： 3つのパラメータをインスタンス変数に格納する必要があります。

- public int processTrade(int quantity): パラメータである在庫量に応じて、変数currentPriceの値を増減させる必要があります。価格は「数量÷100」だけ上昇し、価格が1以下になることはない。売り」の場合、数量はマイナス（価格が下がる）になり、「買い」の場合、プラス（価格が上がる）になります。

### StockBroker.java

Stock brokers take trade orders on behalf of others and process the trades on the securities exchange. The broker must track all their pending orders so they know which trade to process next. Brokers also track a watchlist of companies they advise their clients to purchase, however some dodgy brokers may encourage users to buy a certain stock, but then not process their trades on time as expected!

証券ブローカーは、他人の代わりに取引注文を取り、証券取引所で取引を処理します。ブローカーは、次にどの取引を処理すべきかを知るために、すべての保留中の注文を追跡する必要があります。ブローカーは、顧客に購入を勧める企業のウォッチリストも追跡しますが、一部の悪質なブローカーは、ユーザーに特定の銘柄の購入を勧めておきながら、期待通りの時間通りに取引を処理しないことがあります！

* private PriorityQueue<Trade> pendingTrades: Should be an instance variable using Java’s PriorityQueue class to store Trade objects.
* private DSEListGeneric<String> watchlist – Should be an instance variable of the DSEListGeneric class to store objects of type String.
* public DSEListGeneric<String> getWatchlist() – Public getter for watchlist. It should return a new list, rather than the current list. Modifying the list returned by getWatchlist (e.g. removing an item) should not affect the original version of the list held by the StockBroker.
* public Boolean addWatchlist(String companyCode) – Adds a company code to the watch list. Return false if item is already in the list or is null, true otherwise if added to the list
* public String getName() – Gets the broker’s name.
* public StockBroker(String name) – Create a broker with given name.
* public boolean placeOrder(Trade order) – Adds the Trade to the pendingTrades list if it's not null and not already in there.
* public Trade getNextTrade()
* public int getPendingTradeCount()

- private PriorityQueue<Trade> pendingTrades： Tradeオブジェクトを格納するためのJavaのPriorityQueueクラスを使用したインスタンス変数である必要があります。

- private DSEListGeneric<String> watchlist - String 型のオブジェクトを格納する DSEListGeneric クラスのインスタンス変数である必要があります。

- public DSEListGeneric<String> getWatchlist() - watchlistのパブリックゲッターです。これは、現在のリストではなく、新しいリストを返す必要があります。getWatchlist によって返されたリストを変更する (例えば、項目を削除する) ことは、StockBroker によって保持されるリストの元のバージョンに影響を与えないようにする必要があります。

- public Boolean addWatchlist(String companyCode) - ウォッチリストに会社コードを追加します。リストに追加された場合、アイテムが既にリストにあるか、またはNULLの場合はfalseを、それ以外の場合はtrueを返します。

- public String getName() - ブローカーの名前を取得します。

- public StockBroker(String name) - 指定された名前のブローカーを作成します。

- public boolean placeOrder(Trade order) - その取引がNULLでなく、まだ保留されていない場合、保留中の取引リストに追加します。

- public Trade getNextTrade()

- パブリック int getPendingTradeCount()

### Trade.java

Trade objects represent a specific number of shares to be bought in a specific company. Each trade object is also associated with the stock broker who will be processing that trade.

トレード・オブジェクトは、特定の会社の株式を購入するための特定の数を表します。また、各取引オブジェクトは、その取引を処理する証券会社と関連付けられています。

* public Trade(StockBroker broker, String listedCompanyCode, int shareQuantity): Should store the three parameters into the instance variables
* public int compareTo(Trade other): Compare this trade with another trade. Please see JavaDoc in code for more informationStep Four (Optional) – Command Line Interface

- public Trade(StockBroker broker, String listedCompanyCode, int shareQuantity)： 3つのパラメータをインスタンス変数に格納する必要があります。

- public int compareTo(Trade other)： このトレードと他のトレードを比較する。詳細については、コード内のJavaDocを参照してくださいステップ4（オプション） - コマンドラインインターフェース

### UntradedCompanyException.java

This is class should be an Exception that can be thrown when an unknown company code is used.

このクラスは、不明な会社コードが使われたときに投げられるExceptionであるべきです。

* public UntradedCompanyException(String companyCode): This should allow any exception thrown using this class to show the message “TSLA is not a listed company on this exchange”, assuming the companyCode “TSLA” was passed in as the parameter.

- public UntradedCompanyException(String companyCode)： これにより、このクラスを使用してスローされる例外は、companyCode "TSLA "がパラメータとして渡されたと仮定して、"TSLAはこの取引所の上場企業ではありません "というメッセージを表示できるはずです。

### SecuritiesExchange.java

* public SecuritiesExchange(String name)
* public boolean addCompany(ListedCompany company)
* public boolean addBroker(StockBroker broker)
* public int processTradeRound() throws UntradedCompanyException

## Optional Step Four (no marks allocated)

No marks are allocated for this final step and it is completely optional. You can still obtain a 100% for the assignment without completing this final section.

SecuritiesExchange has an additional function runCommandLineExchange(), that is sent a Scanner object. If you’re looking to push yourself a little further, this should act as a stub where you can develop your own list of commands to be processed by the exchange to add trades to brokers, process a trade round and exit. Whilst the stub for this method is given to you, as is a simple setup to automate testing of commands from a file, they will need to be extended by you. Once you can process commands from command line or a text file, who knows what’s next?!?!?! Accepting trade orders over a network?

この最終ステップには点数は配分されず、完全に任意です。この最終セクションを完了しなくても、課題に対して100%の評価を得ることは可能です。

SecuritiesExchangeには、Scannerオブジェクトを送信するrunCommandLineExchange()という関数が追加されています。自分をもう少し追い込みたいのであれば、これはスタブとして機能し、ブローカーへの取引の追加、取引ラウンドの処理、終了など、取引所が処理するコマンドの独自のリストを開発することができるはずです。この方法のスタブは、ファイルからのコマンドのテストを自動化するための簡単なセットアップとして提供されますが、これらはお客様ご自身で拡張する必要があります。コマンドラインやテキストファイルからコマンドを処理できるようになったら、次に何ができるかは誰にもわかりません！ネットワーク経由で取引注文を受け付ける？

* public int runCommandLineExchange(Scanner sc)

# Marking scheme

Ensure clean, consistent coding style. There is no official style guide for the assignment other than using the standard Java conventions. We can only award marks for code you have written, so if you don’t have any code, you don’t get any marks for style.

Your code should be commented and be easy for us to follow and understand, without too many comments. If it is not easily apparent what something is doing, ensure you comment what it’s doing and why. We can only award marks for code you have written, so if you don’t have any code, you don’t get any marks for commenting.

Marks summary is below. As part of the marking scheme, we will also be reviewing your Git history for that part. **If you haven’t committed that section/class appropriately, for example you just a large, fully functional and complete class in one commit, you may lose up to 50% of the marks for that element.** Smaller/one-line functions obviously can only be committed once complete, which is perfectly fine, but for your larger elements, we need to be able to see it’s history and how it has evolved.

クリーンで一貫性のあるコーディングスタイルを確保すること。標準的なJavaの規約を使用する以外に、課題のための公式なスタイルガイドはありません。私たちは、あなたが書いたコードに対してのみ点数を与えることができますので、コードがない場合は、スタイルに関する点数を得ることはできません。

あなたのコードはコメント付きで、私たちがフォローしやすく、理解しやすいものであるべきで、あまり多くのコメントは必要ありません。何をやっているのかがわかりにくい場合は、何をやっているのか、なぜやっているのかを必ずコメントしてください。私たちは、あなたが書いたコードに対してのみ点数を与えることができます。したがって、あなたがコードを書いていない場合、あなたはコメントのための点数を得ることはできません。

マークスサマリーは以下の通りです。採点方法の一環として、その部分の Git 履歴も確認します。そのセクションやクラスを適切にコミットしていない場合、たとえば大規模で完全な機能を持つクラスを1回のコミットで終わらせてしまった場合、その要素の点数を最大50%まで落とすことがあります。小さな、あるいは一行で完結する機能は、明らかに一度しかコミットできないので問題ありませんが、大きな要素については、その履歴と進化を確認する必要があります。

|  |  |
| --- | --- |
| **Code passes required tests**  Your code passes the tests we run on it. Whilst you are given some tests for the classes, we may use additional tests for the final marking. The proportion of marks per class will be similar to the assignment marker you are given. This means the majority of the marks will still be for the List and ListGeneric classes. Please see the marking program for further breakdown of marks.  あなたのコードは、私たちが実行したテストに合格しています。授業ではいくつかのテストが与えられますが、最終的な採点では追加のテストを使用することがあります。クラスごとの採点比率は、与えられた課題マーカーと同様です。つまり、ListクラスとListGenericクラスの採点が大半を占めます。詳しい採点の内訳は、採点プログラムをご覧ください。 | 80% |
| **Code style**  Clean, consistent coding style. There is no official style guide for the assignment other than using the standard Java conventions. We can only award marks for code you have written, so if you don’t have any code, you don’t get any marks for style.  クリーンで一貫性のあるコーディングスタイル。この課題には、標準的なJavaの規約を使用する以外に、公式なスタイルガイドはありません。私たちは、あなたが書いたコードに対してのみ点数を与えることができますので、コードがない場合は、スタイルに関する点数を得ることはできません。 | 10% |
| **Code commenting**  Your code should be commented and be easy for us to follow and understand, without too many comments. If it is not easily apparent what something is doing, ensure you comment what it’s doing and why. We can only award marks for code you have written, so if you don’t have any code, you don’t get any marks for commenting.  あなたのコードはコメント付きで、私たちがフォローし理解しやすいものであるべきです、あまり多くのコメントはありません。何をやっているのかがわかりにくい場合は、何をやっているのか、なぜやっているのかを必ずコメントしてください。私たちは、あなたが書いたコードに対してのみ点数を与えることができます。したがって、あなたがコードを書いていない場合、あなたはコメントするための点数を得ることはできません。 | 10% |
| **Total** | 100% |

# Submitting

Please ZIP your entire root folder of the project and upload to LearnOnline. This folder should include your src directory.

**We must receive your .java source files from the “src” folder in order to mark any code.**

## Due date

Please see the course website for the due date. Unless you have an extension, there is **NO LATE SUBMISSION**. Late submission without an extension will result in a mark of 0.

Please regularly upload a snapshot of your code to the site so that we have a copy. This means if something does happen and you can’t submit your latest version, we still have an earlier version to mark.

## Extensions

If you need an extension for an on-going issue, e.g. medical, access plan, etc., please submit a request via LearnOnline to be reviewed. If you have any issues please email me directly via email. **Having other assignments or work commitments is not a valid reason for extension.**

1. Image courtesy of <https://wave.fr/reddit-gamestope-bourse-300900> [↑](#footnote-ref-1)