Problem Solving Session

- The remainder of today's class will comprise the ***problem solving session*** (***PSS***).
- Your instructor will divide you into ***teams of 3 or 4 students***.
- Each team will ***work together*** to solve the following problems over the course of ***20-30 minutes***.
  - You may work on paper, a white board, or digitally as determined by your instructor.
  - You will submit your solution by pushing it to GitHub before the end of class.
- Your instructor will go over the solution before the end of class.
- If there is any time remaining, you will begin work on your homework assignment.

Class participation is a significant part of your grade (20%). This includes in class activities and the problem solving session.

Your Course Assistants will grade your participation by verifying that you pushed your solutions before the end of the class period each day.

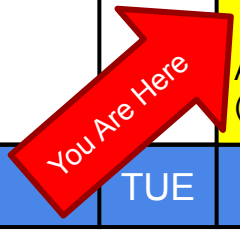
Problem Solving Team Members

Record the name of each of your problem solving team members here.

Do not forget to ***add every team member's name***! Your instructor (or course assistant) may or may not use this to determine whether or not you participated in the problem solving session.

Coming Up

SUN	MON (4/15)	TUE	WED (4/17)	THU	FRI (4/19)	SAT
	Unit 11: Thread Cooperation		Project Time		Unit 12: Networking	
	Club Chèvre Assignment 11.1 Due (start of class)		Project Part 3 Team Problem-Solving Assignment 11.2 Due (start of class)		Unit 11 Mini-Practicum	



SUN	MON (4/22)	TUE	WED (4/24)	THU	FRI (4/26)	SAT
	Unit 12: Networking				Unit 13: Semester Review	
	Project Part 3 Due (start of class)		Assignment 12.1 Due (start of class)		Unit 12 Mini-Practicum Assignment 12.2 Due (start of class)	

Pete's Pike

Pete's Pike is a logic game where the goal is to get Pete the mountain climber safely to the top of the mountain. Pete is accompanied by his trusty goats who will assist him in his journey.

Pete and his goats, collectively known as pieces, can move either horizontally or vertically. However, if there is no other piece in the direction they are moving, they will slide off the mountain to certain doom. If there is a piece in the direction of the move, the moving piece will stop at the space directly before the stationary piece.

The game ends when either Pete is on the space designated as the mountaintop or there are no valid moves remaining.

Pete's Pike board game was released by Thinkfun in 2007.

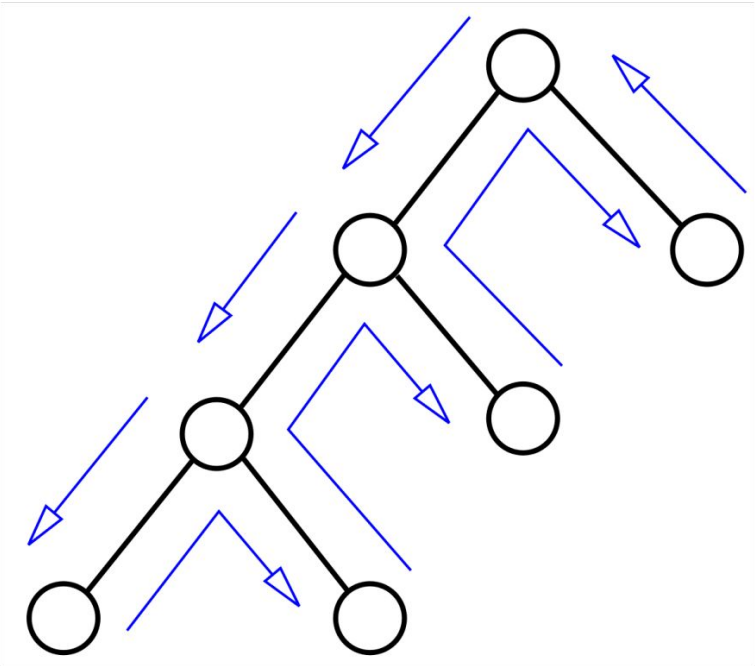
- ***Pair programming*** is a technique during which two developers collaborate to solve a software problem by writing code together.
- One developer takes on the role of ***the driver***.
  - Shares their screen.
  - Is actively writing code.
- The other developer(s) takes on the role of ***the navigator***.
  - Watches while the driver codes.
  - Takes notes.
  - Asks questions.
  - Points out potential errors.
  - Makes suggestions for improvements.
- The driver and navigator regularly ***switch roles***, e.g. every ***10-20 minutes***.
  - Set a timer!
  - ***Push your code!***
- For the rest of today's problem solving session, you and your team will practice pair programming with ***one*** team member acting as the driver and the ***remaining*** team members acting as the navigators.

Pair Programming

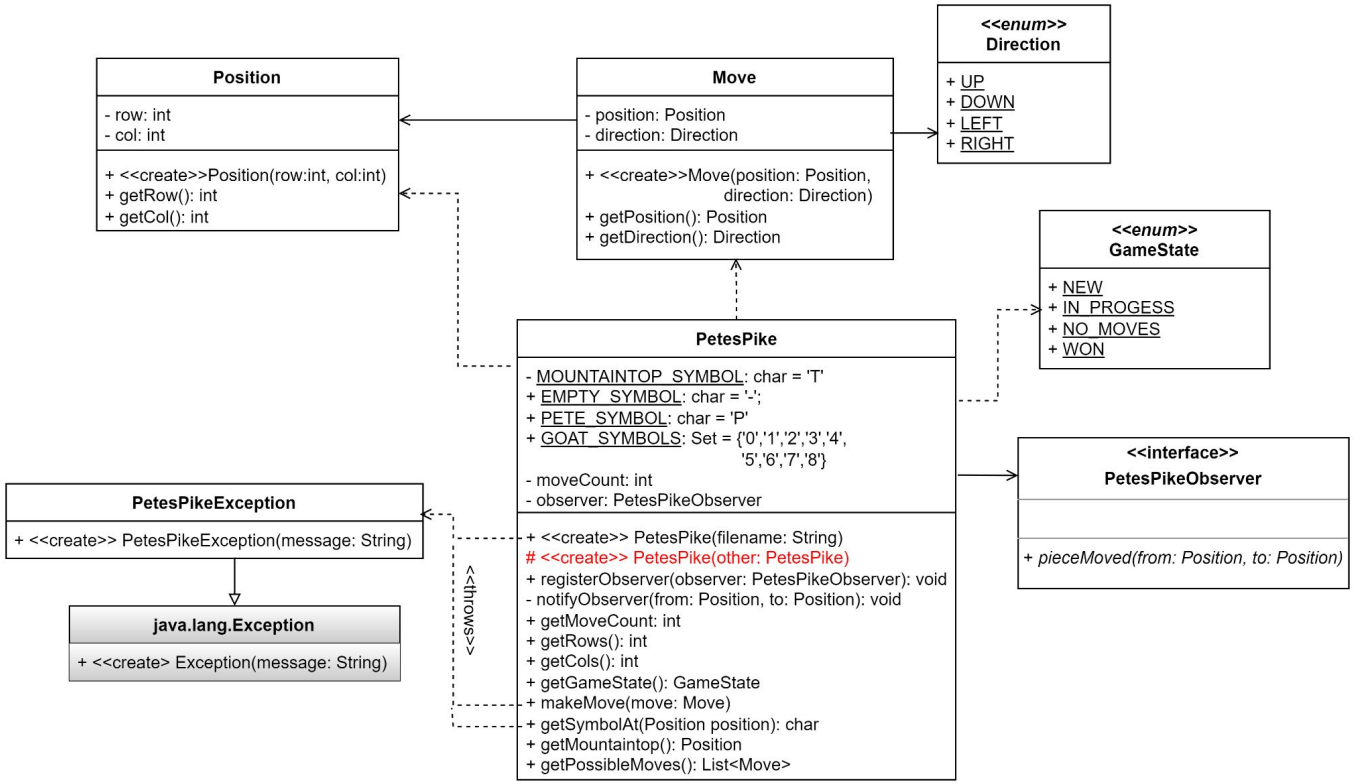
When you are the driver, you should be using Zoom or Discord to ***share your screen***. Be sure to ***push your code*** and ***switch roles*** every 10-20 minutes!

Pete's Pike Part 3

- This is the ***third*** part of a ***three*** part project.
  - This part is due on ***Friday (12/6)*** at your ***class time***.
- In this part of the project, you will primarily be focused on creating a ***backtracking configuration*** that will attempt to find a series of piece moves that will win a Pete's Pike game.
  - ***Zero or more*** moves may have already been made.
  - You will need to provide the ***list of winning moves***.
- Your configuration will be used to implement a solve feature for both your command line interface and graphical user interface.
- While you might be able to solve a game by using `getPossibleMoves()`, your project must implement a backtracking configuration.

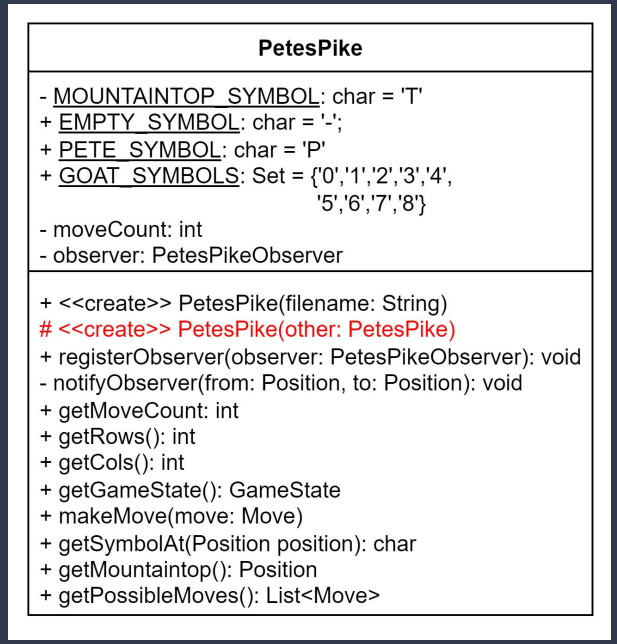


A Partial Design



Refer to this full UML class diagram as needed while you and your team work through the rest of this assignment.

Copy Constructor



A ***copy constructor*** creates a new object by making a ***deep copy*** of an object of the same class.

- Your ***backtracking configuration*** will almost certainly need to create a ***deep copies*** of your `PetesPike` implementation as it creates successors.
- To support the ability of a `PetesPike` object to make a copy of itself, we will need to add another constructor to the class.
- Next, you and your team should begin working on implementing the ***copy constructor*** method in your `PetesPike` implementation.
    - When called to create a new instance of the class, it should make a ***deep copy*** the `PetesPike` object that was passed in.
    - Don't forget that you will need to make deep copies of any ***mutable types*** or ***data structures*** like lists and arrays!
    - Do not make a copy of the `observer`, but rather set it to `null`.
        - The `observer` was registered to be notified of the changes to ***original*** object.
        - If the `observer` is replicated in the ***copy constructor***, your GUI will be notified for every successor!

What state will your ***configuration*** need to keep track of as it attempts to find a solution?

The board, the pieces, list of Valid moves

How will you make ***successor*** configurations?

These will be made by using one of the possible and valid moves which would represent one of the choices made.

How will you determine if a configuration is ***invalid***?

If the piece attempts to go out of bounds

How will you determine if the configuration is ***the goal***?

The configuration is a goal if Pete makes it the top of the mountain.

Do you need to be concerned with ***cycles***? If so, how will you avoid them?

Cycles can be avoided by using a setlist of the configurations that were previously used.

Backtracking

In this part of the project you will be creating a ***backtracking configuration*** that will attempt to solve a Pete's Pike Game. You will also need to provide the list of selections needed to win.

Examine your code and think about how you will implement your configuration and answer the questions to the left in a ***backtracking.txt*** file. Be as detailed as possible. Push your file to the ***data directory*** of your repository when complete.

Remember:

- A ***configuration*** is at least a partial attempt at a solution.
- A ***successor*** is a new configuration that includes one additional choice.
- A configuration is ***invalid*** if it is impossible to find a solution from this point.
- A configuration is the ***goal*** if it is a valid solution to the problem.

A Configuration

You may want to consider adding a ***main*** method to your configuration to manually test it. Use the following example to guide you.

```
1  MySolver initial = new MySolver(game);
2  Backtracker<MySolver> backtracker
                          = new Backtracker<>(true);
3  MySolver solution = backtracker.solve(initial);
4  if (solution == null) {
5     System.out.println("No solution.");
6  } else {
7     System.out.println(solution);
8  }
```

You will of course need to create an instance of your PetesPike Game implementation class to use as well.

- The `Backtracker` and `Configuration` classes have been provided in your repository.
  - You have also been provided with output from both the ***command-line interface*** and ***graphical user interface*** at the end of the Part 3 Assignment. You can use these to guide the implementations of the new feature.
- Using the information that your team brainstormed in the previous activity, begin implementing a ***backtracking configuration*** to solve a `PetesPike` game.
  - ***Do not*** change the provided classes.
  - **It is recommended that your `Configuration` implementation be a separate class from `PetesPike`.**
  - Other than the copy constructor, do you need to refactor your `PetesPike` class at all?
  - Consider overriding the `toString()` method in your configuration for use when debug mode is enabled in the backtracker.

Date/Time	Location	Area of Focus
4/20 @ 1:00PM	Discord	Successors

Meeting Times

This part of the project is due ***Wednesday April 22nd, 2023*** at ***class time***.

You should plan to ***work together*** with your team as much as you can, even if that means setting up a remote meeting using Zoom or Discord.

Plan ***at least 2 meetings*** over the course of this part of the project. Each meeting should be at least ***1 hour***. Following the example on the left, put the information for all scheduled meetings in a *meetings.txt* file and *push to your repository.*

If You Made it This Far...

Your team is off to a good start, but you are ***not quite*** finished with ***Part 3*** of the Project yet. Remember that this part of the project is due on ***Monday April 22nd, 2023*** at ***class time***.

You still need to implement the automatic solve feature in your ***command-line*** and ***graphical user interfaces***.

If you have time remaining in class, you should begin reading the full project description, which you will find on MyCourses.