CS 284: Homework Assignment 3
Due: October 17, 11:55pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

2 Assignment

This assignment consists in implementing a circular doubly linked list to create a `MultiplayerGame`. A `MultiplayerGame` consists of a circular list of entities. Each entity can either be a `GamePlayer` or a `GamePiece`. Each turn of the game consists of iterating over each player and each of their game pieces and processing their moves.

The implementation of `MultiplayerGame` is based on a doubly-linked list. Each node in the list is called a `GameEntity`. Each `GameEntity` will have a reference to the next entity and to the previous one. A `GameEntity` is either a `GamePlayer` or `GamePiece`. The application is organized into four Java classes. UML diagrams for each class are given at the end of the assignment.

- `MultiplayerGame`: This is the main class. Your will be implementing this class.
- `GameEntity`: Implemented for you. Abstract class.
- `GamePlayer`: Implemented for you. Subclass of `GameEntity`.
- `GamePiece`: Implemented for you. Subclass of `GameEntity`.

The `GamePlayer` and `GamePiece` classes are implemented for you and code is available on Canvas. A `GamePlayer` has an `int id`. A `GamePiece` has a `String name` and a `int strength`. The abstract class `GameEntity` that both `GamePlayer` and `GamePiece` extend is also available on Canvas.

3 Implementation

3.1 Part 1: Basic Operations

You are asked to implement the operations for `MultiplayerGame` given below. Please take a look at the detailed UML diagram of the class at the end of the assignment before starting to implement these.

The `MultiplayerGame` class has two private fields, a `GameEntity turnTracker` and `GameEntity[] index`. The `turnTracker` is described in detail in the next section, but is basically a marker used to point at the `GameEntity` who is currently taking their turn. The `index` is an array where each entry in the array refers to a `GamePlayer`. There should be n entries in index, where n is the number of players in the game (n is given in the constructor). For example, `index[0]` should be a reference to the `GamePlayer` with `playerId` 0. Using `index`, you will be able to access each `GamePlayer` directly without walking through the entire linked list of `GameEntities`.

A turn is processed as follows. In each player's turn, each `GamePiece` of the player makes their move. It does not matter what order those `GamePieces` make their moves in. The `GamePlayer` with `playerId` 0 starts. Iterating through the `MutliplayerGame` using the next fields of the entities should produce this order. It should also be the order given by `toString`. The following operations should be implemented:

- `MultiplayerGame(int n)`, creates a new `MultiplayerGame` with n players. It should initialize index to an array of size n and create a new `GamePlayer` with the appropriate `playerId` to store at each index. Note that the next field for the n'th player should reference the first `GamePlayer`.

- `public int size()` returns the size of the `MultiplayerGame`. The size is the number of `GamePieces` in play. `GamePlayers` do not add to size.

- `public void addGamePiece(int playerId, String name, int strength)` adds a `GamePiece` owned by specified player of the given strength to the game. If the player already owns a `GamePiece` of that name, then an `IllegalArgumentException` with the message "addGamePiece: duplicate entity" should be thrown. If the player does not exist, then an `IllegalArgumentException` with the message "addGamePiece: no such player" should be thrown.

- `public void removeGamePiece(int playerId, String name)` removes the `GamePiece` owned by specified player of the given name. If no such `GamePiece` exists, then an `IllegalArgumentException` with the message "removeGamePiece: entity does not exist" should be thrown. If the player does not exist, then an `IllegalArgumentException` with the message "removeGamePiece: no such player" should be thrown.

- `public boolean hasGamePiece(String name)` checks if any player has a `GamePiece` of the given name.

- `public void removeAllGamePieces(int playerId)` removes all the `GamePieces` owned by specified player. If the player does not exist, then an `IllegalArgumentException` with the message "removeAllGamePieces: no such player" should be thrown.

- `public void increaseStrength(int playerId, int n)` increases the strength of all `GamePieces` owned by the specified player by n. n can be negative. If the player does not exist, then an `IllegalArgumentException` with the message "increaseStrength: no such player" should be thrown.
- `public String toString()` produces a String representation of the `MultiplayerGame`. Should output the String representation of the first player, then each of their pieces. Then likewise for the second player and so on.

Important: For all methods above that have a `playerId` parameter, you must only search through the `GamePieces` of that player. In particular, you cannot exhaustively search through the entire `MultiplayerGame`.

3.2 Part 2: Processing a Turn

To process a turn, the following operations should be implemented.

- `public void initializeTurnTracker()` sets the `turnTracker` to the first `GamePlayer`.
- `public void nextPlayer()` moves the `turnTracker` to the next `GamePlayer`.
- `public void nextEntity()` moves the `turnTracker` to the next `GameEntity`, which could be either a `GamePlayer` or a `GamePiece`
- `public void prevPlayer()` backtracks the `turnTracker` to the previous `GamePlayer`.
- `public String currentEntityToString();` returns the string representation of the current entity pointed to by the `turnTracker`.

4 Submission instructions

Submit a single file named `MultiplayerGame.zip` through Canvas that includes all files in stub (which must have been completed according the to instructions above) and a file `MultiplayerGameTest.java` with your test cases. No report is required. Your grade will be determined as follows:

- You will get 0 if your code does not compile.
- The code must implement the following UML diagram precisely.
- We will try to feed erroneous and inconsistent inputs to all methods. All arguments should be checked.
- Points will be given for comments, style and readability. JavaDoc comments are required for each function you implement.
- Points will be given for JUnit tests.

GameEntity
protected GameEntity prev; protected GameEntity next;

public GameEntity();
public abstract boolean isGamePlayer();
public abstract int size();
public abstract String getName();

GamePiece extends GameEntity
private String name; private int strength;

public GamePiece(Entry prev,
Entry next, String name, int strength);
public boolean isGamePlayer();
public int size();
public int getStrength();
public void updateStrength(int n);
public String getName();
public String toString();

GamePlayer extends GameEntity
private int playerId;

public GamePlayer(GameEntity prev, GameEntity next,
int playerId);
public int getPlayerId();
public boolean isGamePlayer();
public int size();
public String getName();
public String toString();

The class `MultiplayerGame` should include the following operations:

MultiplayerGame
private GameEntity turnTracker; private GameEntity[] index;

public MultiplayerGame(int n);
public int size();
public void addGamePiece(int playerId, String name, int strength);
public void removeGamePiece(int playerId, String name);
public boolean hasGamePiece(String name);
public void removeAllGamePieces(int playerId);
public void increaseStrength(int playerId, int n);
public String toString();
 	
public void initializeTurnTracker();
public void nextPlayer();
public void nextEntity();
public void prevPlayer();
public String currentEntryToString();