

California State University, Sacramento
College of Engineering and Computer Science

Computer Science 130: Data Structures and Algorithm Analysis

Fall 2026 – Project #3 – The Arcade Abstract Data Type

Overview



Now that you have a proper O(1) linked list, it is time to take advantage of it to create a couple more classes.

In this project, you are going to use the **`GameList`** Class to create a Stack and Queue. Because the linked list does all the work, these two classes will be incredibly easy.

Then, you are going to use them (the queue in particular) to create a nice Arcade Class.

Part 1: Double-Checking your Work

For this first assignment, you are going to use the code you developed in Project #2. Before you continue, it is a great idea to double-check your code.

Static

It is important to reiterate that you need to check your code for **`static`**. A static function is associated with the class, but not an instance. Normally, your program has a static function called **`main()`**. This should be the **only** static definition in your program. **So, there should be only one static.**

Linked List Logic

Also, double check if you have the complete logic for **`removeHead()`** in your GameList class. If you didn't program all three cases, then your classes may (and probably will) crash.

Part 2: Your Stack & Queue

The Stack Class

The following is the interface (public functions) for the Stack Class. This will be fairly easy to write – since it wraps around your GameList class. In other words, these functions merely call the appropriate function on the instance of the GameList. **Don't inherit**.

GameStack Class Constructors	
GameStack()	Internal GameList is set to a new instance.
GameStack(GameList)	Creates a stack based on an existing game list. You can simply assign the internal list to this parameter.

The methods of your GameStack Class will simply call the appropriate methods of the internal instance of the GameList Class. As a result, you will write very little code for this class.

GameStack Class Methods		
void	**push(Game)**	Pushes a Game onto the stack.
Game	**pop()**	Pops (removes) a Game from the top of the stack.
boolean	**isEmpty()**	Returns true if the queue is empty.
String	**toString()**	Return the string from the internal linked-list class.



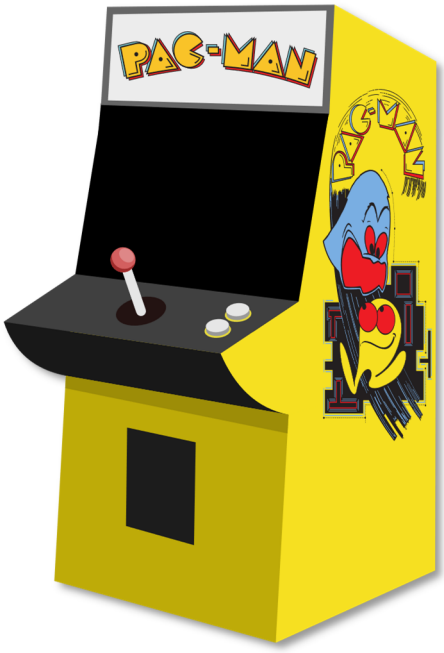
The Queue Class

Like before, these functions should call the appropriate function on the private GameList instance. **Don't inherit**.

GameQueue Class Constructors	
GameQueue()	Internal GameList is set to a new instance.
GameQueue(GameList)	Creates a stack based on an existing game list. You can simply assign the internal list to this parameter.

GameQueue Class Methods		
void	**enqueue(Game)**	Enqueues a Game onto the queue.
Game	**dequeue()**	Dequeues (removes) a Game from the front of the queue.
boolean	**isEmpty()**	Returns true if the queue is empty.
String	**toString()**	Return the string from the internal linked-list class.

Part 3: Arcade Class



It is now time to create a very useful **Arcade** class. Like well-designed abstract data types, the data structure will be hidden from the user. This will allow us to change it later in the semester (hint, hint).

Internally, we are going to store a copy of the **GameList** and the **GameQueue** class. Why two collections? Well, one will contain all the games in our arcade.

The queue will used to support a video game tournaments. When a video game tournament is started, the queue will be filled with the games (by either Category or Platform). Games will be then dequeued and played by our players.

Why not use the main list? Well, that would make us lose our data! By duplicating the main list, we can have a working queue **and** preserve our original data.

Arcade Class PRIVATE Members		
GameList	**games**	This linked list is used to maintain the main list of games.
GameQueue	**playQueue**	This will be created using a sublist of the "games" list.

Constructors

The Arcade Class will have only one constructor. You should set **games** to a new instance of the Linked List class.

Arcade Class Constructors
Arcade()

Methods

The Linked List data structure will be doing most of the work in our class. As a result, most of the methods below will contain very little code.

Arcade Class Methods		
void	**add(Game)**	Adds an Game instance to main list. This is called "**games**" in the field list above.
string	**toString()**	Returns the contents of the main list. Simply return the result of the linked list's **toString()** method.
void	**start(Platform)**	Duplicates the main linked list using the **sublist(Platform)** method. The result is stored into the **playQueue** field of this class.
void	**start(Category)**	Duplicates the main linked list using the **sublist(Category)** method. The result is stored into the **playQueue** field of this class.
Game	**playNext()**	Removes the head from the **playQueue** and returns it to the caller.
boolean	**donePlaying()**	Returns true of either the **playQueue** variable is null or (if it is not null), the result of **isEmpty()**.

Part 4: Testing

You need to test the logic of your Arcade Class. In the **`main()`** function of your Tester Class (or whatever you call it – it doesn't matter to me), create at least fifteen **(15)** instances of the Game class. Add them **all** to an instance of the Arcade Class.

You can try using **`toString()`** method. Also, create a loop and feed the games (outputting the nick name each time). Try it using both **`start()`** methods. Do **not** read values from a file or the keyboard.

To test your Arcade Class: 1. Start a tournament, 2. Create a loop that runs until **`donePlaying()`** is true.

```
Print what type of tournament you are starting.
call start() on your arcade

while not arcade.doneplaying()
    game = arcade.playnext()
    print game
end while
```

Allowed Programming Languages

You may use any of the following programming languages.

- C#
- Java

The following **cannot** be used:

- C++
- Groovy
- JavaScript
- Kotlin
- Lua
- Nim
- Python
- Ruby
- Scala
- Swift
- TypeScript
- Visual Basic .NET

Requirements

The following are the requirements for this assignment:



- The classes and methods defined above.
- You **must** use your code from Project #2
- You may use any of the programming languages listed above.
- Create some excellent testing for your class.
- Proper style (see below).
- Do **not** put your `main()` method inside the Linked List class. This will cause major issues going forward.
- This **must** be completely all your code. If you share your solution with another student or re-use code from another class, you will receive a zero.
- Do **not** use any built-in collection classes such as ArrayList. Many programming languages have nice collection classes available. Do not use any of them. **If you use any of these, you will receive a zero. No exceptions. No resubmissions.**

Grading

1	Arcade Class	40%
2	Stack and Queue classes	20%
3	Proper interfaces for the new classes	20%
4	Proper Style	5%
5	Proper testing (see above).	10%
6	**`main()`** is in a testing class (not in Game, Node or GameList)	5%

Proper Style

Points will be deducted if your program doesn't adhere to basic programming style guidelines.

1. If programming C++, Java, or C#, I don't care where you put the starting curly bracket. Just be consistent.
2. Indentation must be used.
3. Indentation must be consistent. Three or four spaces works. **Beware of the tab character!** It might not appear correctly on my computer (tabs are inconsistent in size).
4. Proper commenting. Not every line needs a comment, but sections that contain logic often do. Add a comment before every section of code – such as a loop or If Statement. Any complex idea, such as setting a link, must have a comment.



Due Date & Submission

Due **October 23, 2026** by **11:59 pm.**

E-Mail your source code files to **dcook@csus.edu.** Please zip your source code files into a single .zip or .tar file. Please take note of the following. If you send your files incorrectly, I either won't receive them or won't be able to grade them.



1. Send **ONLY** **`.java`**, or **`.cs`** files. The e-mail server will delete all attachments that have file extensions it deems dangerous. This includes **`.jar`**, **`.exe`**, and **`.class`**.
2. Do **NOT** simply zip an entire folder. The server will delete it.
3. Do **NOT** send a cloud link. I cannot open or grade these. Send an attachment.
4. Do **NOT** send it to canvas. I will not read nor grade Canvas e-mails.