California State University, Sacramento
College of Engineering and Computer Science

Computer Science 130: Data Structures and Algorithm Analysis

Fall 2026 – Project #1 – Video Game Class

Overview

For this semester, we are going to create a basic program that helps organize an arcade.

For this first assignment, you are going to create the main class that we are going to maintain this semester. Given you already have developed excellent programming skills in CSc 20, this shouldn't be a difficult assignment.

Part 1: Enumerations

Category Enumeration

The arcade needs to keep track of the category of the game. Obviously, there are many different ways of classifying games. But, this is *our* arcade. Many classifications also contain the group "strategy", but, practically all games have strategy. So, that category doesn't make much sense. Our categories are as follows.

Note: In Java, the proper naming convention, for enumeration constants, is UPPERCASE. The enum itself, like classes, is in PascalCase.

- `ACTION`
- `ADVENTURE`
- `FIGHTING`
- `PLATFORM`
- `RPG`
- `PUZZLE`
- `SPORTS`
- `OTHER`

Platform Enumeration

Our arcade will also keep track of the game's platform. This is not an extensive list, but it covers the major ones.

- **`ARCADE`**
- **`ATARI_2600`**
- **`DOS`**
- **`COMMODORE_64`**
- **`APPLE_2`**
- **`NES`**
- **`SUPER_NINTENDO`**
- **`NINTENDO_64`**
- **`PLAYSTATION`**
- **`PLAYSTATION_2`**
- **`PLAYSTATION_3`**
- **`PLAYSTATION_4`**
- **`PLAYSTATION_5`**
- **`XBOX`**
- **`XBOX_360`**
- **`XBOX_ONE`**
- **`XBOX_X`** *(Seriously? Microsoft, what the heck?)*
- **`OTHER`**

Part 2: Game Class

Attributes

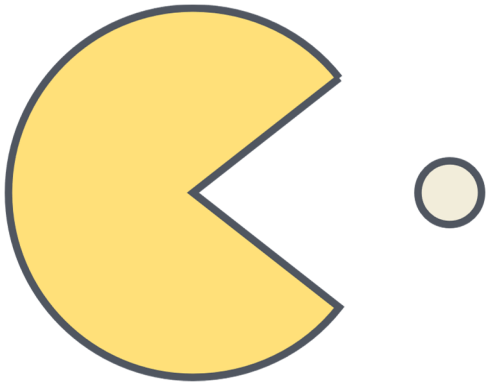
The Game Class will keep track of six different attributes. We will use these in different ways as the semester progresses. We might add some more later.

Game Class Attributes		
int	**ID**	Each game is assigned a unique number.
string	**name**	The name of the game. e.g. Pac-Man.
Platform	**platform**	The platform (type of computer) the game runs on. You can have multiple games, with the same name, but different platforms. (i.e. “ports”)
Category	**category**	The category does the game belong to.
int	**rating**	The game’s rating from 1 (poor) to 4 (excellent).
int	**cost**	How much does the game cost (when it was new).

Constructors

We are just beginning to create our class, so we won't have that many methods (at least for now). Our basic constructor will contain the six fields above. Naturally, you can change the name of the parameters.

Game Class Constructors
Game(int id, string name, Category category, Platform platform, **int rating, int cost)**



Private methods

You will need to create the following private methods. Since they are private, you can change the name of them, if you wish.

Required Private Methods		
string	**toName(Category)**	Converts a category into a string. Using a Switch Statement is recommended.
string	**toName(Platform)**	Converts a platform into a string. Using a Switch Statement is recommended.

Interface (public methods)

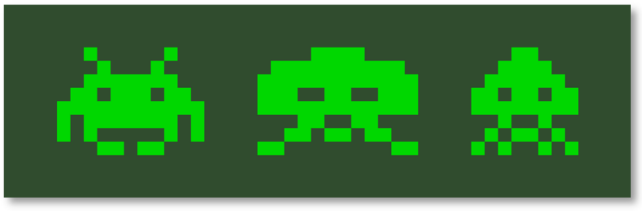
We won't have that many methods (at least for now). Besides the methods displayed below, create accessors (getters) for each of the six properties.

Game Class Interface		
String	**nickname()**	Returns a string with the game's name + "(" + it's platform + ")" For example, if the game's name is "Zak McKracken" on the Commodore 64, the function will return "Zak McKracken (Commodore 64)".
String	**description()**	Returns a string describing the game. All the attributes will be used to construct the string. Please see below.
String	**toString()**	If you are using Java, override the "toString()" method to return **nickname()**. This is just one line of code.

description() function

This function will construct a string using the attributes listed above. I've made the text a different color for each attribute (just to make it easier to read).

Zak McKracken, on the Commodore 64, is a 4 star adventure game that costs $20.



Part 4: Testing

You need to test the logic of your Game Class. In the `main()` function of your Tester Class (or whatever you call it – it doesn't matter to me), create at least **ten (10)** instances of the Game Class. Print the description and nicknames for testing. Do **not** read values from a file or the keyboard.

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:

- 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.
- 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 Game Class. This will cause major issues going forward.

Due Date & Submission

Due **September 25, 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.

Grading

1	Correct Interface	20%
2	Private methods	20%
3	Proper naming conventions	10%
4	Proper Style	10%
5	Proper testing (see above).	20%
6	`main()` is in a testing class (not in the Game class)	20%

Beware of Static

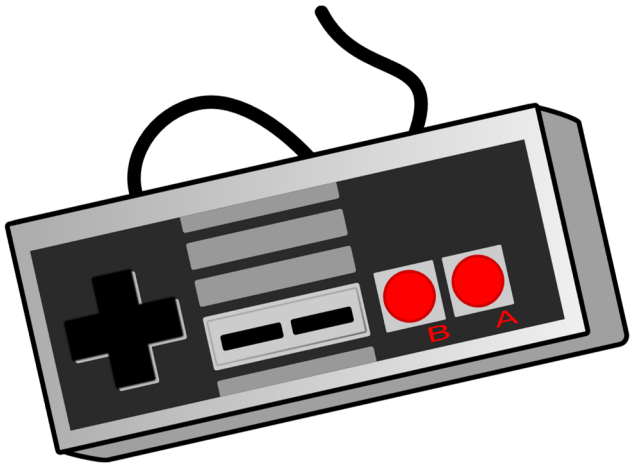
The `static` keyword in Java indicates something that will have just 1 copy and it is not associated with an instance.

- If you define a static class, then you cannot create instances of it. Java will also force you to make every function static as well.
- 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.
- If you declare fields in a class as static, there is only one copy that is shared by all instance. This can be disastrous in your code. For example, if you declare `head` and `tail` as static, you probably won't get weird side effects if you create **only one** instance of your class. However, if you create more than one instance, **they will fail** (they are using each other's data).

Proper Style

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

1. 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.



Well-formatted code

The following code is well formatted and commented.

```
void foo()
{
    int x;

    //Print off the list
    x = 0;
    while (x < this.count)
    {
       items.Add(this[x]);  //Add the item to the temporary list
       x++;
    }
}
```

Poorly-formatted code

The following code is poorly formatted and documented.

```
void foo()
{
    int x;

x = 0;
while (x < this.count)   {
     items.Add(this[x]);  //Call add on items.
       x++;
}
}
```