

Draw It or Lose it

# **CS 230 Project Software Design Informative**

Version 2.0

Johnathan D. Phillips

Southern New Hampshire University

CS-230-H4756 Operating Platforms 21EW4

Dr. Vivian Lyon, DIT, MSIT, MSc.IT, MBA

March 20, 2021

## Table of Contents

Contents

[**CS 230 Project Software Design Informative** 1](#_Toc67743940)

[*Table of Contents 2*](#_Toc67743941)

[Document Revision History 2](#_Toc67743942)

[Executive Summary 3](#_Toc67743943)

[Design Constraints 3](#_Toc67743944)

[System Architecture View 4](#_Toc67743945)

[Domain Model 4](#_Toc67743946)

[Evaluation 7](#_Toc67743947)

[Recommendations 12](#_Toc67743948)

[**Definitions 14**](#_Toc67743949)

[*Citations 16*](#_Toc67743950)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/25/2021 | Johnathan Phillips | Initial proof of concept for the structuring of entities within the game. |
| 2.0 | 03/25/2021 | Johnathan Phillips | * Added general recommendations and plans on evaluation section specific to client-side and server-side development and development tools. * Updated design constraints section. The information has been added in order to be more in depth/specific to the needs of the “Draw It or Lose It” application. * Revised grammar and formatting. * Provided security suggestions. * Added in-text UML class diagram and large print UML class diagram. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has tasked Creative Technology Solutions (CTS) with turning their gaming app, “Draw It or Lose It”, into a web-based version of itself. The idea is that by making the web-based version, it can be run on multiple operating platforms. Software requirements state that unique instances of games can be created with uniquely named teams and uniquely named players. Only one instance of a game can be in running in the system’s memory at any given time. Team and player names should also be unique.

Hardware requirements will be further evaluated once the software is complete. Since the game is web-based and intended for use on multiple platforms, both mobile and desktop versions of the site should be created. If the budget should allow, more niche versions of the site should accommodate for various gaming console browsers, and high-end simple cell phones, and low-end smart devices (tablets, smartphones, etc.). “Draw It or Lose it” should present pleasant functionality on systems with limited resources such as ram, processing power, and connected networks that cannot handle large transmit/receive loads. Again, should the budget allow, and the platform support it, the operating platform be recognized, and features be optimized for a “lag-free” experience. The user should also be able to toggle features on or off at their discretion if the capacity for the operating platform is evaluated falsely. None of this may be necessary though, as the most demanding part of the game would be slowly rending the drawing players use to solve puzzles – not very demanding for most modern platforms.

Using a **singleton** called *Game Service*, *Game***objects** are initialized and stored in **lists**. Understand that a “singleton” is a type of programming pattern used to ensure only one instance of an object is made and is self-initializing (initialized with the start of the program) and cannot be initialized elsewhere. These games can create *Team*objects which are stored in lists which create *Player* objects stored in lists. Each list is respective to the object that creates it. Games, teams, and players are created with unique names, or else a game/team with a name matching the user’s input will be selected for use. This process works by using the “iterator” pattern, which, in this case, iterates through the lists where either Game objects, Team objects, or Player objects are stored looking to see if the chosen object has already been initiated with the chosen name. The *Singleton Tester* is for diagnostic purposes only to ensure the proper functioning of the *Game Service.*

## [Design Constraints](#_2et92p0)

“Draw It or Lose It” presents a few standard design constraints because it is web-based to support multiplatform functionality. To cater to most devices, the web application cannot be very demanding, support basic mouse and keyboard integration, as well as a mobile version to cater to those platforms most common today: smart devices. Understandably, there is already an app version, but one streamlined mobile version would reduce the need to update software that has to function on multiple devices.

The simplest approach would be to create a site that supports standard desktops – a platform many non-desktop devices, including Smart TVs and consoles, are also equipped to navigate – as well as a mobile version. The mobile site version may require the least work as its overall construction would resemble the already present mobile application and could even forward the user to the mobile app if it is installed on their device.

Application users must be created uniquely, as well as game instances and teams, as to prevent application errors and illogical functionality. Games will be timed and therefore require access to a system clock on the backend in order to synchronize the timers displayed on client-side platforms. Users will be provided interfaces to login or set up user info and credentials for future logins. It may be decided that users not willing to commit to an account or simply trying out the game be generated a temporary guest account with a unique guest ID. This information can be used to generate a temporary username such as “guestID1234”.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_8h2ehzxfam4o)

The *ProgramDriver* is where the software begins its functions as it contains the ***main* method**, a **function** standard in all Java software. Within the *main* method, the *GameService* is accessed to create the required program objects. The *GameService* is what is known as a singleton class, it follows the singleton programming pattern. To be less redundant, this class is to be running only one instance at a time and that no others are to be created nor this one destroyed. The *GameService* is a class that is self-initializing with static data members that remain and are updated and referenced throughout the program’s execution. These data members include the list where games are stored, the nextGameId, the nextPlayerId, and the nextTeamId.

*Entity* is a superclass of which *Game*, *Team*, and *Player* **inherit** from. “0…\*” means from zero to any number. So, *Game* can exist in any number of instances and a line is drawn containing the “0…\*” to GameService because games are presently created from the game service – *Game* and *GameService* are connected. Any number of Teams can exist and are instantiated from *Game*. And, as the line stemming from Player to Team says, a team can have any number of players. At present, teams are not exclusive to existing games and players are not exclusive to existing teams and games an existing instantiation of the game service. This is due to the nature of the respective classes. “Draw It or Lose It” may be designed to function where players cannot exist without a *Team* object, teams cannot exist without a *Game* object, etc.

Data and methods can be exclusive to their respective subclasses and or parent classes. Members presented with “+” are public, “-” private, and “#” are protected. See **encapsulation** in the **Definitions** section. The Entity method could have very well been made **abstract** (also included in **Definitions**).

****

## [Evaluation](#_2o15spng8stw)

*Below will consists of initial development recommendations.*

**Server-Side**

Backend development requires a physical server platform capable of interfacing with any number of client platforms. Mobile devices have the widest appeal and should be considered during the development of server-side infrastructure. Linux-based servers can be built using free open-source software and can communicate with any number of devices with a **RESTful API** (see details under development tools) or any other multi-platform solution. Linux-based operating platforms typically work with a plethora of hardware (ARM, Intel, AMD) allowing for budget-friendly set-ups (cheap to purchase, install, and maintain). These set-ups may require slightly more time to set-up to ensure hardware and software behave as expected but most Linux operating systems offer solutions for this. Prebuilt Linux servers are also an option. Relative to the target userbase, system resources should not be in high demand as storing simple images, simple user credentials, and simple user data, as well as supporting basic interface for “Draw It or Lose It” does not require a heavy amount of processing power or storage. Again, the more users created as well as the more users online at once, the more resources required – that goes for any platform. There should be a consideration for ample breathing room in the event of an unexpected surge of users or a DOS (denial of service) attack in which resources are drained creating a poor user experience and making the system vulnerable.

Further considerations should be made concerning system security both physically and in cyberspace. (Details addressed in the Security section underneath Recommendations)

Server-side functionality will include hosting games with users on multiple or the same platforms, ensuring duplicate game instances, players, or teams are not instantiated and will have the responsibility of keeping a copy of each user’s data and credentials, game history, and other logged information (such as errors, system history, etc.).

Taking considerable time to ensure backend development is initially secure both functionally and from cyber-attacks may ensure significantly less time spent for maintenance and updates later on.

**Client-Side**

Frontend development may be accomplished on any personal computer operating platform. Catering to Apple-specific devices may be more easily accomplished (but is not recommended) with the use of devices supporting Mac OS. Development with Apple may be more secure as well as more efficient but does cost more in the end. Since mobile operating platforms are not limited to but including those Apple-specific are targeted user bases personal computers with Microsoft Windows 10 would be more than sufficient. The cost of the operating platform is still included in any system’s cost, yet any variety of systems may be selected. This includes budget-friendly platforms that are cheap to buy and maintain or bring-your-own-device solutions (which may pose a threat to security if this solution is accepted).

The client-side application will require user sign-in to identify the specific user playing the game as users must all be unique. A guest system may be integrated later that implements unique guest IDs but limits features such as creating a personalized username that is not kin to “guestID1234”. A user will only be permitted to play one game at a time but maybe signed into the “Draw It or Lose It” application on multiple devices. Note that the concept of multiple-device-sign-in is not set in stone, neither is the guest user concept.

Due to the simplicity of the graphical user interface (despite being animated which is required for rendering the image for gameplay) resource requirements are minimalistic for the frontend operation. The highly suggested use of mobile operating platforms allows for the adoption of systems with any budget.

“Draw It or Lose It” is also intended for use on any web browser. This is where a RESTful API (described below) comes in handy. It may take considerable development time to support both mobile devices as well as any browser, but this extends the user base with the greatest ease: streamlining more refined content to mobile users while also allowing an other device with a web browser to access the service.

**Development Tools**

Eclipse IDE for Java Developers is highly recommended for its ease of use and an enormous library of available plugins/add-ons. This development environment works well with a development tool known as Dropwizard which helps with some real heavy lifting when developing and maintaining an application with a RESTful API. (Skobalj, *Dropwizard Tutorial: DEVELOP RESTful Web Services Faster*)

A RESTful API, or a well-made implementation of the REST API, is an API that supports these functions (copied directly from Red Hat, *What is a REST API?*):

* A client-server architecture made up of clients, servers, and resources, with requests managed through HTTP.
* Stateless client-server communication, meaning no client information is stored between get requests and each request is separate and unconnected.
* Cacheable data that streamlines client-server interactions.
* A uniform interface between components so that information is transferred in a standard form. This requires that:
  + resources requested are identifiable and separate from the representations sent to the client.
  + resources can be manipulated by the client via the representation they receive because the representation contains enough information to do so.
  + self-descriptive messages returned to the client have enough information to describe how the client should process it.
  + hypertext/hypermedia is available, meaning that after accessing a resource the client should be able to use hyperlinks to find all other currently available actions they can take.
* A layered system that organizes each type of server (those responsible for security, load-balancing, etc.) involved the retrieval of requested information into hierarchies, invisible to the client.
* Code-on-demand (optional): the ability to send executable code from the server to the client when requested, extending client functionality.

*The following table assess alternatives for development against further recommendations. Advantages are denoted with “+”, disadvantages are denoted with “-”.*

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| ***Server Side*** | + Unix OS  + Open Source servers  + Secure  + User and file access controls  + Support LDAP and ADP  - Docker support virtual only  - Difficult to implement server-only environment | + Unix OS  + Open source servers  + Secure  + User and file access controls  + Support LDAP and ADP  + Docker support  - Need to decide on vendor support; Redhat or Ubuntu  + Large online/cloud support, the internet runs on Linux | + Secure  + User and file access controls  + Support ADP  - Need to use Server version  - Limited LDAP  - Poor open source server support*.* | + iOS developer tools  + iOS multi-device support (iPhone, iPad, watch, TV)  + Android developer tools  - Android poor multi-device support |
| **Client Side** | + Support Safari browser  + Firefox  + Chrome  - No IE  - Small install of client desktops | + Support Firefox  + Chrome  - No IE  - No Safari  - Small install of client desktops | + Support Large installed base of desktop clients  + Firefox  + Chrome  + IE  - no Safari | + iOS: Firefox, Chrome, Safari  + Android: Firefox, Chrome |
| **Development Tools** | + iOS native and Android development supported  - Small pool of developers  + High-productivity environment  - Difficult /impossible to test IE browsers  - Few IDEs, but high quality  + Large language support including Swift and objective-C  + Java support  + Unix shell scripting | *- Android dev support only: native*  *+ Large developer pool*  *+ New technology first platform*  *+ Largest set of IDEs with quality but no always support*  *+ Large language support including Swift and objective-C*  *+ .net support*  *+ Java support*  *+ Unix shell scripting*  *+ Powershell* | + Large set of IDEs with support and quality  - Android dev support only: native  + Large language support including C#  + .net support  + Powershell  + Java support | Development should not take place on mobile platforms. Perhaps the application may be tested on a mobile platform, but the limited development tools on mobile devices are far exceeded by those on other platforms. |

## Recommendations

1. **Operating Platform**: The recommended operating platform is the mobile operating platform because of the larger/ever expanding market share. For back-end processes a Linux based server or third-party option is recommended. Owning a Linux server is far cheaper than owning a server based on Mac or Windows because the software is typically free. Linux servers also have a lot of open-source software available for use and plenty of support options. A third party may also supply computing power for back-end processes at a rate that increases with use. A third party is able to provide maintenance, storage redundancy and set up at a cost much lower (the rate that increases with use) than there would be with owning a server.
2. **Operating Systems Architectures**: Mobile platforms are more abstracted towards consumers and are intended to be more streamlined. Mobile platforms run on ARM CPUs. Linux servers and third-party options are understood for being able to support multiple front-end platforms. Both options provide a way to manage memory, file storage, and multiple running applications. Linux can operate on any number of platforms: Intel, AMD, and ARM, 32-bit and 64-bit. A third-party solution may be managed (most likely) by a remote Linux based user interface. Though items such as storage and running applications may be more abstract as a single remote server would not be responsible for back-end operation, rather, parts of a collective whole. Files may be stored not on multiple hard drives, which is expected, but these drives may be spread out across a large physical area. When accessing data or making changes, the remote instance may be instantiated on one physical motherboard, closed, and later re-instantiated on another. This sort of process is an abstraction to a back-end user, however, and any actions needing to be performed by the user are streamlined by the service.
3. **Storage Management**:

Due to the aspects of gameplay for “Draw It or Lose It!”, a lot of high-resolution images need homes on hard drives – specifically two hundred of them. These images need to be rendered at a fixed rate (rendered in 30 seconds) in accordance with gameplay (during the set time that team members are meant to guess the puzzle). For the client to have full control over the rate of which an image is rendered the image needs to be downloaded and ready for use on the client device before the game is instantiated. Pulling the image from across the internet while the image is needing to be rendered may cause the image to render at a varying rate (the rendering process may speed up or slow to a halt). Simply storing the images locally, which would take up an estimated 1.56 gigabytes (based on 200 images averaging 8 megabytes in size each), is not the best solution regarding storing less on the client device. Other programs will need space as well and the client user may already have a substantial amount of local storage in use.

A mobile application such as “Draw It or Lose It!” should not require a full gigabyte of storage on the client-side platform – at least not in this case. Images could be stored on the server and downloaded by the client when a game is about to begin. Images could also be stored locally in an archival state, compressed for the sake of storage. This option would still require the use of significant storage space considering that mobile clients typically have less storage space than others. A hybrid solution of archiving data, data delivery across the internet, and locally storing images could be the most beneficial.

For users playing games across a local network without an internet connection or against simulated opponents while also offline (should these features ever be implemented) a fraction of the total 200 image library could be stored as compressed local files. A set of randomly selected images would be unarchived and loaded into the system’s memory every so many rounds. Doing this once a game is an option but would require more space in system memory than unarchiving a few images every few rounds instead.

The image files stored offline – say 75% of the total images (150 images) – could be stored server-side in an archival state, sent to the client upon request every few rounds of the game, and then unarchived by the client to avoid sending larger files. By not sending larger files, the download to the client device is faster and courteous towards the user’s internet subscription and also asks less of the local network.

1. **Memory Management**:

Memory management for the “Draw It or Lose It!” application is straight forward for both client and server-side development. The server will need to allocate appropriate resources for tasks such as games, specifically keeping track of game timers, sending drawings to the client after every few rounds and keeping track of scores. The server will also be responsible for user authentication, adding and keeping track of users – menial tasks. Though miniscule, measures should be put in place to ensure front-end and backend systems do not run out of memory during program execution, possibly even due to the execution of other programs. Running out of memory can cause issues all the way up to a system crash/failure which impacts functionality and security. Concepts like virtual memory (in Windows it is called a page file) can be used to cater to any overflow from physical memory (random access memory modules or RAM).

During games, the application can load images for a set number of rounds (loading in more images to memory after those rounds to replace images from previous rounds) that can be ready for rendering on screen in any given instant. It is best to have the images rendered from RAM rather than a hard drive, especially if the hard drive is not a solid state but a disk drive. This is because hard disk drives tend to be slower which would impact the continual 30 second rendering of the images displayed in game. The operability of this function is dependent of the client device’s installed memory as well as virtual memory management solutions (offered by the platform).

Server-side, the concern of hosting games and carrying out application functions will be dependent on available system memory as well as software solutions (virtual memory) offered by the platform. If the allotted resources are not sufficient (though assumingly quite small for the use case) the application could crash on both ends. I suggest the use of third-party services to streamline the back-end of the application.

1. **Distributed Systems and Networks**:

“Draw It or Lose It” may support online multiplayer. In the event of a faulty internet connection or any other lack of proper connection the LAN (Local Area Network) may be utilized to connect nearby players. A Linux-based server operating platform or Windows Server is recommended for cross-platform online multiplayer and as well as a RESTful API. LAN may be exclusive to nearby devices of the same or similar variant of a platform to reduce development costs.

A development solution that would implement the techniques discussed regarding archiving data, data delivery across the internet, and locally storing data involves microservices and serverless-architecture. Microservices are “a particular way of developing software, web or mobile applications as suites of independent service”. (Arsov, 2017) Serverless-architecture “is a cloud computing execution model where the cloud provider dynamically manages the allocation and provisioning of servers. A serverless application runs in stateless compute containers that are event-triggered, ephemeral (may last for one invocation), and fully managed by the cloud provider”. (Bashir, 2019)

Both microservices and serverless-architecture embody the concept of letting a third-party service manage server-end needs at a cost that is typically lower and a means that is more efficient in overall execution. The following bullet point list is provided by Faizan Bashir on freecodecamp.org and relates quite specifically to not only memory and storage management but also the overall functionality of “Draw It or Lose It!”:

The Serverless App

A Serverless solution consists of a web server, Lambda functions (FaaS), security token service (STS), user authentication and database.

• Client Application — The UI of your application is rendered client side in Modern Frontend Javascript App which allows us to use a simple, static web server.

• Web Server — Amazon S3 provides a robust and simple web server. All of the static HTML, CSS and JS files for our application can be served from S3.

• Lambda functions (FaaS) — They are the key enablers in Serverless architecture. Some popular examples of FaaS are AWS Lambda, Google Cloud Functions and Microsoft Azure Functions. AWS Lambda is used in this framework. The application services for logging in and accessing data will be built as Lambda functions. These functions will read and write from your database and provide JSON responses.

• Security Token Service (STS) — generates temporary AWS credentials (API key and secret key) for users of the application. These temporary credentials are used by the client application to invoke the AWS API (and thus invoke Lambda).

• User Authentication — AWS Cognito is an identity service which is integrated with AWS Lambda. With Amazon Cognito, you can easily add user sign-up and sign-in to your mobile and web apps. It also has the options to authenticate users through social identity providers such as Facebook, Twitter or Amazon, with SAML identity solutions, or using your own identity system.

• Database — AWS DynamoDB provides a fully managed NoSQL database. DynamoDB is not essential for a serverless application but is used as an example here.

The above list depicts not only the usage cases for serverless-architecture (The Editors of Encyclopaedia Britannica, *Client-Server Architecture*), but scenarios for functionality fulfillment for the “Draw It or Lose It!” application. In other words, the benefits of using this architecture go well beyond memory and storage management. Bashir touches on how the serverless architecture is better for business in being cost effective and provides faster delivery to market, better for developers by simplifying system administration, enhancement and scalability to the exact needs of the application at any given time, and better for users by being better catered to by businesses and connecting the application to services that they already use like their Google account as a means of signing in or their own server-side storage like Google Drive. It’s also important to note “…that these kinds of apps may offer client-side caching, which provides a better offline experience”. (Bashir, 2019)

Kristijan Arsov goes in detail about what microservices are in their article “Microservices vs. SOA — Is There Any Difference at All?”, where microservices are compared to an older concept known as service-oriented architecture (SOA). According to Arsov, SOA is better suited for enterprise/large business-type applications. Microservices, on the other hand, “are the next step in the evolution of Service-Oriented Architectures”. (Arsov, 2017) The use of “lightweight HTTP, REST or THRIFT APIs” help to define microservices, which can carry out functions using less resources and complications than traditional SOA of which Arsov describes as being a concept of its own. In their article conclusion, Arsov notes that microservices “are better suited for smaller and well-partitioned web based systems”. Microservices are also easier to manage regarding storage, as each microservice an application uses can have its own storage (such a microservice made specifically for authenticating users/storing credentials). Also, should a microservice have a memory fault, it will not affect other microservices whereas a memory issue with SOA could back up the whole system. (Arsov, 2017)

1. **Security**: The application, “Draw It or Lose It”, should implement the principle of least privilege. “It dictates that programs, users, and even systems be given just enough privileges to perform their tasks.” (Silbershatz, Galvin, Gagne, Weisman, Santor, 2009) (Chapter 14)

Role-based access control, or RBAC, is a form of the principle of least privilege. Users are assigned roles/ given their accounts. Examples include users who are admins, who have full privileges, or users who are players, who have limited privileges. Users without accounts, designated by the guest role, will have the barest of privileges (if any). (Silbershatz, Galvin, Gagne, Weisman, Santor, 2009) (Chapter 14)

Keeping the level of application complexity low would work well at addressing security concerns. Development would be better able to assess security breaches or concerns while attackers would have fewer angles to find a security hole. (Silbershatz, Galvin, Gagne, Weisman, Santor, 2009) (Chapter 14)

Consideration must also be given to the physical protection of server-side and client-side development. Computer systems used to develop the application should be guarded under lock and key. The principle of least privilege comes handy again, preventing the entirety of the system from being destroyed, if a mishap were to only happen in one sector, governed/accessed by a less trusted user, such as a graphic designer spilling soda on their work computer or accidentally downloading malicious software, making it available to the local network shared by other graphic designers. A systems administrator, specified as being a more trustworthy individual, would also have only the necessary resources for completing their tasks such as access to the main server, which only handles back up and maintenance of systems and provides no direct access to items such as digital artwork or the game’s source code. (Silbershatz, Galvin, Gagne, Weisman, Santor, 2009) (Chapter 14)

In more malicious circumstances, involving attackers or unauthorized users, the first line of defense needs to be social engineering. Members of The Gaming Room should be well informed and aware of the dangers within cyberspace. Failsafe countermeasures such as the implementations of firewalls, digital tools used to partition a local network from the rest of the internet as well as create partitions within itself. The idea behind a firewall is to allow solely what is permissible to pass through to the system it is trying to protect. Antimalware/antivirus programs are also good for active protection/surveillance over a system. This type of software can also help prevent an incompetent user from accessing malicious software on accident. Encrypting data so that it may not be easily read if it is intercepted would also be wise. (Silbershatz, Galvin, Gagne, Weisman, Santor, 2009) (Chapter 15)

## Definitions

**Singleton** – “Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.”

The singleton pattern is a programming structure where a class is self-instantiating, has only one instance of itself, and protects its constructor so that it may not be instantiated again elsewhere. Instead of normally calling to a variable where a class has been used to instantiate an object the class name is called instead. In this software, it is common to find that a variable is still created, not for creating an instance of the class, but for storing a pointer to the object instance within the class. (Tutorials Point, Design Pattern - Singleton Pattern)

**Iterator** – “Iterator pattern is very commonly used design pattern in Java and .Net programming environment. This pattern is used to get a way to access the elements of a collection object [such as the array lists used to store games, teams, and players] in sequential manner without any need to know its underlying representation.

Iterator pattern falls under behavioral pattern category.” (Tutorials Point, Design Patterns - Iterator Pattern)

**Object**– The core component of object-oriented programing. Objects are defined as having states and behaviors. For a non-programming example: a car can be on or off (states), turn left or right, and drive forwards or backward (behaviors). (Tutorials Point, Java - Object and Classes)

**Class** – “A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.” (Tutorials Point, Java - Object and Classes)

**Method/Function** – Methods, also known as functions, are intended to be reusable pieces of code that run when called upon. Methods can also take is data as a parameter to aid in execution. (W3Schools, Java Methods)

**List**– The various functions of this class can make sorting through a collection easier by offering the functionality of its parent class, Collection. List also brings along methods specific to accessing information about a collective or modifying entries. (Tutorials Point, Java - the list interface)

**UML Class Diagram** – “Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.” (Tutorials Point, UML - class diagram)

In the class diagram for “Draw It or Lose It” we see that Game, Team, and Player **inherit** from Entity. GameService interreacts with Game and ProgramDriver interreacts with SingletonTester.

**Inheritance** – “Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.” (JavaTpoint, Inheritance in Java)

Player is a child object of Entity. Player is an Entity. Exactly so, Player has all the same functions as any instantiation of the Entity class would, the same variables too. Player overrides the toSring function and adds some of its functionality to the table as well. The same goes for Team and Games concerning their inheritance from Entity.

**Polymorphism** – Polymorphs or something (programming or not) that is characteristic of polymorphism is a thing capable of presenting itself in more than one form. Specific to programming (like in the java language) and our example, instantiations of the class Entity have methods of which carry out different computations given their class type. In the subclasses of Entity, the Team, Game, and Player classes override the toString method in their way. (JavaTpoint, Polymorphism in Java)

**Encapsulation** – Variables and methods that are not global are stored (encapsulated) inside of any sort of bracketed statements and are only directly accessible through that statement and its sub-statements. A private integer variable named temp is only accessible in the if-statement it was declared by. In our example, there are private variables, id and name, which only directly accessible underneath/within the Entity class. Note, private variables/methods are fully encapsulated by their parent statement. Protected variables/methods are accessible by statements in the same directory. Public variables/methods are not encapsulated at all.

Encapsulation is also known as data hiding. (Tutorials Point, Java - Encapsulation)

**Abstraction** – Classes and methods made abstract use the abstract keyword and are italicized when displayed in UML diagrams. Abstract classes cannot be instantiated and are only utilized by non-abstract classes that inherit from it. Abstract methods can only exist within abstract classes. (W3Schools, Java Abstraction)

In “Draw It or Lose It”, the class Entity has the potential to be written as an abstract class. Since the UML class diagram provided did not represent it as such and must be adhered to, it was not made abstract.

The Entity class is used exactly as if it were abstract, though. No calls are made to instantiate it (as none are ever needed) other than to instantiate its non-abstract subclasses. The toString method might as well be made abstract since there is no need for its definition in the parent class.

## Citations

Tutorials Point. (n.d.). Design Pattern - Singleton Pattern. Retrieved March 27, 2021, from <https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm>

Tutorials Point. (n.d.). Design Patterns - Iterator Pattern. Retrieved March 27, 2021, from <https://www.tutorialspoint.com/design_pattern/iterator_pattern.htm>

Tutorials Point. (n.d.). Java - Object and Classes. Retrieved March 27, 2021, from <https://www.tutorialspoint.com/java/java_object_classes.htm>

W3Schools. (n.d.). Java Methods. Retrieved March 27, 2021, from <https://www.w3schools.com/java/java_methods.asp>

Tutorials Point. (n.d.). Java - the list interface. Retrieved March 27, 2021, from <https://www.tutorialspoint.com/java/java_list_interface.htm>

Tutorials Point. (n.d.). UML - class diagram. Retrieved March 27, 2021, from <https://www.tutorialspoint.com/uml/uml_class_diagram.htm>

JavaTpoint. (n.d.). Inheritance in Java. Retrieved March 28, 2021, from <https://www.javatpoint.com/inheritance-in-java#:~:text=Inheritance%20in%20Java%20is%20a,behaviors%20of%20a%20parent%20object.&text=The%20idea%20behind%20inheritance%20in,fields%20of%20the%20parent%20class.>

JavaTpoint. (n.d.). Polymorphism in Java. Retrieved March 29, 2021, from <https://www.javatpoint.com/runtime-polymorphism-in-java#:~:text=Polymorphism%20in%20Java%20is%20a,single%20action%20in%20different%20ways.&text=So%20polymorphism%20means%20many%20forms,method%20overloading%20and%20method%20overriding.>

Tutorials Point. (n.d.). Java - Encapsulation. Retrieved March 29, 2021, from <https://www.tutorialspoint.com/java/java_encapsulation.htm>

W3Schools. (n.d.). Java Abstraction. Retrieved March 29, 2021, from <https://www.w3schools.com/java/java_abstract.asp>

Hook, T. (n.d.). I Want to Develop Android Apps – What Languages Should I Learn? Retrieved March 29, 2021, from <https://www.zipcodewilmington.com/blog/i-want-to-develop-android-apps-what-languages-should-i-learn#:~:text=The%20official%20language%20for%20Android,t%20something%20that%20Google%20promotes.>

Silbershatz, A., Galvin, P. B., &amp; Gagne, G. (2009). Operating System Concepts (8th ed.) (C. Weisman & K. Santor, Eds.). Hoboken, NJ: John Wiley &amp; Sons.

Skobalj, V. (n.d.). Dropwizard Tutorial: DEVELOP RESTful Web Services Faster. Retrieved April 08, 2021, from <https://stackabuse.com/dropwizard-develop-restful-web-services-faster/>

Red Hat. (n.d.). What is a REST API? Retrieved April 08, 2021, from <https://www.redhat.com/en/topics/api/what-is-a-rest-api>

Arsov, K. (2017, November 10). Microservices vs. SOA — Is There Any Difference at All? Retrieved April 11, 2021, from <https://kikchee.medium.com/microservices-vs-soa-is-there-any-difference-at-all-2a1e3b66e1be>

Bashir, F. (2019, July 13). What is Serverless Architecture? What are its Pros and Cons? Retrieved April 11, 2021, from <https://www.freecodecamp.org/news/what-is-serverless-architecture-what-are-its-pros-and-cons/>

The Editors of Encyclopaedia Britannica. (n.d.). Client-Server Architecture. Retrieved April 11, 2021, from <https://www.britannica.com/technology/client-server-architecture>