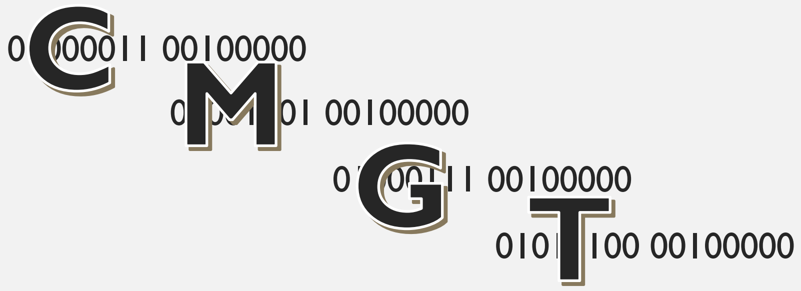
SOFTWARE ARCHITECTURE

Think Big and Keep It Simple

WHAT IS SOFTWARE ARCHITECTURE?

IN THEORY: ABSTRACTION

- An "intellectually graspable" abstraction of a complex system.



- Humans need abstraction to interact with computers…

IN THEORY: ABSTRACTION

- …and programmers too.

```
Move();
HandleCollisions(enemies);
UpdatePlayerState();
UpdateGameState();
```

```
[illegible] gravity; this.x+= velocity.x; this.y+= velocity.y;
    foreach(GameObject aGameObject in enemies){ if(aGameObject.transform.x >
this.x+this.width) this.health--;}
    if([illegible]
    gameOver = true; enemies.Clear();}
```

IN PRACTICE: DISCIPLINE

In big companies: write the code the company’s way or the highway.

In small companies: usually it’s the bad software architecture(not bad UI/UX design!) that causes a project to fail miserably.

Indie studios or freelancers: good architecture saves precious time and even more precious money.

School projects: a good foundation in the first week make your debugging less painful in later weeks.

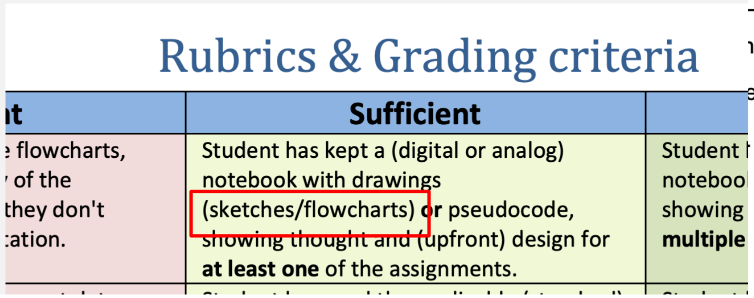
IN PRACTICE: DISCIPLINE

So let’s begin with the assignment of this module!

Manual time!

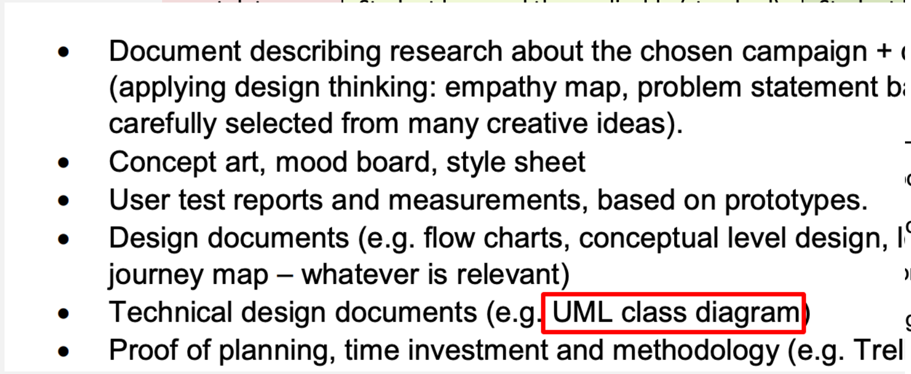
- First, lets get one thing out of the way…

REMEMBER THESE?

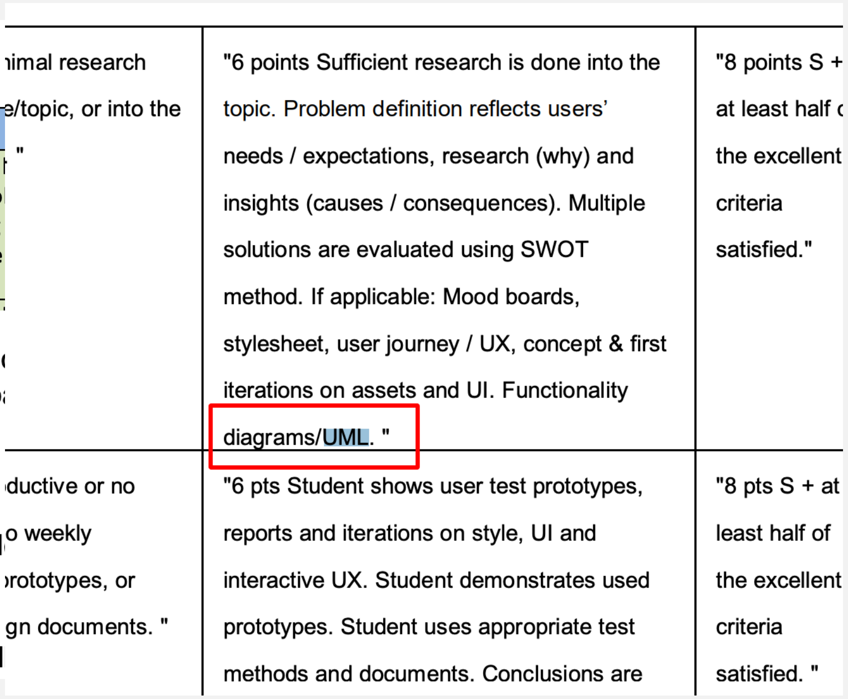


Rubrics & Grading criteria

t	Sufficient	
flowcharts, of the they don't ation.	Student has kept a (digital or analog) notebook with drawings (sketches/flowcharts) or pseudocode, showing thought and (upfront) design for **at least one** of the assignments.	Student notebook showing **multiple**



- Document describing research about the chosen campaign + (applying design thinking: empathy map, problem statement b carefully selected from many creative ideas).
- Concept art, mood board, style sheet
- User test reports and measurements, based on prototypes.
- Design documents (e.g. flow charts, conceptual level design, l journey map – whatever is relevant)
- Technical design documents (e.g. UML class diagram)
- Proof of planning, time investment and methodology (e.g. Trel



nimal research e/topic, or into the "	"6 points Sufficient research is done into the topic. Problem definition reflects users' needs / expectations, research (why) and insights (causes / consequences). Multiple solutions are evaluated using SWOT method. If applicable: Mood boards, stylesheet, user journey / UX, concept & first iterations on assets and UI. Functionality diagrams/UML. "	"8 points S + at least half the excellent criteria satisfied."
ductive or no o weekly rototypes, or gn documents. "	"6 pts Student shows user test prototypes, reports and iterations on style, UI and interactive UX. Student demonstrates used prototypes. Student uses appropriate test methods and documents. Conclusions are	"8 pts S + at least half of the excellent criteria satisfied. "

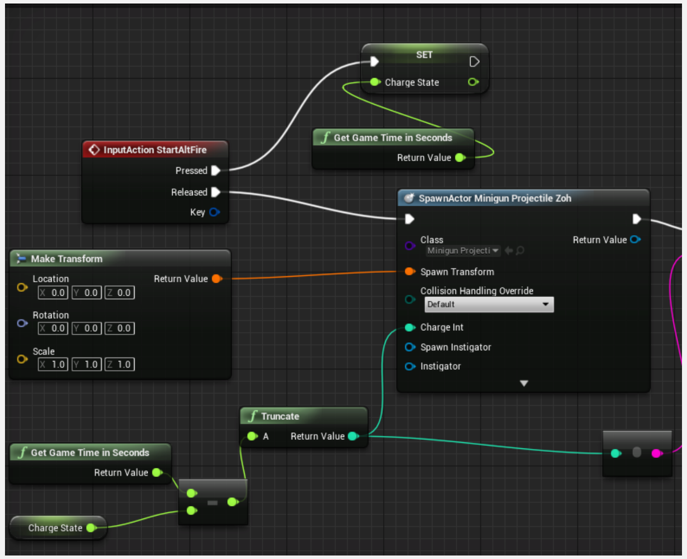
UML

- Feedback:
  - I don’t know what is UML.
  - I know what UML is, and I hate it.
- Questions:
  - Seriously, what is UML?
  - What’s the purpose?(Other than pleasing teachers and getting a good grade)
  - When and how to use it?

WHAT IS UML?

- Before we formally introduce UML, let’s meet some of its cousins…

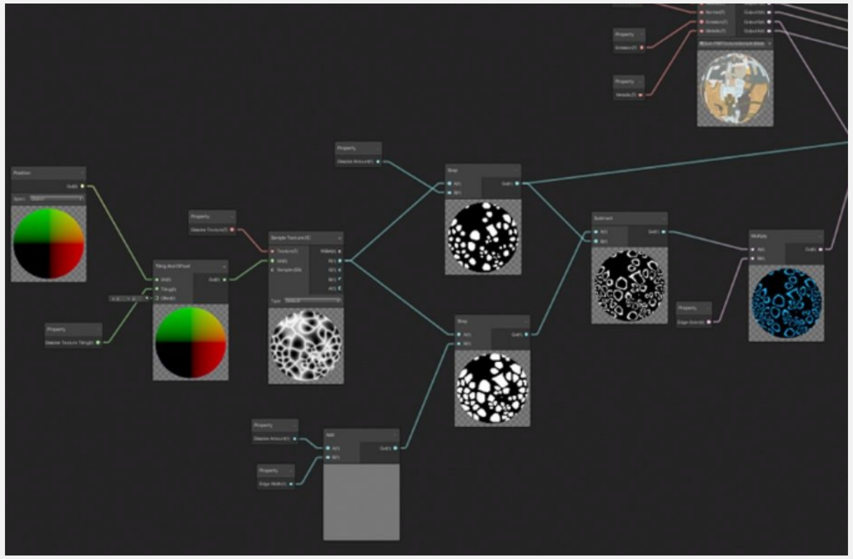
WHAT IS UML?



Blueprint, preferred by many over Unity.

Why?

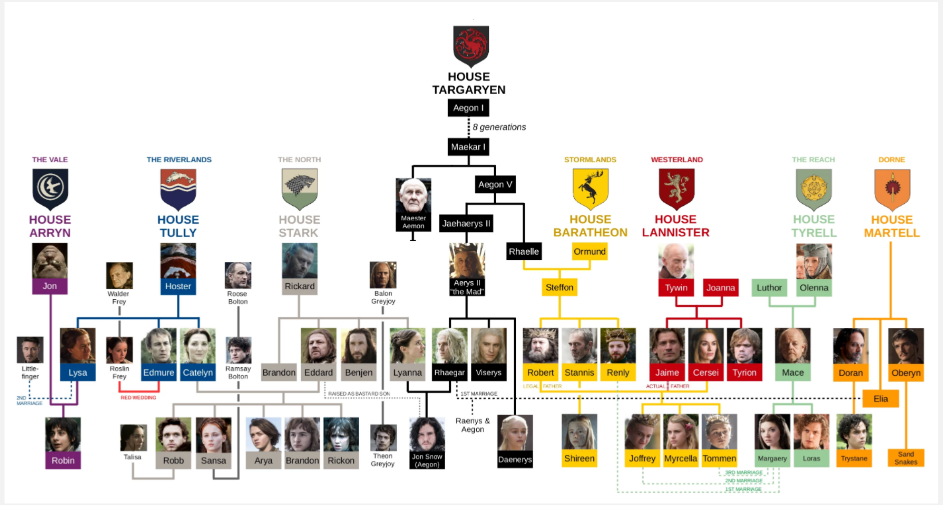
WHAT IS UML?



Shadergraph, developed to replace shader language.

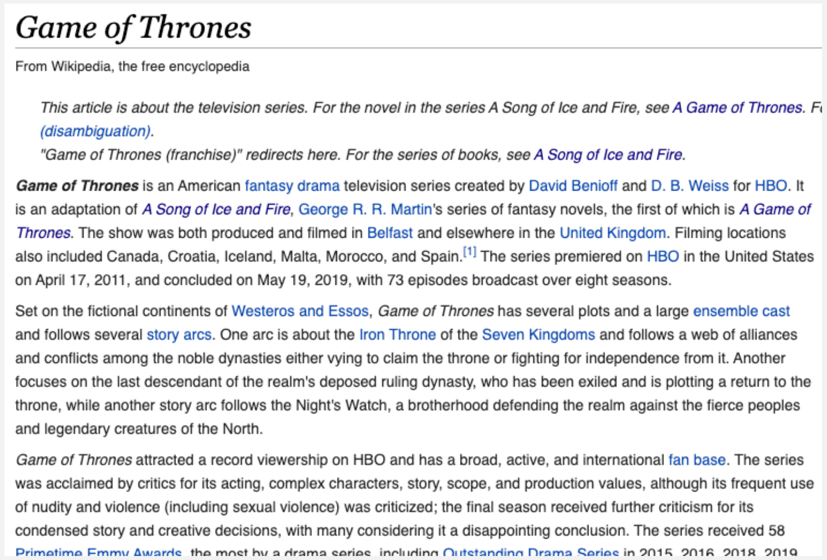
Hated by 3d Rendering teachers.

WHAT IS UML?



If you are building a database for all the family relations for a GOT game, would you prefer this…

WHAT IS UML?



Game of Thrones

From Wikipedia, the free encyclopedia

This article is about the television series. For the novel in the series A Song of Ice and Fire, see A Game of Thrones. F
(disambiguation).
"Game of Thrones (franchise)" redirects here. For the series of books, see A Song of Ice and Fire.

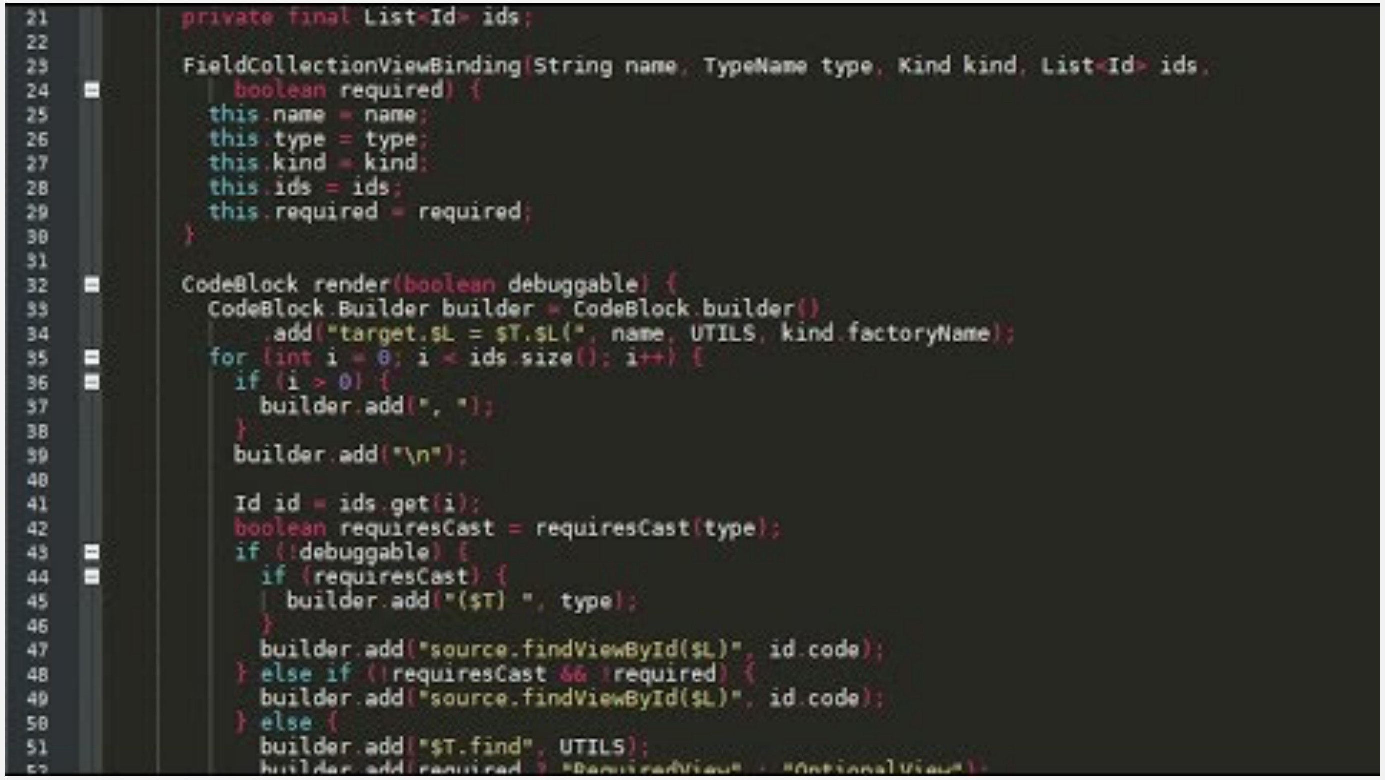
Game of Thrones is an American fantasy drama television series created by David Benioff and D. B. Weiss for HBO. It is an adaptation of *A Song of Ice and Fire*, George R. R. Martin's series of fantasy novels, the first of which is *A Game of Thrones*. The show was both produced and filmed in Belfast and elsewhere in the United Kingdom. Filming locations also included Canada, Croatia, Iceland, Malta, Morocco, and Spain.[1] The series premiered on HBO in the United States on April 17, 2011, and concluded on May 19, 2019, with 73 episodes broadcast over eight seasons.

Set on the fictional continents of Westeros and Essos, *Game of Thrones* has several plots and a large ensemble cast and follows several story arcs. One arc is about the Iron Throne of the Seven Kingdoms and follows a web of alliances and conflicts among the noble dynasties either vying to claim the throne or fighting for independence from it. Another focuses on the last descendant of the realm's deposed ruling dynasty, who has been exiled and is plotting a return to the throne, while another story arc follows the Night's Watch, a brotherhood defending the realm against the fierce peoples and legendary creatures of the North.

Game of Thrones attracted a record viewership on HBO and has a broad, active, and international fan base. The series was acclaimed by critics for its acting, complex characters, story, scope, and production values, although its frequent use of nudity and violence (including sexual violence) was criticized; the final season received further criticism for its condensed story and creative decisions, with many considering it a disappointing conclusion. The series received 58 Primetime Emmy Awards, the most by a drama series, including Outstanding Drama Series in 2015, 2016, 2018, 2019

Or this?

EXPLAINING A LARGE PROJECT WITHOUT UML



```java
private final List<Id> ids;

FieldCollectionViewBinding(String name, TypeName type, Kind kind, List<Id> ids,
    boolean required) {
  this.name = name;
  this.type = type;
  this.kind = kind;
  this.ids = ids;
  this.required = required;
}

CodeBlock render(boolean debuggable) {
  CodeBlock.Builder builder = CodeBlock.builder()
      .add("target.$L = $T.$L(", name, UTILS, kind.factoryName);
  for (int i = 0; i < ids.size(); i++) {
    if (i > 0) {
      builder.add(", ");
    }
    builder.add("\n");

    Id id = ids.get(i);
    boolean requiresCast = requiresCast(type);
    if (!debuggable) {
      if (requiresCast) {
        builder.add("($T) ", type);
      }
      builder.add("source.findViewById($L)", id.code);
    } else if (!requiresCast && !required) {
      builder.add("source.findViewById($L)", id.code);
    } else {
      builder.add("$T.find", UTILS);
      builder.add(required ? "RequiredView" : "OptionalView");
```

WHAT IS UML?

- Programming languages are not abstract enough for an object oriented design, therefore some OOP researchers invented UML.
- UML stands for **U**nified **M**odelling **L**anguage.
- It is a notation standard widely used by many companies.

USES FOR UML

- As a sketch:
- Forward design: UML before coding.
- Backward design: UML after coding.
- As a blueprint
- Requires much more experience.
- Very hard to get it right in the first few tries.
- As pseudo code:
- Handy when designing algorithms.

UML DIAGRAMS

UML2.0 has 14 types of diagrams.

The diagrams have two categories: Structural diagrams and Behavioral diagrams.

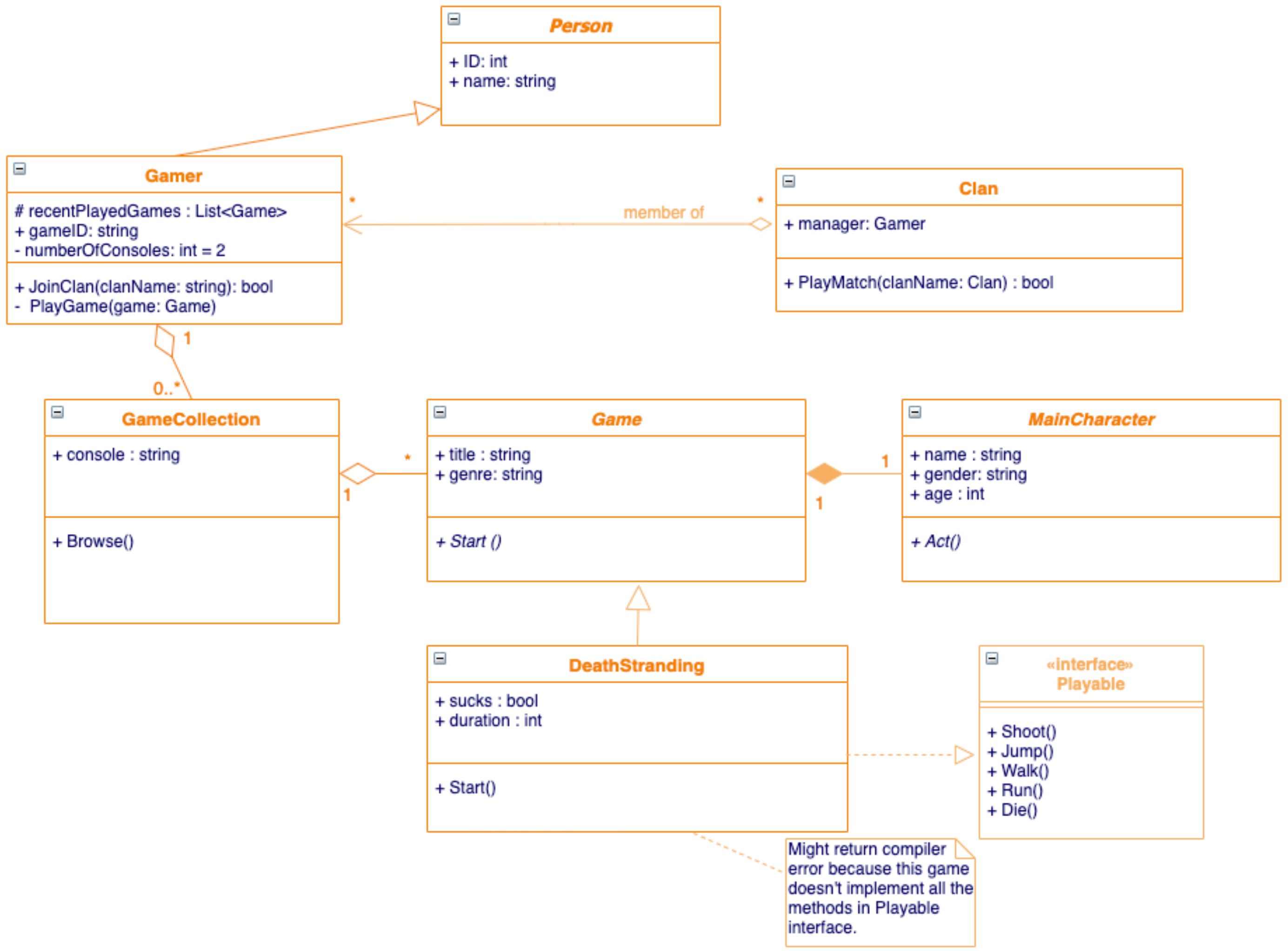
Out of the 14, only 4 types of diagrams are frequently used.

Out of the 4, only 1 type of diagrams are used in this module.

Class diagrams: one diagram to rule the all!

CLASS DIAGRAMS

- When you see 'UML' in rubrics, mostly it means class diagrams.
- Class diagram is a static diagram to visualize your code's structure.
- It can't be used in workflows to show how classes interact with each other at a specific point in time, or to show how a particular behavior is implemented.(That would be like using GOT family tree to describe what happened in season 8(no diagrams can do this))



Might return compiler error because this game doesn't implement all the methods in Playable interface.

CLASS ATTRIBUTES(FIELDS)

Name conventions: Class, *AbstractClass*, <<interface>>

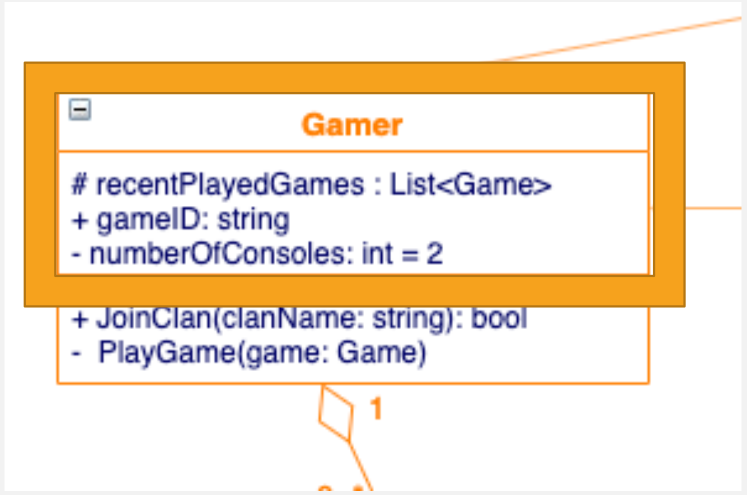
visibility name : *type (= default_value)*

Visibility: + public

- private

# protected

Underline <u>static attributes</u>


Gamer
# recentPlayedGames : List<Game>
+ gameID: string
- numberOfConsoles: int = 2
+ JoinClan(clanName: string): bool
- PlayGame(game: Game)
1

CLASS OPERATIONS(METHODS)

visibility name(*parameters*): *return_type*

Visibility: + public

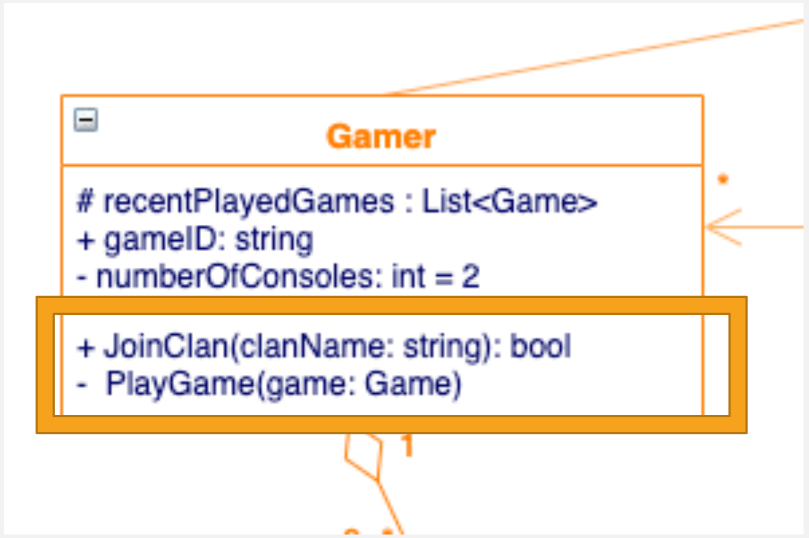
- private

# protected

Underline <u>static methods</u>

Parameter as (name:type)

Omit *return type* when return type is void.

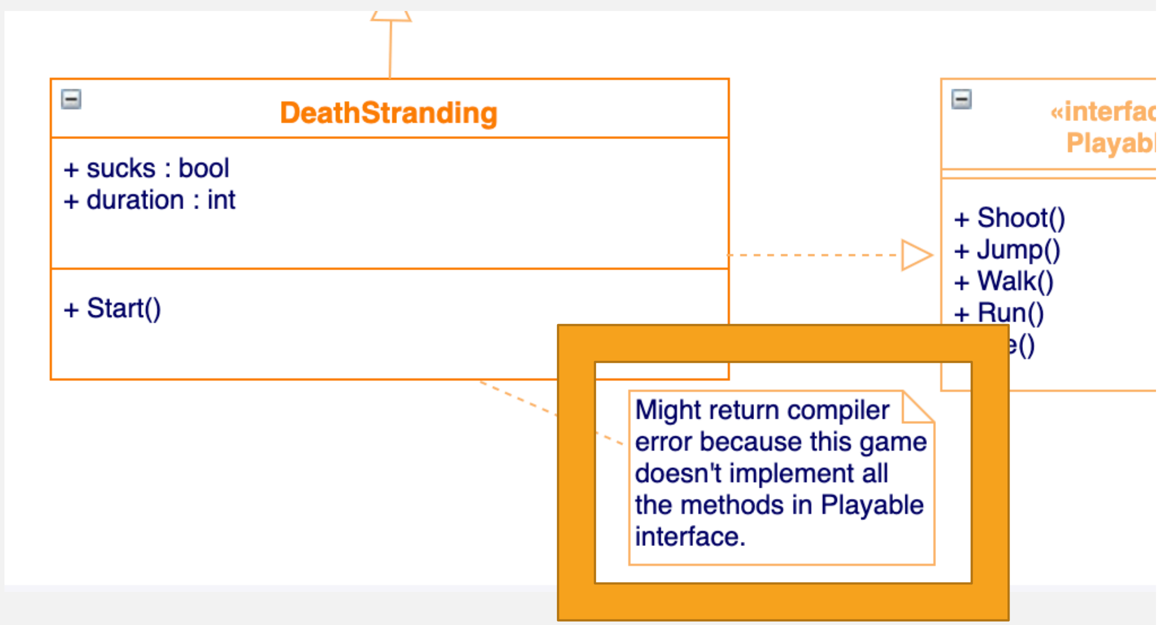


```
# recentPlayedGames : List<Game>
+ gameID: string
- numberOfConsoles: int = 2

+ JoinClan(clanName: string): bool
-  PlayGame(game: Game)
```

COMMENTS

Folded note attached to the target class/operation/etc. by a dashed line.



RELATIONSHIPS BETWEEN CLASSES

Generalization(inheritance)

Inheritance between classes

Interface implementation.

Association(usage)

Dependency

Aggregation

Composition

GENERALIZATION

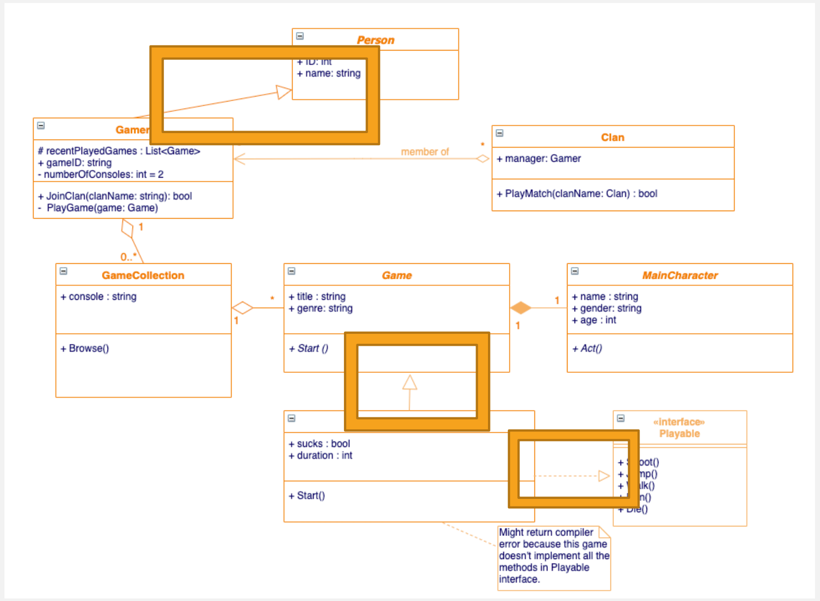
Top to down hierarchies.

Arrows point upward to parent.

Parent is a class: solid line, black(colored) arrow.

Parent is an abstract class: solid line, white arrow.

Implementing an interface: dashed line, white arrow



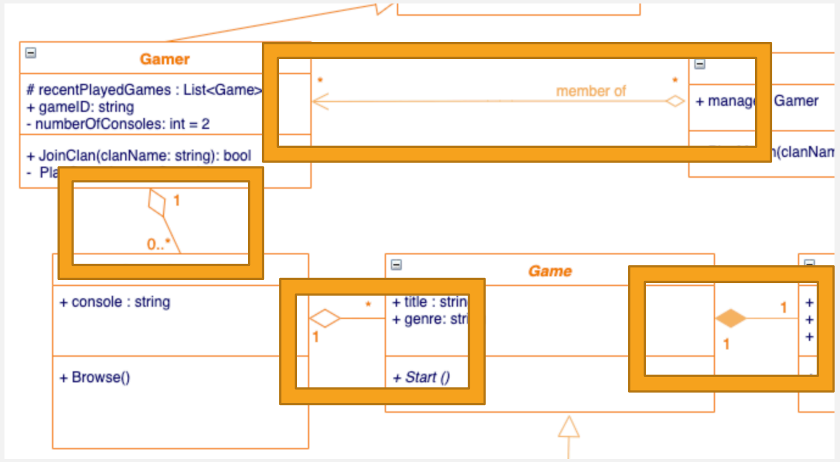
ASSOCIATION

Multiplicity:

- * : 0 or above
- 1 : exactly 1
- m..n : between m and n(inlusive)
- 1..* : 1or above

Name: describing relationship.

Navigability: direction of usage.

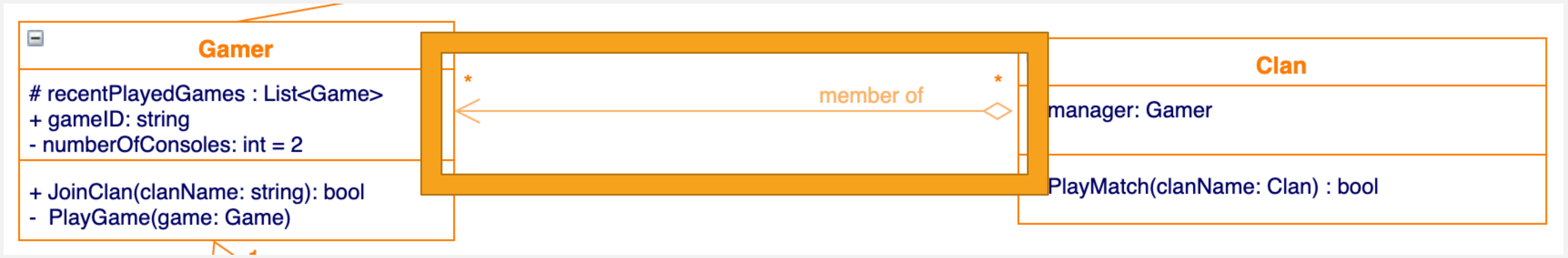


AGGREGATION

White diamond.

Part to whole relationship.

One **can exist** without the other.

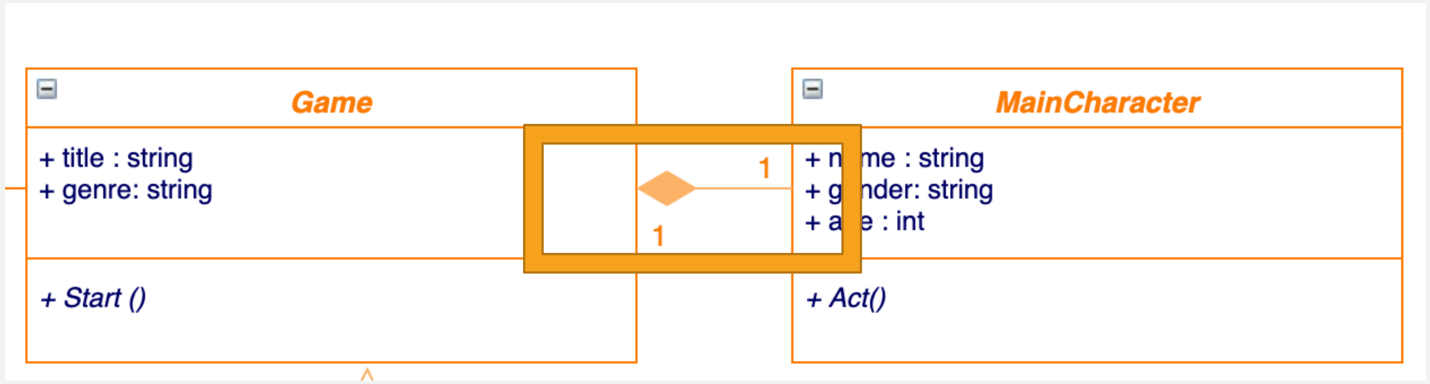

Gamer
# recentPlayedGames : List<Game>
+ gameID: string
- numberOfConsoles: int = 2
+ JoinClan(clanName: string): bool
- PlayGame(game: Game)
*
member of
*
Clan
manager: Gamer
PlayMatch(clanName: Clan) : bool

COMPOSITION

Black(colored) diamond.

Part to whole relationship.

One **can't exist** without the other.



UML IN THIS MODULE

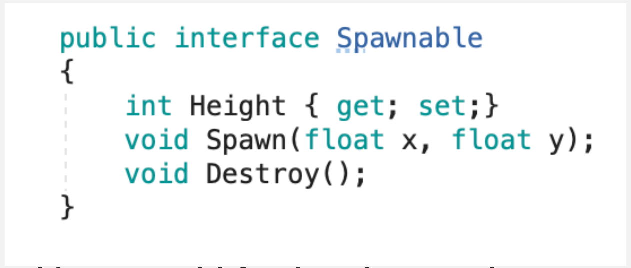
- A class diagram to visualize their code sketch before implementation and a class diagram to visualize the final code structure after the implementation.
- Use https://www.draw.io/ or https://staruml.io/, or any other tool you prefer.

PREPARATION FOR THE ASSIGNMENT

- Interface
- Generics
- Delegate and event
- Unit test

INTERFACE

- Interface is like a contract a class signs, the class implementing the interface must define all the methods defined in the interface.



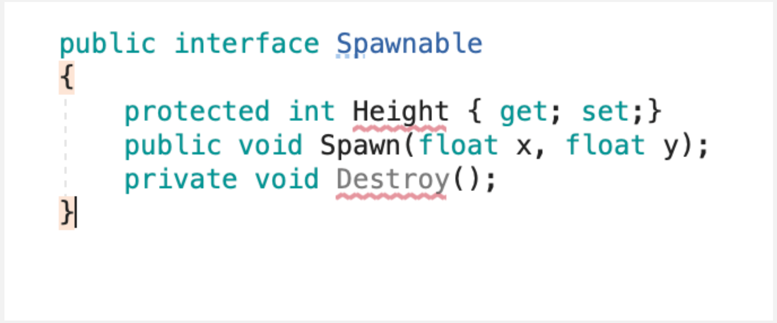
```
public interface Spawnable
{
    int Height { get; set;}
    void Spawn(float x, float y);
    void Destroy();
}
```

- This works because like in real life, the client only cares about what you must do, not how you do it.

INTERFACE: TECHNICAL DETAILS

Technical details:

All members are by default public, you can’t use any access modifier(public, private, protected) on them:

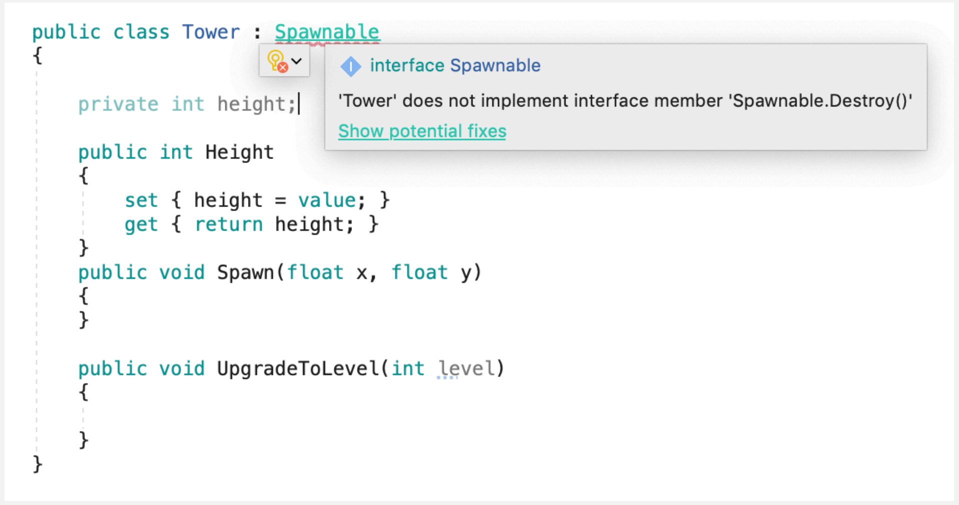


```
public interface Spawnable
{
    protected int Height { get; set;}
    public void Spawn(float x, float y);
    private void Destroy();
}
```

INTERFACE: TECHNICAL DETAILS

Technical details:

The class that implements the interface must define all the members in the interface.



```
public class Tower : Spawnable
{

    private int height;

    public int Height
    {
        set { height = value; }
        get { return height; }
    }
    public void Spawn(float x, float y)
    {
    }

    public void UpgradeToLevel(int level)
    {

    }
}
```

interface Spawnable

'Tower' does not implement interface member 'Spawnable.Destroy()'

Show potential fixes

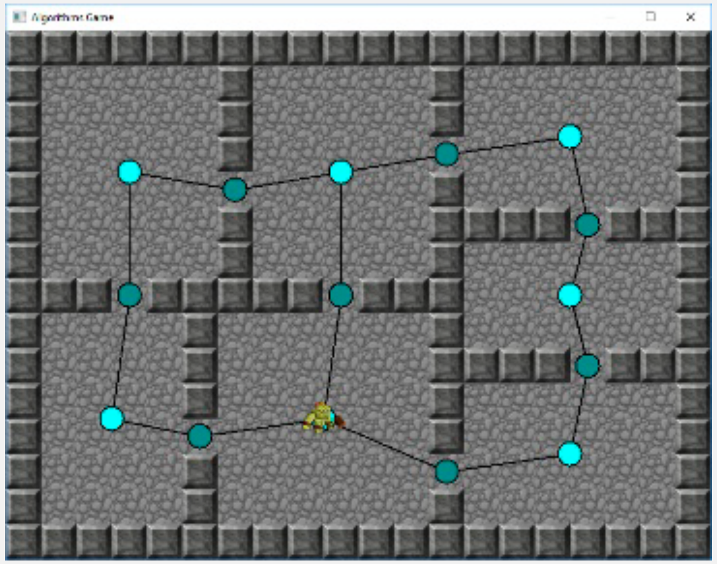
INTERFACE: USAGE

Plug and play.

By separating the implementation from the interface, we make the code more reusable, check the example:

INTERFACE: EXAMPLE

Let's meet an old friend:



You put so much work into this, what if you want to do the same thing again in Unity?

INTERFACE: EXAMPLE

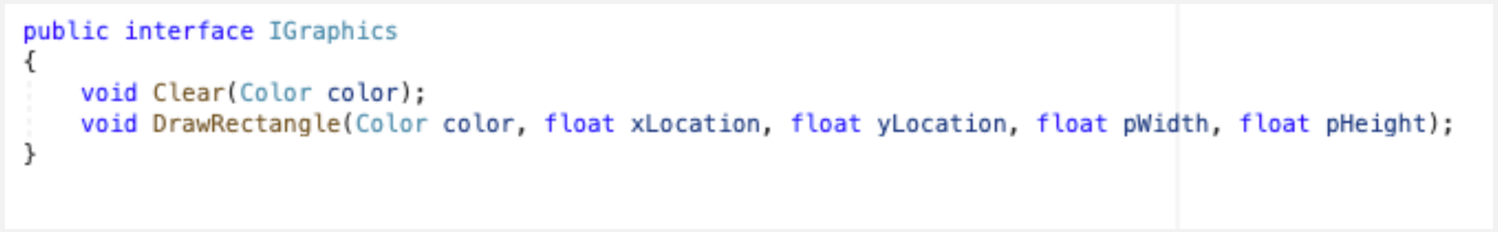
Notice that in Dungeon.cs, the only functions that handle the graphic drawing are:

graphics.Clear() and graphics.DrawRectangle().

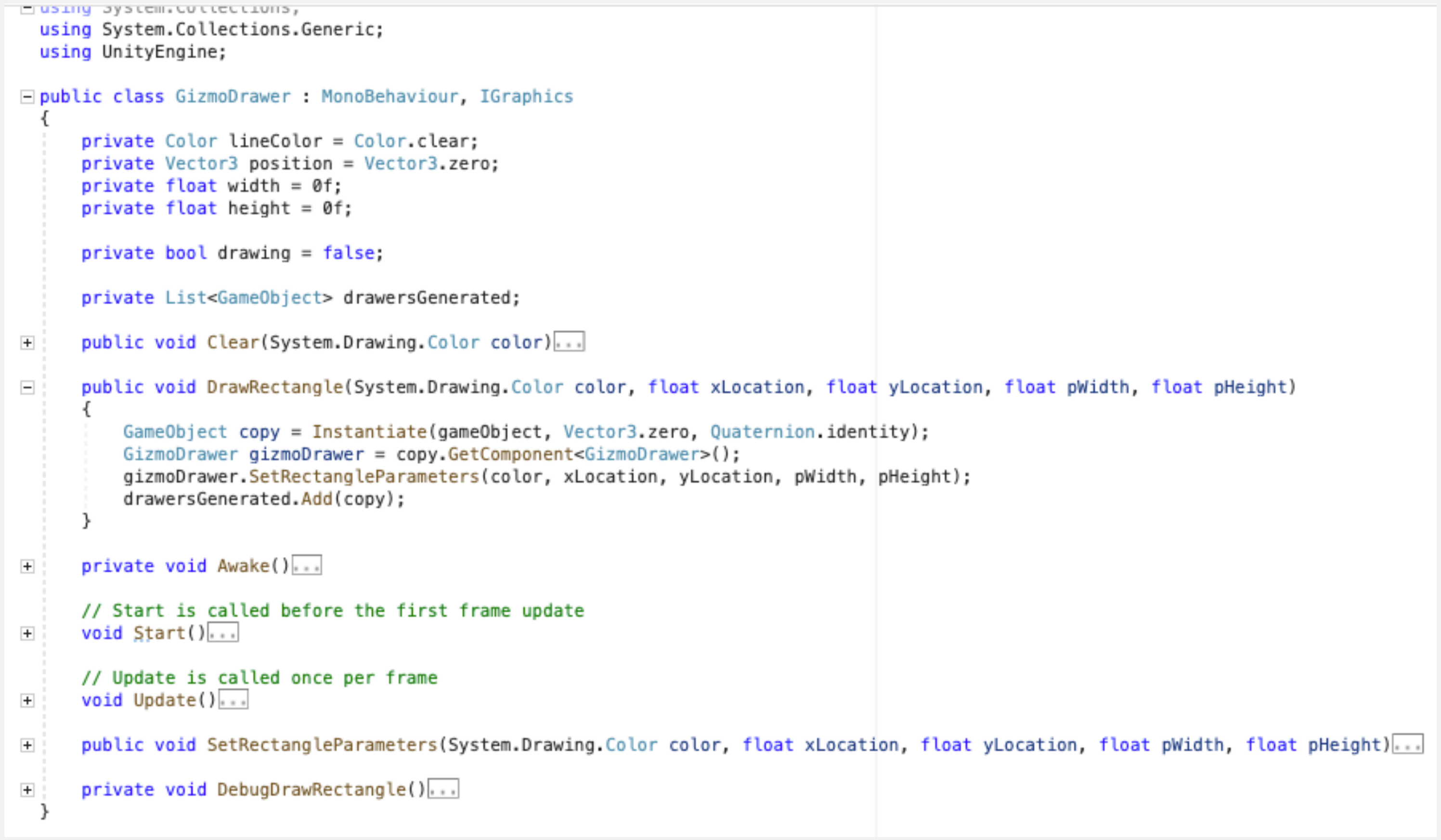
The dungeon generator doesn’t care about how to draw the graphics, it only call these methods to output the dungeon.

Therefore, we can make an interface to handle the drawing.

INTERFACE: EXAMPLE


```
public interface IGraphics
{
    void Clear(Color color);
    void DrawRectangle(Color color, float xLocation, float yLocation, float pWidth, float pHeight);
}
```

- Unity is picky because it only welcomes MonoBehaviour
- But we can force it to work by plugging in our interface →
- Take that Unity!



```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GizmoDrawer : MonoBehaviour, IGraphics
{
    private Color lineColor = Color.clear;
    private Vector3 position = Vector3.zero;
    private float width = 0f;
    private float height = 0f;

    private bool drawing = false;

    private List<GameObject> drawersGenerated;

    public void Clear(System.Drawing.Color color)...

    public void DrawRectangle(System.Drawing.Color color, float xLocation, float yLocation, float pWidth, float pHeight)
    {
        GameObject copy = Instantiate(gameObject, Vector3.zero, Quaternion.identity);
        GizmoDrawer gizmoDrawer = copy.GetComponent<GizmoDrawer>();
        gizmoDrawer.SetRectangleParameters(color, xLocation, yLocation, pWidth, pHeight);
        drawersGenerated.Add(copy);
    }

    private void Awake()...

    // Start is called before the first frame update
    void Start()...

    // Update is called once per frame
    void Update()...

    public void SetRectangleParameters(System.Drawing.Color color, float xLocation, float yLocation, float pWidth, float pHeight)...

    private void DebugDrawRectangle()...
}
```

GENERICS

We use this all the time:

List<Room> rooms = new List<Room>();

Rigidbody rigidBody = GetComponent<Rigidbody>();

Also, remember this from procedural art?

protected T CreateSymbol<T>(string name,Vector3 localPosition=new Vector3(), Quaternion localRotation=new Quaternion(),Transform parent=null) where T : Shape {…}

GENERICS TECHNICAL DETAILS

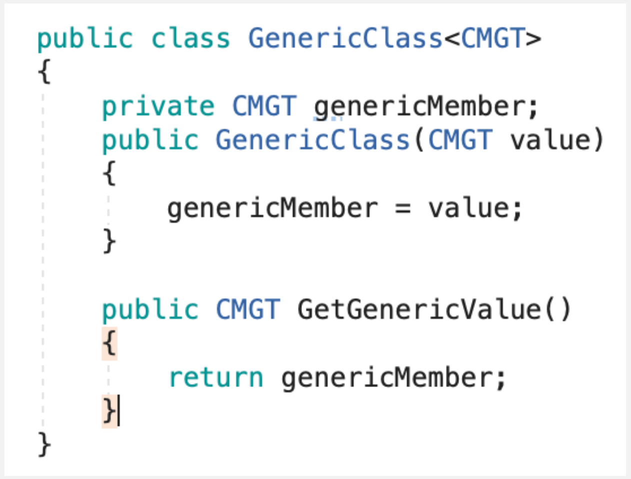
Define classes or methods with a placeholder, then replace the placeholder with actual types at compile time.

Placeholders are put in angle brackets <>.

This is how to use the generic class→

> *GenericClass<string> genericClass*
>
> *= new GenericClass<string>("Best programme ever!");*
>
> *string outputString = genericClass.GetGenericValue();*
>
> *Console.WriteLine(outputString);*

What is the output of the above code?



```
public class GenericClass<CMGT>
{
    private CMGT genericMember;
    public GenericClass(CMGT value)
    {
        genericMember = value;
    }

    public CMGT GetGenericValue()
    {
        return genericMember;
    }
}
```

GENERICS USAGE

Use it to avoid writing a lot of repeated code for different object types.

Instead of:

Knife UpgradeKnife(Knife knife){return knife.Upgrade();}

Sword UpgradeSword(Sword sword){return sword.Upgrade();}

Bow UpgradeBow(Bow bow){return bow.Upgrade();}

Assuming all the classes above inherit Weapon which has a virtual Upgrade method:

Weapon UpgradeWeapon<T>(T weapon) where T:Weapon {T.Upgrade();}

GENERICS USAGE

Now we know why we should do this:

Rigidbody myRigidbody = GetComponent<Rigidbody>();

myRigidbody.mass = 1f;

myRigidbody.velocity = Vector3.zero;

…

Instead of this:

GetComponent<Rigidbody>().mass = 1f;

GetComponent<Rigidbody>().velocity = Vector3.zero;

…

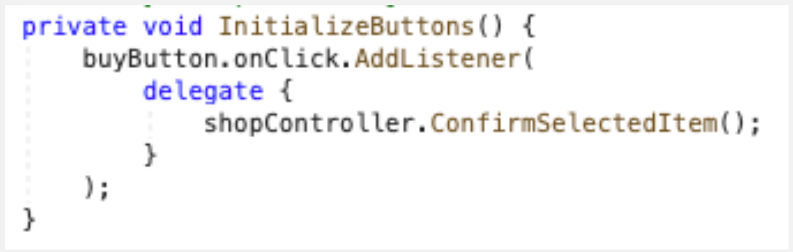
To avoid comparing generic types all the time.

DELEGATE AND EVENT

- A delegate specifies a particular method signature, references to one or more methods can be added to a delegate instance.
- Event is a special kind of delegate that can’t be called outside of the class.
- Delegate and event are fundamental to MVC pattern.
- Let’s just get to the examples to clear things up.

DELEGATE AND EVENT: EXAMPLES

From the assignment's base code, a nice trick to do in Unity:



```
private void InitializeButtons() {
    buyButton.onClick.AddListener(
        delegate {
            shopController.ConfirmSelectedItem();
        }
    );
}
```

This is an anonymous delegate, it will automatically convert the part inside delegate{} into methods by the compiler. This code will call shopController.ConfirmSelectedItem() when the buy button is clicked without having to drag and drop in Unity Editor.

DELEGATE AND EVENT: EXAMPLES

Scenario: we are making a clicker game, a bunch of objects do some stuff whenever the left mouse button is clicked. Do we want to do the following?

```
public class InputManager
{
        …
        void Update()
        {
                if (Input.GetMouseButtonUp(0))
                {
                        player.PlayAnimation();
                        monster.PlayAnimation();
                        archievementManager.AddButtonClickByOne();
                        soundManager.PlayClickSound();
                 }
        }
}
```

Probably not:

- We need to know all the classes that do stuff when the left mouse button is clicked.
- Changes in any of the related classes will likely require us to rewrite this part.
- we almost definitely can’t reuse this in another project.

How about this:

```
public delegate void MouseClickedHandler(); //Define a delegate, doesn't have to be in the class.
public class InputManager
{
    public static MouseClickedHandler OnMouseClicked;
    void Update()
    {
        if (Input.GetMouseButtonUp())
        {
                if(OnMouseClicked != null)
                        OnMouseClicked();
          }
    }
}
public class Player{
    public Player(){
        …
        InputManager.OnMouseClicked+= PlayAnimation;
    }
}
public class SoundManager{
    public SoundManager(){
        …
        InputManager.OnMouseClicked+= PlayClickSound;
    }
}
```

This way our InputManager doesn't care how many classes do stuff after mouse button is clicked, those classes take care of this by themselves.
When OnMouseClicked() is called,
all the methods at the right side of += will be invoked.

DELEGATE AND EVENT: EXAMPLES

But there is a catch:

- If some where in your code you mistakenly use = instead of +=, it will remove all the other methods from the delegate.
- Any class can just call InputManager.OnMouseClicked() and trigger all the methods when mouse button is not clicked.

To fix these, use event:

public static event MouseClickedHandler OnMouseClicked;

Instead of

public static MouseClickedHandler OnMouseClicked;

DELEGATE AND EVENT: EXAMPLES

Assignment:
Review your Unity Game Programming assignment, if you used delegate and event in the project, check if you did it properly. If you didn't use them, see if using them would make anything better.

Think about this question:
What are the differences between using event and using delegate?

TWO LAST THINGS

Unit test and MVC

A basic setup of unit tests is included in the assignment codebase.

Tutorial: https://www.raywenderlich.com/9454-introduction-to-unity-unit-testing
Manual: https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html

More about unit testing in lecture 3.

ONE LAST THING

MVC

Here's a good place to apply Show Don't Tell principle. Let's check the assignment codebase.