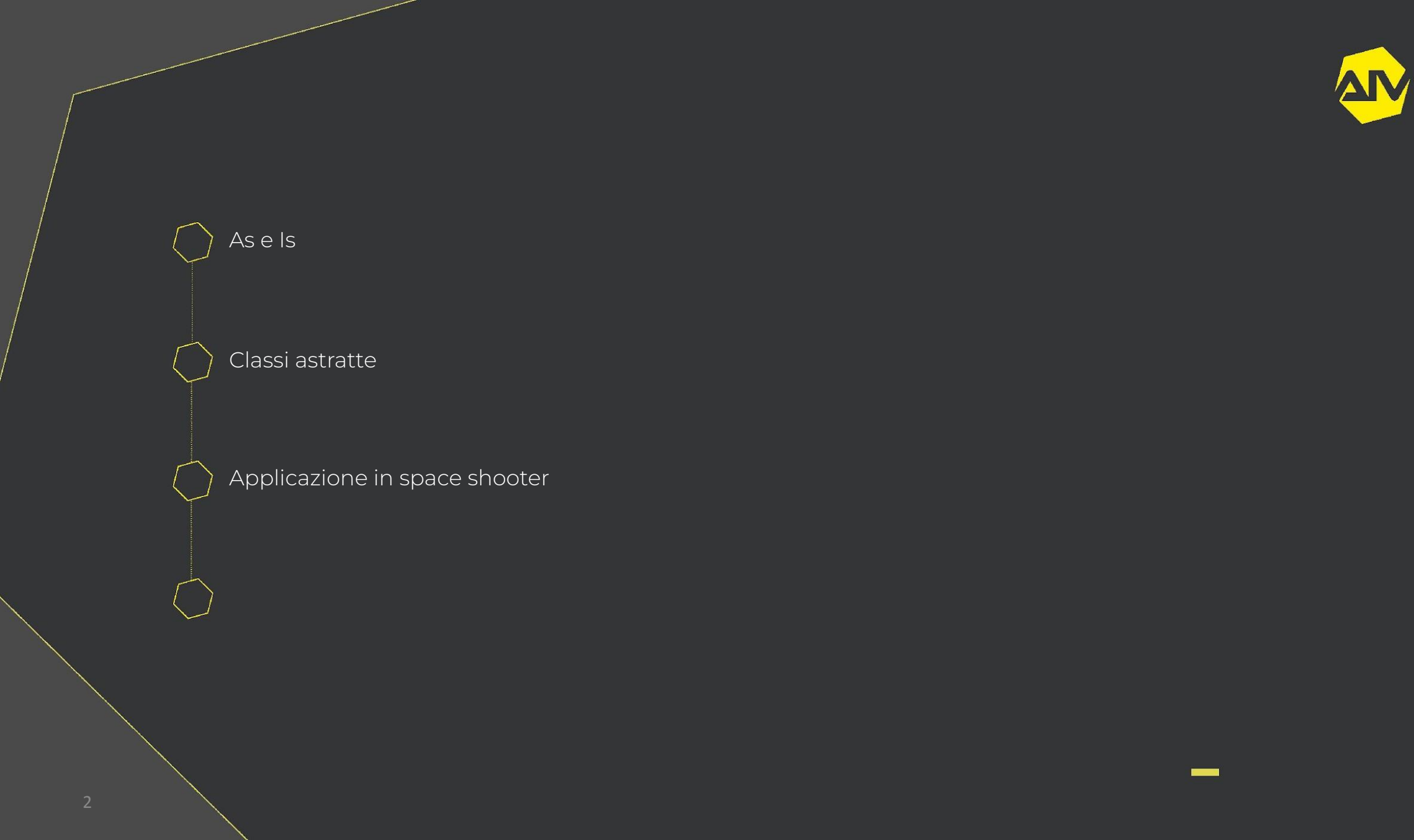
CLASSI ASTRATTE, AS e IS

CORSO DI GAME PROGRAMMING
1° ANNO

Docente **Davide Caio**


As e Is
Classi astratte
Applicazione in space shooter

AS & IS

is:

- Verifica se un oggetto è di un determinato tipo o deriva da esso.
- Restituisce un valore booleano (true o false).

as:

- Tenta di eseguire un cast del tipo.
- Restituisce l'oggetto castato se riesce, oppure null in caso di fallimento

```
Animal myAnimal = new Dog("Fido");

// Uso di is
if (myAnimal is Dog)
{
    Console.WriteLine("L'oggetto è un Dog.");
}

// Uso di as
Dog myDog = myAnimal as Dog;
if (myDog != null)
{
    myDog.Bark();
}
```

Differenze - Quando usare is e as?

is:

- Usato per controllare se un oggetto è di un determinato tipo.
- Tipico per decisioni logiche.

```
if (myAnimal is Dog)
{
    Console.WriteLine("È un cane!");
}
```

as:

- Usato per eseguire un cast sicuro.
- Preferibile quando il cast è necessario per accedere a metodi specifici.

```
Dog myDog = myAnimal as Dog;
if (myDog != null)
{
    myDog.Bark();
}
```

Classi astratte

Problema: voglio raccogliere del codice comune tra entità che hanno un vincolo di appartenenza, ma l'entità che le racchiude non è veramente un'entità del mio gioco (un oggetto che posso istanziare)

Classi astratte! Definizione e caratteristiche:

- Una classe che non può essere istanziata direttamente.
- Serve come modello per le classi derivate.
- Può contenere metodi astratti (senza implementazione).
- Può contenere metodi concreti (con implementazione).

Classi astratte - esempio

```
public abstract class Animal
{
    public string Name { get; set; }

    public Animal(string name)
    {
        Name = name;
    }

    public abstract void Speak();
}

public class Dog : Animal
{
    public Dog(string name) : base(name) { }

    public override void Speak()
    {
        Console.WriteLine($"{Name} dice: Bau!");
    }
}
```

Classi astratte - quando utilizzarle

- Usata per rappresentare un concetto comune con implementazioni parziali.
- Utile quando si vuole centralizzare il codice, creando un'entità generica che non ha senso esistere da sola nel programma (ad esempio, un Animal generico).
- Fornisce una base comune che riduce il rischio di errori, poiché forza chi utilizza le classi derivate a implementare metodi specifici.
- Evita chi utilizzerà il nostro sistema, a creare oggetti che non avrebbero senso di esistere

Ereditarietà - The final slide

ATTENZIONE

Non utilizzate l'ereditarietà in ogni caso in cui si deve definire codice comune.

L'ereditarietà è, come dice il nome stesso, una proprietà di diversi oggetti di ESSERE la stessa cosa.

Se state cercando di usare l'ereditarietà per cercare di raccogliere un comportamento polimorfico (quindi una classe base con classi derivate) del tipo "FANNO" la stessa cosa, NON si usa una classe (e neanche una classe astratta). Quando diversi oggetti FANNO una stessa cosa, si utilizza l'INTERFACCIA. (Che vedremo prossimamente)