Interfaces

Using Interfaces allow you to achieve Abstraction similarly to the way Abstract classes with Abstract methods work.

Note: It is considered good practice to start with the letter “I” at the beginning of an Interface name, as it makes it easier for yourself and others to identify and know that its not a normal class.

Interfaces can contain **Properties** and **Methods** but not fields.

To use an Interface, it must be **implemented** by another class.

```
public class abstract DamagableBase
{
  public abstract void TakeDamage(float damage);
}
public interface IDamagable
{
  float Damage {get;set;}
  void TakeDamage(float damage); // Just like an abstract method, interface methods have no body/definition or concrete implementation
}

public class Crate : MonoBehaviour, IDamagable// Implementing the IDamagable interface
{
  public float Health = 1;

  public float Damage
  {
    get; set;
  }

  public void TakeDamage(float damage) // Concrete Implementation of the TakeDamage method from the Interface
  {
    // ToDo - You will do your logic here.
  }
}
```

▼ Notes

- Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "IAnimal" object in the Program class)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interfaces can contain properties and methods, but not fields/variables
- Interface methods are by default `abstract` and `public`
- An interface cannot contain a constructor (as it cannot be used to create objects)

▼ Why and When to use Interfaces

- To hide certain details about your program
- C# does not support multiple inheritance (a class can only inherit from one base class), however, a class can implement many interfaces.

▼ Usecases for Unity

- Damagable Objects
- Healable Objects
- Interactable Objects
- Item Objects

▼ Examples

```
public interface IDamagable
{
  void TakeDamage(float damage);
}

public interface IHealable
{
  void Heal(float healAmount);
}

public class Hero : MonoBehaviour, IDamagable, IHealable
{
  public void TakeDamage(float damage)
  {
    // Concrete Implementation
  }

  public void Heal(float healAmount)
  {
    // Concrete Implementation
  }
}
```