**TEMASEK POLYTECHNIC**

**SCHOOL OF INFORMATICS & IT**

**DIPLOMA IN IMMERSIVE MEDIA & GAME DEVELOPMENT**

**AY2025/2026 APRIL SEMESTER**

**GADV (CGE2C25)**

**Unity Programming**

**Generics in Unity**

To see the additional comments and resources, make sure you select **All Markup** in the **Review/Tracking** pane



**Overview**

Generics are a powerful feature in Unity’s scripting system. They let us use one function or class to work with different types of data.

You’ve actually used generics already, even if you didn’t know it — for example, when using GetComponent<Rigidbody>() or creating a List<GameObject>.

This worksheet is designed to help you:

* Recognise generics when you see them
* Understand the basic syntax (especially the angle brackets < >)
* Use Unity’s most common generic features confidently

You don’t need to write advanced generic code. The focus is on practical usage of generics in Unity.

By the end of this worksheet, you will be able to:

* Understand what <T> means in Unity C# code
* Use generic Unity functions like:

GetComponent<T>()

TryGetComponent<T>()

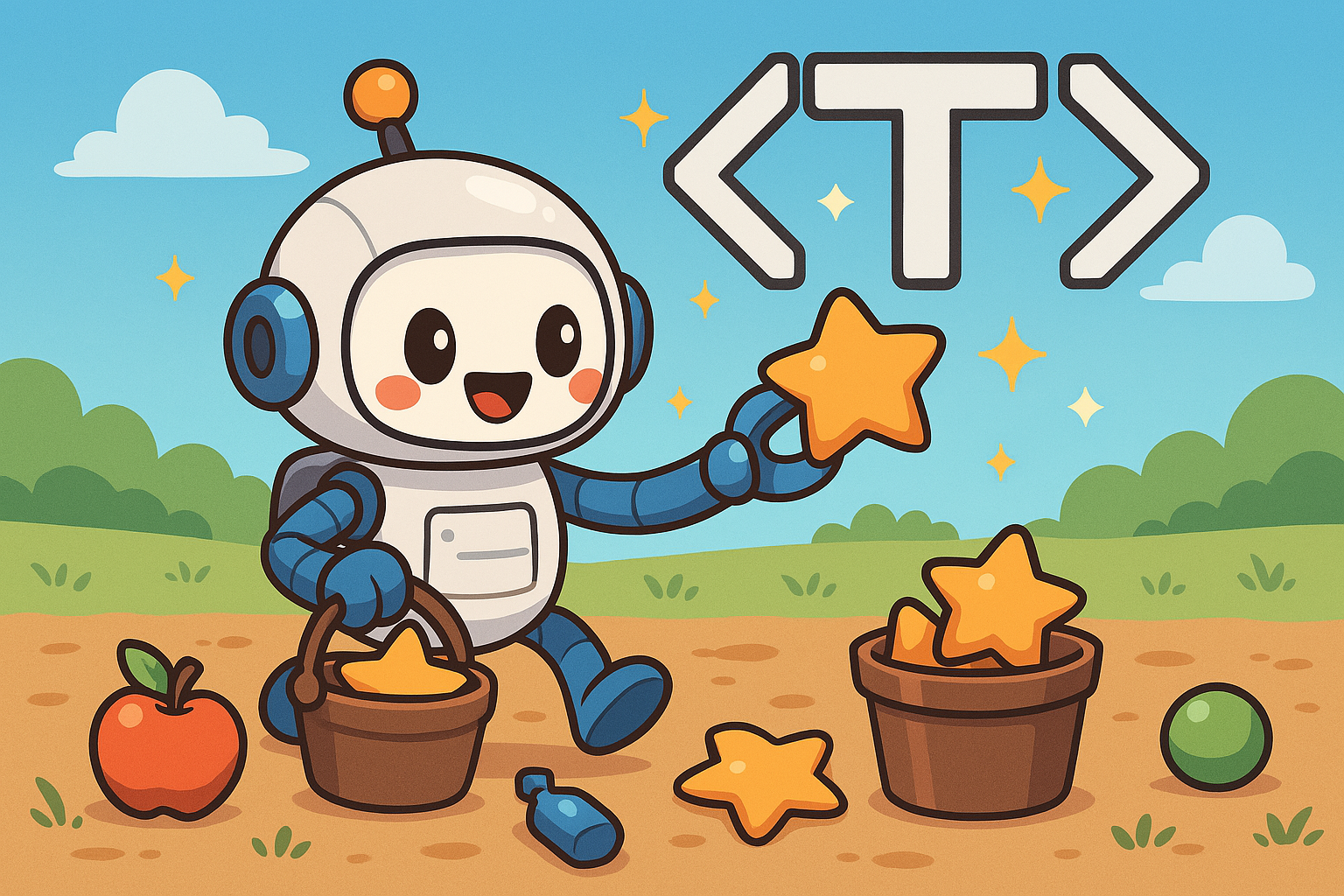
Instantiate<T>()

* Work with generic collections such as:

List<T>

Dictionary<TKey, TValue>

* Read Unity documentation and examples involving generics without being confused or overwhelmed.



In fact, imagine this cheerful robot as a Unity generic function like GetComponent<T>() or List<T>.

It’s a single, reusable robot that can be programmed to search for and collect one type of object at a time — apples, gems, stars — whatever you specify.

The robot doesn’t mix types in its bucket. It only collects the type it’s been told to find (here, the robot is collecting only stars).

Just like Unity’s GetComponent<T>() retrieves one specific component type, this robot knows exactly what it’s looking for — and it ignores the rest.

1. **What Does <T> Mean?**

When you see angle brackets like **<T>** in Unity, it means the function or class can work with *any type*.

The T stands for "Type" — it’s a placeholder for the real type you want to use.

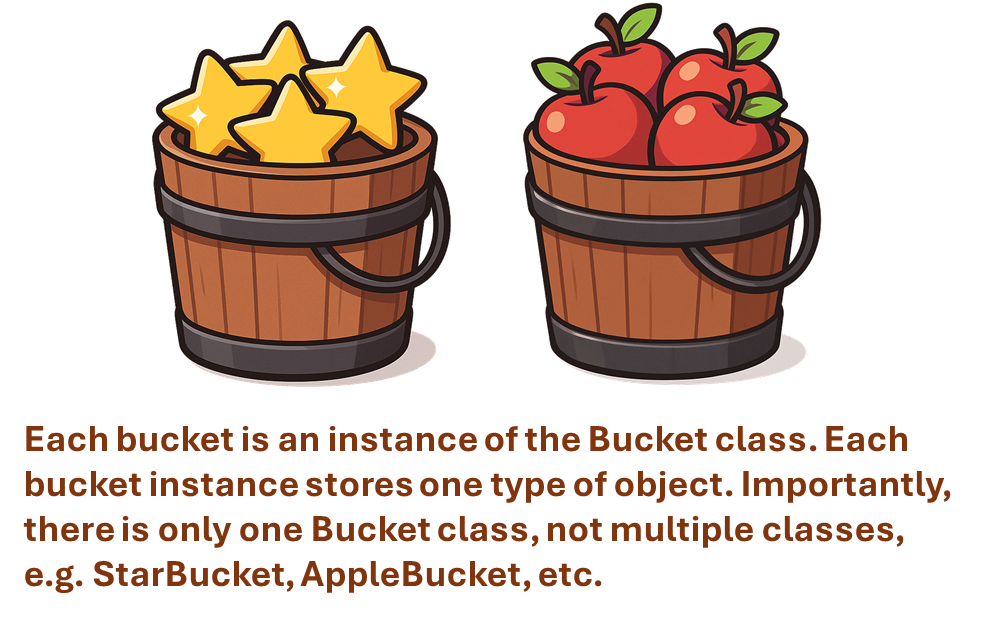
So if you want the robot to collect stars, you'd write <Star>. If apples, you'd use <Apple>. The robot will then only search for and collect that one type, just like a generic function only works with the specific type you assign to <T>.

Here’s a generic **List** used in general C# code:

* List<int> numbers = new List<int>(); // T = int
* List<string> names = new List<string>(); // T = string

There’s one **List** class, but it can store many types.

Think of the robot’s bucket again:



Here’s an example from Unity:

Rigidbody rb = GetComponent<Rigidbody>(); // T = Rigidbody

Unity reads the <T> and understands what component you want.

Here’s a generic datatype with two types:

Dictionary<string, int> scores = new Dictionary<string, int>();

This means that each entry has a **string key** and an **int value**.

You could also define a generic Dictionary like this:

Dictionary<int, SomeClass> scores = new Dictionary<int, SomeClass >();

This means that the Dictionary has an **int key**, and a value of **SomeClass**.

**Exercises**

* Open Unity and create a new scene called **Generics**
* Add a GameObject and rename it GameManager
* Create a new C# script called GenericsGame.cs and attach it to GameManager
* Open the script — you’ll do everything in this file

**Exercise 1**: GetComponent<T>()

Add this code to your script:

using UnityEngine;

public class GenericsGame : MonoBehaviour

{

void Start()

{

Rigidbody rb = GetComponent<Rigidbody>();

Debug.Log("Mass: " + rb.mass);

}

}

What is printed out?

Try changing **Rigidbody** to **BoxCollider**, **MeshRenderer**, or another Unity component type.

**Exercise 2**: TryGetComponent<T>()

This is a safer way to get a component — it won’t cause an error if the component doesn’t exist on the GameObject.

Change your code to:

if (TryGetComponent<AudioSource>(out var audio))

{

Debug.Log("Found AudioSource");

}

else

{

Debug.Log("No AudioSource attached.");

}

Try changing AudioSource to something else. See what happens if the component isn't there.

Also try with Rigidbody, since you know the GameObject does have a Rigidbody component attached to it.

**Exercise 3**: List<T>

As discussed earlier, a List<T> is used to store multiple objects of the same type. In C#, any data structure designed to store multiple items is called a **collection**.

**Note on Using Generic Collections**

To use generic collections like List<T> or Dictionary<TKey, TValue>, make sure you include the correct namespace at the top of your script:

using System.Collections.Generic;

This gives your code access to the classes needed to work with generic data structures.

Change your code to:

using UnityEngine;

using System.Collections.Generic;

public class GenericsGame : MonoBehaviour

{

public List<GameObject> enemies = new List<GameObject>();

void Start()

{

GameObject enemy1 = new GameObject("Creeper");

GameObject enemy2 = new GameObject("Skeleton");

enemies.Add(enemy1);

enemies.Add(enemy2);

Debug.Log("Enemy count: " + enemies.Count);

}

}

This instance of List<T> stores **GameObjects**.

Now change to this:

public List<string> enemieNames = new List<string>();

void Start()

{

string enemy1Name = "Creeper";

string enemy2Name = "Skeleton";

enemieNames.Add(enemy1Name);

enemieNames.Add(enemy2Name);

Debug.Log("Enemy names count: " + enemyNames.Count);

}

This instance of List<T> stores **string** objects. **List<T>** is like a bucket that holds any type of object — GameObjects, strings, int, float, SomeClass, etc.

**Exercise 4**: Dictionary<TKey, TValue>

A Dictionary<TKey, TValue> is a collection that stores pairs of keys and values, where each key is unique and maps to a specific value.

You can think of it like a calendar, where each date (the key) points to a specific event (the value) — every key is unique, and you can quickly look up the value by its key.

Here, we track game scores using two types. Change your code to:

Dictionary<string, int> scores = new Dictionary<string, int>();

void Start()

{

scores["Alex"] = 10;

scores["Jordan"] = 15;

Debug.Log("Alex has " + scores["Alex"] + " points.");

}

Think of it as labels (string) linked to values (int).

**Exercise 5**: Instantiate<T>()

Unity also uses generics when you create objects. Change your code to:

public class GenericsGame : MonoBehaviour

{

public Asteroid AsteroidPrefab;

void Start()

{

SpawnAsteroid(AsteroidPrefab, new Vector3(0,0,0));

}

public GameObject SpawnAsteroid(

GameObject asteroidPrefab, Vector3 position)

{

// The non-generic approach, which returns a GameObject

GameObject clone = Instantiate(

asteroidPrefab, position, Quaternion.identity);

// The generic approach, which returns the actual type

// specified by <T>. Note that the prefab must have a

// component of the specified type, e.g. ateroidPrefab must // have an Asteroid component (script) attached to it.

Asteroid clone = Instantiate<Asteroid>(asteroidPrefab);

Debug.Log("Spawned: " + clone.name);

return clone;

}

}

This example is a little more advanced because it relies on how Unity handles types behind the scenes. If you're unsure about how it works, feel free to ask your tutor!

**Summary**

* A **generic** is like a reusable robot that works with different types — but only one type at a time.
* The <T> in code like GetComponent<T>() or List<T> means “this function or class works with a specific type you choose.”
* You’ve seen how Unity uses generics in functions like GetComponent<T>(), TryGetComponent<T>(), and Instantiate<T>().
* You’ve also worked with **generic collections** like List<T> (to store multiple objects) and Dictionary<TKey, TValue> (to store key–value pairs).
* You now understand how to read and use generic functions and data structures in Unity — even if you’re not writing your own generics yet.

Generics might look a bit strange at first, but you’ve already been using them in your previous wroksheets.

Now you have a basic understanding of what they mean.

It's always better to understand core concepts like this, rather than just copying and pasting code you don’t fully grasp — that way, you’ll become a more confident and capable programmer.

Congratulations for getting this far! 😊

