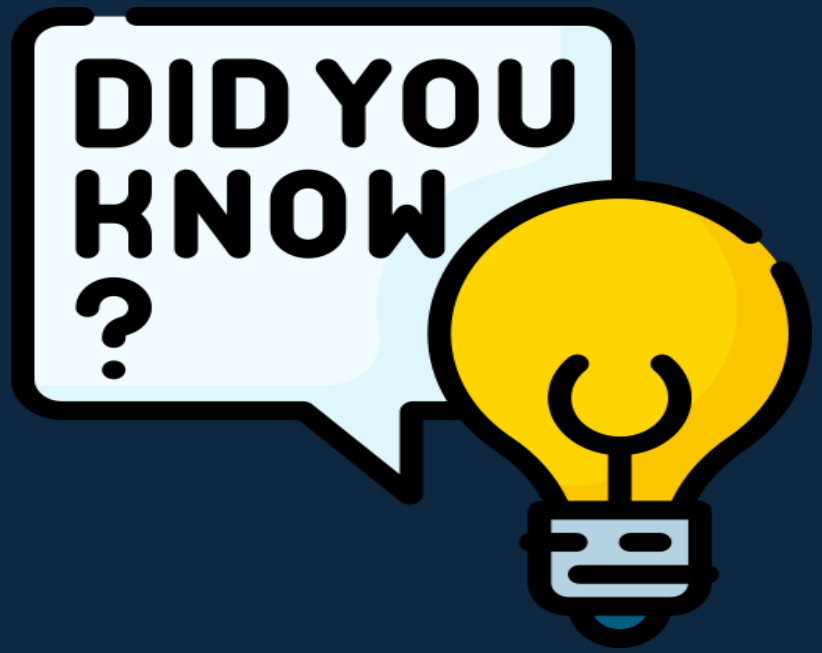
Why do **plugins** implement the **IPlugin interface**, and Why do **custom Workflows** inherit from the **code Activity** abstract class?

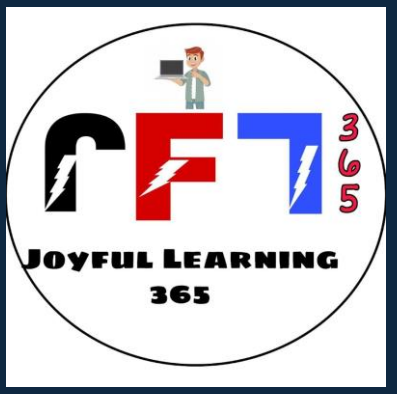


Mohammed Shafiuddin

Microsoft Certified Trainer

Joyful Learning 365

To get the answer, we first gotta know what makes an **interface** different from an **abstract class**.



Mohammed Shafiuddin

Microsoft Certified Trainer

What is an Abstract Class?

Mohammed Shafiuddin

Microsoft Certified Trainer

Imagine you're designing a zoo 🦁🐘🐼, and you have different types of animals: mammals, birds, and reptiles 🐍🐢. Each of these animals shares some common features, like they all need to eat and move around.

So, you create an abstract class called "Animal" 🐾.
This class defines these common features, but you don't create animals directly from it.

Instead, you create subclasses like "Mammal," "Bird," and "Reptile" 📝.
These subclasses inherit from the "Animal" class and add specific traits and behaviors unique to each type of animal 🧬.

What is an Interface?

Mohammed Shafiuddin
Microsoft Certified Trainer

Think of a recipe book 📖 where you want to add new recipes easily.
You create an "Interface" that lists what every recipe should have: ingredients, instructions, etc.

You can't make a recipe directly from this **Interface**, but you use it to create specific recipes like "Cake Recipe" or "Pizza Recipe." Each recipe follows the Interface's guidelines, ensuring they all have the same basic structure while allowing for unique ingredients and steps 🎂🍕.

Enough theory?

Mohammed Shafiuddin

Microsoft Certified Trainer

Mohammed Shafiuddin

Microsoft Certified Trainer

Coming to the main point, the main difference between an interface and an abstract class is that it is required to implement all the methods of an interface, whereas in the case of an abstract class, it's optional which methods you choose to implement

When implementing an interface, it implies that we must implement all of its methods.

Mohammed Shafiuddin

Microsoft Certified Trainer

Now, let's discuss the structure of the IPlugin interface and the CodeActivity abstract class

Mohammed Shafiuddin
Microsoft Certified Trainer

IPlugin Interface	CodeActivity Abstract Class
It only has “one method”, which is Execute, and we implement it in our plugin code.	If you examine the structure of the CodeActivity abstract class closely, you'll find that it has “35 different methods”.

Conclusion

Mohammed Shafiuddin
Microsoft Certified Trainer

As **CodeActivity** possesses **35 methods**, it appears logical to designate it as an abstract class. This grants developers the flexibility to implement the method or methods of their choice or requirement in the class that inherits CodeActivity.

Had CodeActivity been an interface, it would become obligatory to implement all 35 methods in the class implementing the interface, which is impractical.

Hence, **Microsoft opted for CodeActivity as an abstract class, while IPlugin remains an interface**.

https://www.youtube.com/@joyfullearning3659

YouTube

Support us by subscribing to our YouTube channel.

Mohammed Shafiuddin

Microsoft Certified Trainer

Joyful Learning 365