Abstract Classes vs. Interfaces

Interfaces:

- Specifies what. A class. Must do and not how
- "Blueprint"of the class

Abstract classes:

- Classes that provide methods that can be defined **(concrete methods)** or that will eventually need to be implemented **(abstract methods)**
- **Concrete methods:** defined methods that are already implemented in abstract classes
- **Abstract methods:** methods that need to be implemented
    - Has "abstract" keyword in method header in abstract classes

Key differences & similarities

	Interfaces	Abstract Classes
Methods	- Mainly abstract methods o Unless keyword "default"	- Both abstract and concrete methods
Variables	- Only static, final variables	- Can contain static, non-static, final, and non-final variables
Implementation	- Can't provide implementation to methods unless using "default" keywords	- Can provide implementation to an interface o And not provide implementations to all of interfaces' methods
Inheritance	- "implements" keyword - Classes can implement multiple interfaces	- "extends" keyword - Classes can extend **ONLY ONE** abstract class
Access	- Public by default	- Private, protected, etc. (all different access levels are fine)

How to decide when to use which?

- If you want to have a set of different objects that share certain behaviors (e.g., objects that can be compared – use comparable interface), use **interfaces**.
- If you have a general category of objects (e.g., shapes), use **abstract classes**.