**Takeaways**

* Practical scenarios
* The Java Virtual Machine is an abstraction. Your code can use the exact same java methods irrespective of whether your code will run on Linux or Windows or any other OS. Underneath, a lot of things will change depending on the OS, which the user doesn’t need to know.
* You can use a simple API call to fetch weather data or stock prices. These API abstract out the internal complexities of how it is implemented.
* Software libraries are good examples of abstraction. They tell you the methods you can use without telling you how they are implemented.
* Summary of Abstraction
* It separates the method signature from its implementation.
* Closely related classes can separate out common fields/methods into a different class and reuse that class.
* **Why** is it useful? - To the implementers of the class, it is useful since it provides them with the flexibility to change things as long as the method signatures don’t change. For the users, it keeps them from getting exposed to anything other than the method signatures they need to know, reducing complexity.
* **How** to apply? - Use a smaller set of public methods
* What is the drawback if we don’t use abstraction? - Low maintainability because responsibilities are not clearly differentiated. Higher Complexity with larger code bases because many objects interact with others and it becomes difficult to add functionality without impacting others.
* Language specific notes
* In Java, interface classes provides total abstraction and abstract classes provide partial abstraction
* In C++, you can use either header files to achieve abstraction or private methods to hide implementation details.
* Python interface can be applied using the the Abstract Base Class library
* General Abstraction notes
* Hides underlying complexity of data from users
* Helps avoid repetitive code
* Presents only the signature of internal functionality
* Gives flexibility to programmers to change the implementation of the abstract behaviour
* Difference between Encapsulation and Abstraction
* Abstraction is a natural extension of Encapsulation. Encapsulation recommends that you create entities/objects which contain closely related fields and functionality/methods. Abstraction further recommends that the entities expose only their method signatures to external users who shouldn’t need to worry about the underlying implementation.
* Encapsulation - hide data. Abstraction - hide complexity.

**References**

1. [**Interface vs. Abstract Class in Java**](https://howtodoinjava.com/java/oops/exploring-interfaces-and-abstract-classes-in-java/)
2. [**Abstraction vs Encapsulation**](https://www.quora.com/What-is-the-difference-between-abstraction-and-encapsulation-2)
3. [**OOPS Concepts Java - Abstraction**](https://raygun.com/blog/oop-concepts-java/#abstraction)

**Curious Cats**

* What is the difference between a regular class, an abstract class and an interface class in Java?
* How can you spot if a code base has used Abstraction in its design?

**Newfound Superpowers**

* You are now cruising along the *Abstraction Understood* highway, ready to design larger pieces of software more efficiently.

**Now you can**

* Think about the bigger picture with situations where abstraction adds value.
* Crack that interview question with a real example
* Answer these interview questions
* What is Abstraction?
* What is the difference between Encapsulation and Abstraction?

**Java Abstraction — Key Takeaways**

**Definition**

* **Abstraction**: Hiding internal implementation details and exposing only the required method signatures to the user.
* Allows focus on *what* a method does, not *how* it does it.

**Practical Scenarios**

* **Java Virtual Machine**: Same Java code runs on different OS without worrying about OS-specific differences.
* **APIs**: Example—fetching weather or stock prices via a single method call, hiding complex backend logic.
* **Software Libraries**: Provide methods you can call without revealing internal code.

**Summary**

* **Separates**: Method signature from implementation.
* **Organizes**: Common fields/methods into a base class for reuse.
* **Benefits**:
  + Flexibility for implementers to change code without affecting users.
  + Reduced complexity for users.
* **Drawbacks without abstraction**:
  + Low maintainability.
  + Higher complexity in large codebases.
  + Difficulty in adding new features without breaking existing code.
* **How to Apply**: Use a small, focused set of public methods.

**Language-Specific Notes**

* **Java**:
  + interface → total abstraction.
  + abstract class → partial abstraction.
* **C++**: Header files or private methods hide implementation.
* **Python**: Abstract Base Class (abc module) implements abstraction.

**General Abstraction Benefits**

* Hides complexity.
* Avoids repetitive code.
* Shows only method signatures.
* Allows implementation changes without breaking usage.

**Abstraction vs Encapsulation**

| **Aspect** | **Abstraction** | **Encapsulation** |
| --- | --- | --- |
| **Focus** | Hides *complexity* | Hides *data* |
| **Purpose** | Show *what* to do, hide *how* | Protect data from unauthorized access |
| **Relation** | Built on top of encapsulation | Foundation for abstraction |

**Java Class Types**

| **Type** | **Features** | **Example Use** |
| --- | --- | --- |
| **Regular Class** | Fully implemented methods. | Utility classes, data classes. |
| **Abstract Class** | Partial abstraction (can have abstract + concrete methods). | Base class for related subclasses. |
| **Interface** | Total abstraction (only method signatures + constants). | Multiple unrelated classes implementing common behavior. |

**Spotting Abstraction in a Codebase**

* Use of interface or abstract keywords.
* Public methods with hidden/private implementations.
* API calls where internal processing is not exposed.

**Interview-Ready Answers**

**Q1: What is Abstraction?**

Abstraction is the OOP concept of hiding internal implementation details and exposing only essential functionalities through method signatures. It helps reduce complexity, improve maintainability, and allows changes in implementation without affecting users.

**Q2: Difference between Abstraction and Encapsulation?**

Abstraction hides *complexity*—focusing on what to do, not how to do it.  
Encapsulation hides *data*—restricting direct access to object fields by using access modifiers and getter/setter methods.