Abstraction in Java

1. Delivering only essential information to the outer world while masking the background details.

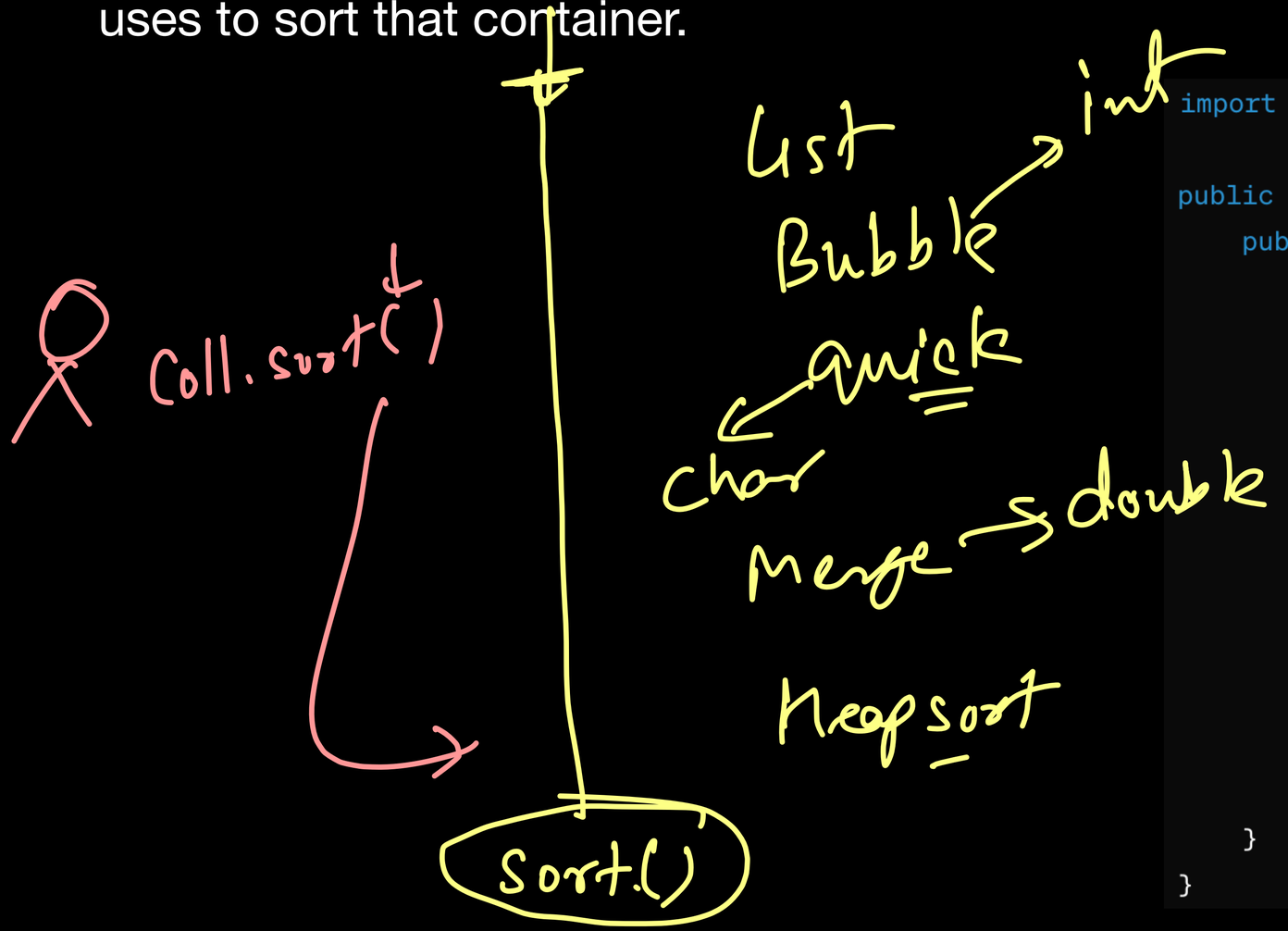
2. It is a design and programming method that separates the interface from the implementation.

3. Real life e.g., various functionalities of AirPods but don’t know the actual implementation/ working

    1. To drive a car, one only needs to know the driving process and not the mechanics of the car engine

Abstraction in Collections

1. E.g., Sort(), for example, is used to sort an array, a list, or a collection of items, and we know that if we give a container to sort, it will sort it, but we don't know which sorting algorithm it uses to sort that container.




```java
import java.util.*;

public class SortExample {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<>();
        numbers.add(15);
        numbers.add(9);
        numbers.add(20);
        numbers.add(3);

        Collections.sort(numbers);

        System.out.println("Sorted Numbers:");
        for (int number : numbers) {
            System.out.println(number);
        }
    }
}
```

Abstraction using Classes

1. Grouping data members and member functions into classes using access specifiers.

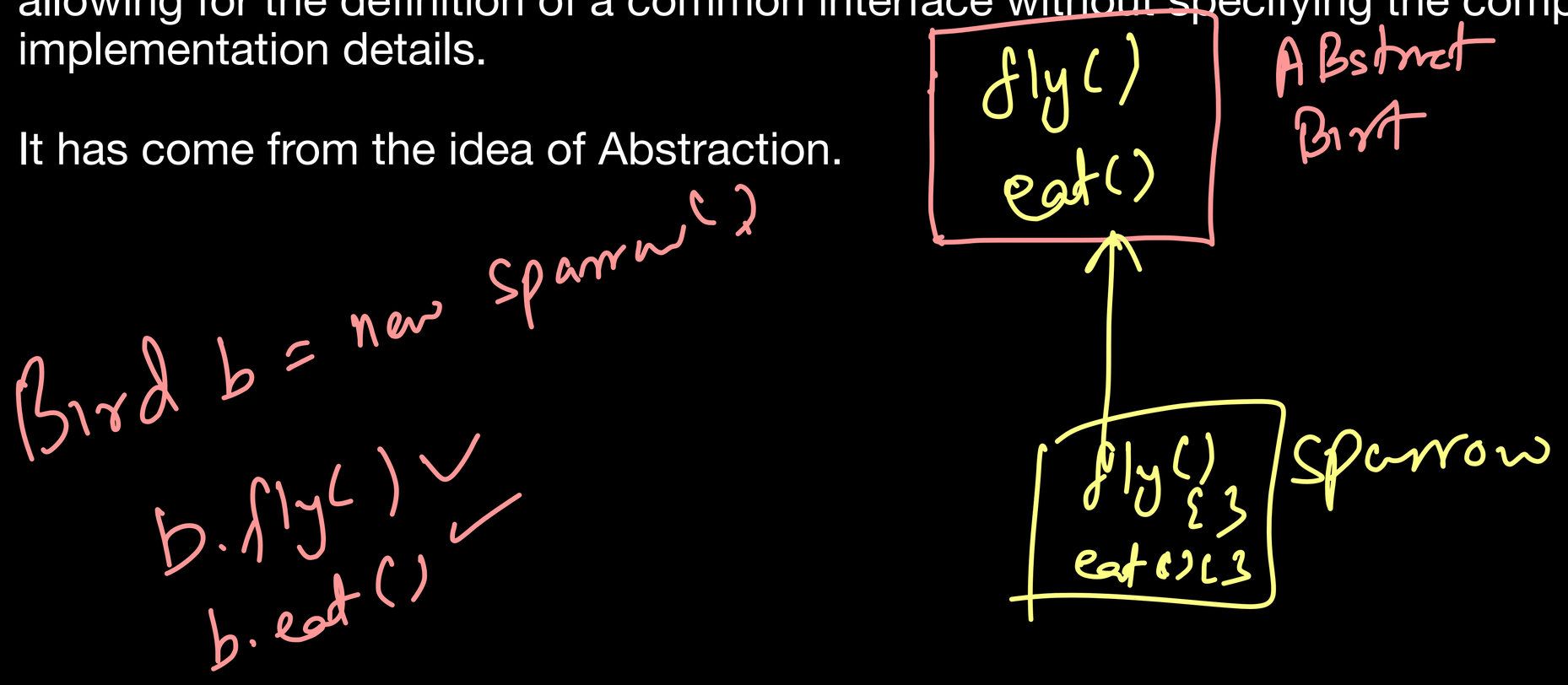
2. A class can choose which data members are visible to the outside world and which are hidden.

What is Abstract Class?

1. An abstract class in Java is a class that cannot be instantiated on its own and is designed to be subclassed. It can serve as a superclass for other classes that share a common structure or behaviour.

2. An abstract class in Java delivers abstraction by serving as a template for its subclasses, allowing for the definition of a common interface without specifying the complete implementation details.

3. It has come from the idea of Abstraction.



Design Strategy

1. Abstraction divides code into two categories: interface and implementation. So, when creating your component, keep the interface separate from the implementation so that if the underlying implementation changes, the interface stays the same.

2. In this instance, any program that uses these interfaces would remain unaffected and would require recompilation with the most recent implementation.

3. Makes code modular and maintainable.

Abstract Class vs Interface

Feature	Abstract Class	Interface
Instantiation	Cannot be instantiated directly.	Cannot be instantiated.
Method Types	Can have both abstract and concrete methods.	Prior to Java 8, could only contain abstract methods. From Java 8 onwards, can also contain default and static methods with implementations.
Constructor	Can have constructors, which are called when a subclass is instantiated.	Cannot have constructors.
Access Modifiers	Methods can have any access modifiers (public, protected, private).	All methods are implicitly public. From Java 9, interfaces can also contain private methods but they must be used within the interface itself.
Inheritance	A class can extend only one abstract class due to single inheritance.	A class can implement multiple interfaces.
Fields and Constants	Can have fields and constants with any access modifier, and they can be non-static and non-final.	All fields are implicitly public, static, and final (constants).
Use Case	Ideal for classes that share a common base of properties and methods.	Ideal for defining a contract that different classes can implement, supporting multiple inheritance of type.

Advantages of Abstraction

1. Reduces Complexity
2. Increase Security
3. Increase Reusability
4. Avoid Code Duplication
5. Loose Coupling