Lec 45 JAVA 8 P1:

Java 8 introduced major features that made Java programming easier, more efficient, and closer to modern programming styles like functional programming. The most important changes include new ways to write code inside interfaces (default and static methods), support for functional programming, and big internal improvements that make programs run faster and require less code.

Java 8: Why It Matters

Java 8 is still widely used in the industry, despite newer versions, because it brought the most significant and practical changes to the language, such as functional programming support and reduced boilerplate code.

Evolution of Java and Functional Programming

Before Java 8, Java was purely object-oriented, but Java 8 added features to support functional programming, making it possible to write code in a more concise and flexible way, similar to languages like Python.

Internal Enhancements in Java 8 Collections

Java 8 improved the efficiency of data structures like HashSet and HashMap by changing how they handle data collisions—from linked lists to trees—making them faster and more reliable without changing how programmers use them.

Funny, I recently learned about the internal workings of a HashSet, and in Java 8+ when a collision occurs, the HashMap stores the elements in a linked list. If the number of elements present at a given bucket exceeds more than 8 elements (due to collision) then it checks the total number of elements stored in the HashMap. If the total number of elements in the HashMap exceeds 64 then your linked list gets converted into a balanced tree.

So, for a linked list basic operations like insert, remove, and contains can give you a linear time complexity.

However, since Java 8+, the linked list has been converted into a balanced tree, which gives logarithmic time complexity for basic operations. This is much better than the earlier Java versions.

Backward Compatibility and Major Changes

Java 8 introduced changes that are compatible with older code, so even if you use Java 17 or 21, most core features and syntax still rely on Java 8. This backward

compatibility allows companies to upgrade Java versions without rewriting existing code.

Key Java 8 Features Overview

The most important Java 8 features for interviews and practical use are functional interfaces, lambda expressions, and the Stream API. Other useful additions include method references, default and static methods in interfaces, the Optional class, and enhancements in collections and file I/O.

Default and Static Methods in Interfaces

Java 8 allows default and static methods inside interfaces. Default methods let you provide a method body in an interface, while static methods allow common code that is not tied to any object. These features help avoid breaking existing code when interfaces are updated.

- Like if we want to add AI feature in Hummer and BMW class and not in Alto and Nano class (Assuming they implements same interface named Vehicle).
- Now if we add abstract method AI() in our Vehicle interface then it will be necessary for alto and nano class to implement it.

- Instead we can use default method in interface Vehicle which will be need not to be implemented by all its children (SELECTIVE).

Default Methods: Purpose and Backward Compatibility

Default methods in interfaces let you add new functionality without forcing all implementing classes to update. This solves the problem where adding a new method to an interface would break all existing implementations, ensuring older code continues to work.

Understanding Backward Compatibility with Default Methods

By using default methods, interfaces can be upgraded (e.g., adding new features) without affecting legacy classes. This enables new classes to use new features, while old classes remain functional without modification.

Static Methods in Interfaces: Common Code Sharing

Static methods in interfaces allow you to write code that is common to all implementing classes, like policies

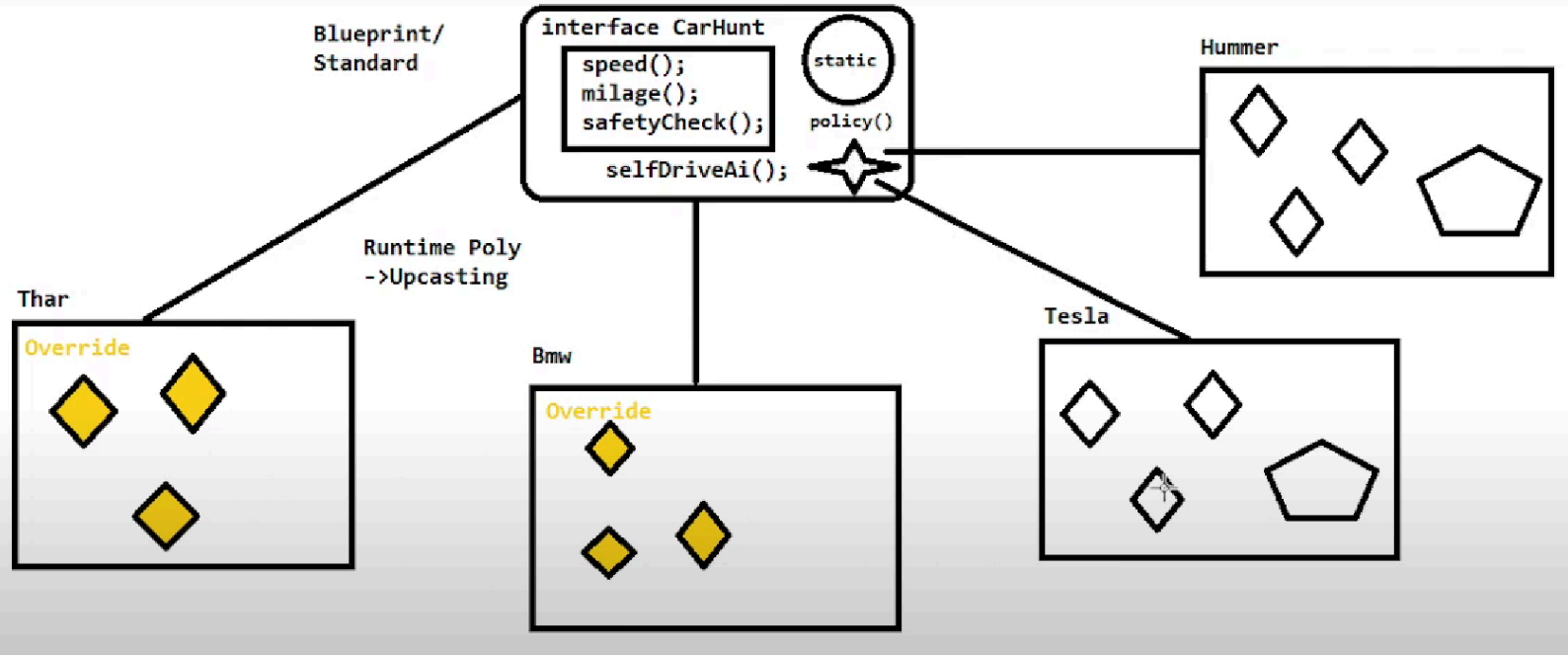
or utility functions, directly inside the interface instead of external helper classes.

The Role and Benefit of Default Methods

Default methods allow you to provide a standard implementation in an interface, which can be used or overridden by implementing classes as needed. This helps when only some classes need new functionality, while others can ignore it.

Practical Example: Upgrading Interfaces with Default Methods

When an interface is upgraded with a new default method, only new classes that need the feature override it, while existing classes remain unaffected. This avoids the need for multiple interfaces or repeated code, keeping the codebase clean and manageable.



Summary: Multiple Inheritance and Interface Improvements

Java 8 interfaces now support multiple inheritance (via multiple interfaces), abstract methods, and default methods, making them more flexible than abstract classes. This allows sharing both empty and implemented methods across many classes.

Next Steps: Functional Interfaces and Lambda Expressions

The upcoming topics include functional interfaces and lambda expressions, which are central to Java 8's approach to functional programming and will be covered after understanding inner classes.