CODING SHUTTLE

OOPS - 4

In This Lecture

1. The abstract keyword
2. Abstraction
3. Java Interfaces
4. Inner class & Nested static classes
5. Anonymous Classes
6. Functional Interfaces
7. Lambda expressions

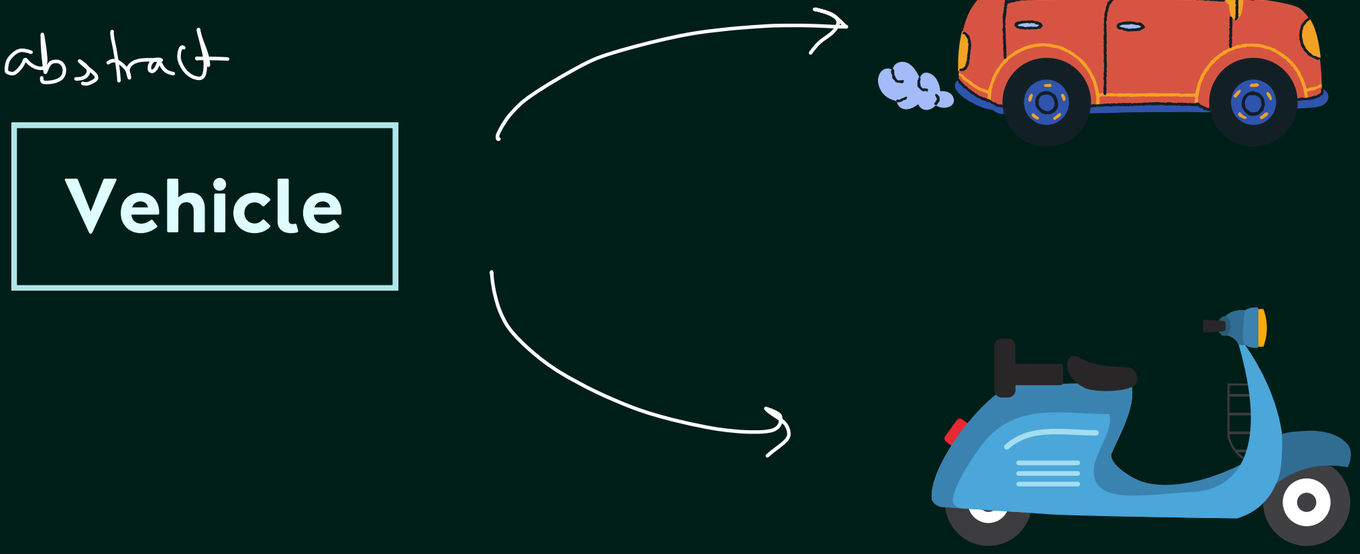
Java Abstract Class

The abstract class in Java cannot be instantiated (we cannot create objects of abstract classes). We use the abstract keyword to declare an abstract class.

- An abstract class can have both the regular methods and abstract methods.
- A method that doesn't have its body is known as an abstract method.
- Though abstract classes cannot be instantiated, we can create subclasses from it. We can then access members of the abstract class using the object of the subclass.
- If the abstract class includes any abstract method, then all the child classes inherited from the abstract superclass must provide the implementation of the abstract method.

Java Abstraction

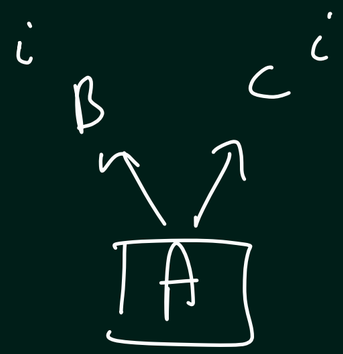
Abstraction is an important concept of object-oriented programming that allows us to hide unnecessary details and only show the needed information.
This allows us to manage complexity by omitting or hiding details with a simpler, higher-level idea.



Java Interfaces

An interface is a fully abstract class. It includes a group of abstract methods (methods without a body).
We use the interface keyword to create an interface in Java.

Like abstract classes, we cannot create objects of interfaces.
To use an interface, other classes must implement it. We use the implements keyword to implement an interface.

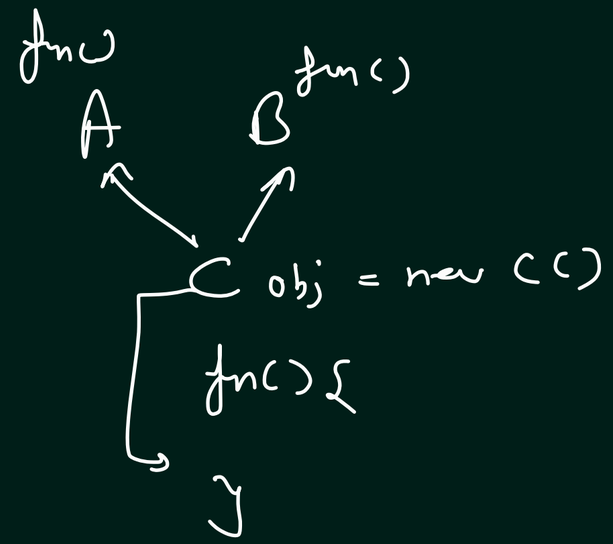



```
interface Language {
  public void getType();

  public void getVersion();
}
```

Advantages of Java Interfaces

- Similar to abstract classes, interfaces help us to achieve abstraction in Java.
- Interfaces are also used to achieve multiple inheritance in Java.
- Note: All the methods inside an interface are implicitly public and all fields are implicitly public static final.

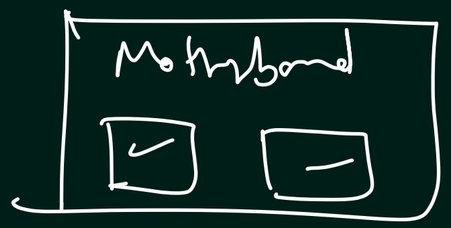
```
interface Line {
…
}

interface Polygon {
…
}

class Rectangle implements Line, Polygon {
…
}
```



Inner classes and Nested Static Class in Java

A non-static nested class is a class within another class. It has access to members of the enclosing class (outer class). It is commonly known as inner class.
Since the inner class exists within the outer class, you must instantiate the outer class first, in order to instantiate the inner class.

Unlike inner class, a static nested class cannot access the member variables of the outer class. It is because the static nested class doesn't require you to create an instance of the outer class.



Using the nested class makes your code more readable and provide better encapsulation.

Anonymous Classes in Java

In Java, a class can contain another class known as nested class. It's possible to create a nested class without giving any name.
A nested class that doesn't have any name is known as an anonymous class.

Anonymous classes usually extend subclasses or implement interfaces.
Here, Type can be

1. a superclass that an anonymous class extends
2. an interface that an anonymous class implements

Functional Interfaces

An Interface that contains exactly one abstract method is known as a functional interface.

Functional Interfaces introduced in Java 8 allow us to use a lambda expression to initiate the interface's method and avoid using lengthy codes for the anonymous class implementation.

```
// interface
@FunctionalInterface
interface Sample{
    // abstract method
    int calculate(int val);
}
```

Lambda Expression

```
(int x) -> x+1                    // Single declared-type argument
(int x) -> { return x+1; }        // same as above
(x) -> x+1                        // Single inferred-type argument, same as below
x -> x+1                          // Parenthesis optional for single inferred-type case

(String s) -> s.length()          // Single declared-type argument
(Thread t) -> { t.start(); }      // Single declared-type argument
s -> s.length()                   // Single inferred-type argument
t -> { t.start(); }               // Single inferred-type argument

(int x, int y) -> x+y             // Multiple declared-type parameters
(x,y) -> x+y                      // Multiple inferred-type parameters
```