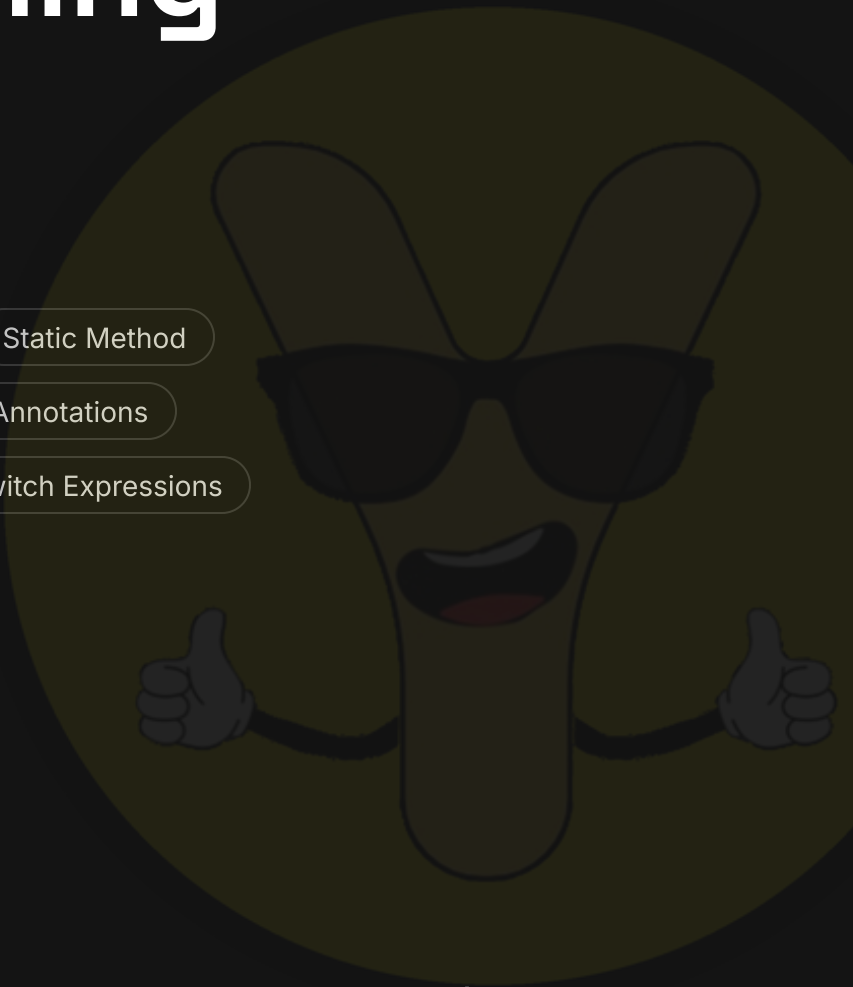
AKTU · 2ND YEAR · SEMESTER 4 · BCS403

Object Oriented Programming with Java

Unit 3 — Java New Features

- Functional Interfaces
- Lambda Expression
- Method References
- Stream API
- Default Methods
- Static Method
- Base64 Encode and Decode
- ForEach Method
- Try-with-resources
- Type Annotations
- Repeating Annotations
- Java Module System
- Diamond Syntax with Inner Anonymous Class
- Local Variable Type Inference
- Switch Expressions
- Yield Keyword
- Text Blocks
- Records
- Sealed Classes



DECODED BY

Chirag Jain

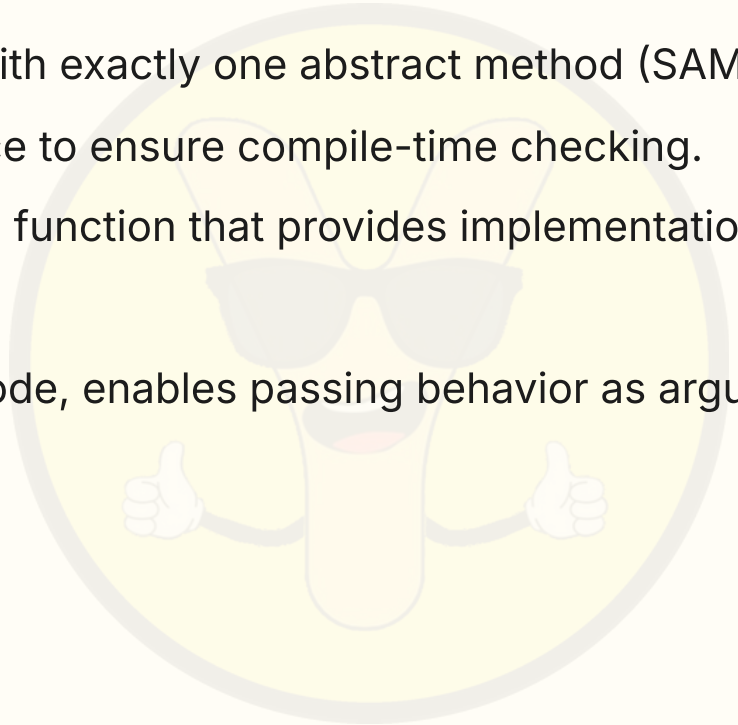
Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

theycotes.com

Functional Interfaces and Lambda Expressions

Java 8 introduced functional programming features to simplify code.

- Functional Interface: An interface with exactly one abstract method (SAM).
- Annotated with @FunctionalInterface to ensure compile-time checking.
- Lambda Expression: An anonymous function that provides implementation for a functional interface.
- Syntax: (parameters) → { body };
- Advantages: Reduces boilerplate code, enables passing behavior as arguments.



Lambda Expression Example

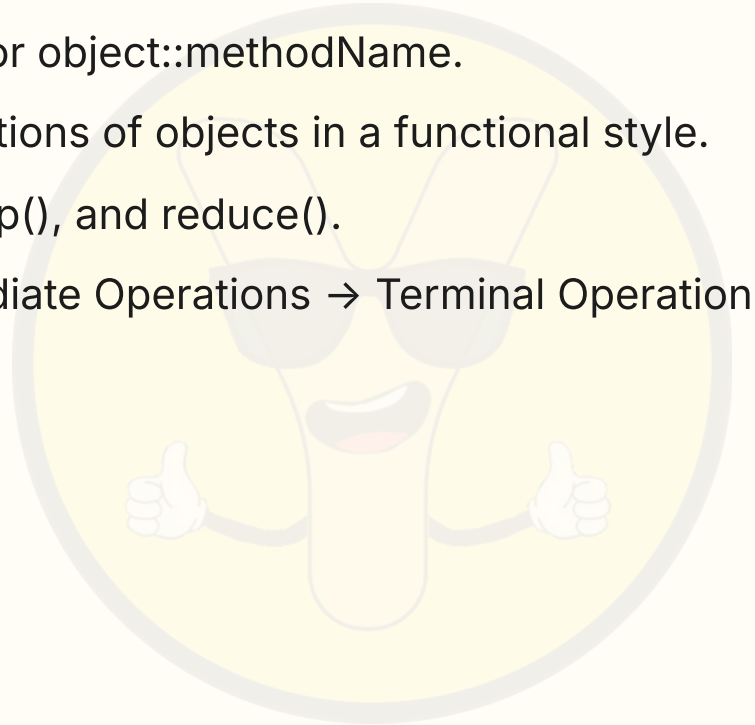
```
interface MathOp { int operate(int a, int b); }

public class Main {
  public static void main(String[] args) {
    MathOp add = (a, b) → a + b;
    System.out.println(add.operate(10, 20));
  }
}
```



- The lambda expression provides the logic for the 'operate' method.
- Type inference allows omitting parameter types.

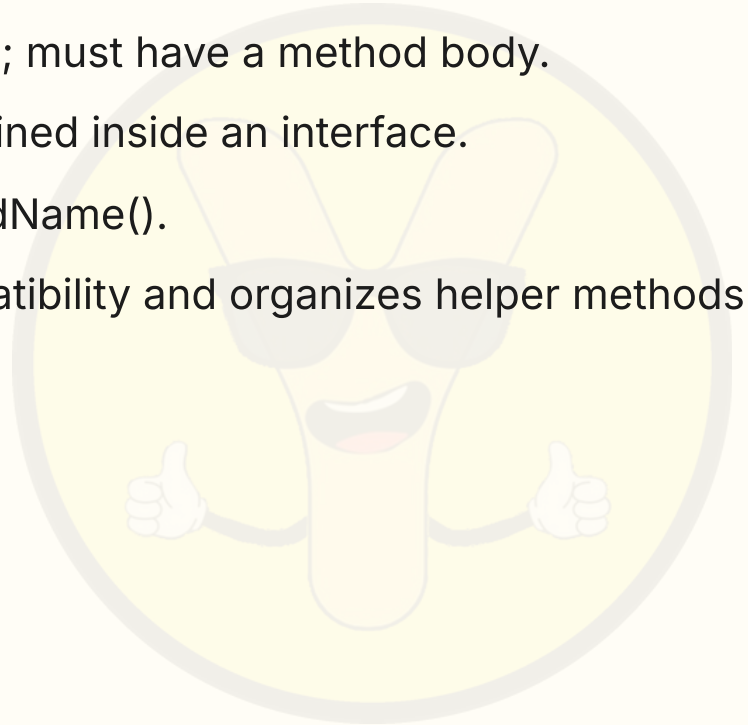
Method References and Stream API

- Method Reference: Shorthand notation for lambda expressions calling existing methods.
- Syntax: ClassName::methodName or object::methodName.
- Stream API: Used to process collections of objects in a functional style.
- Supports operations like filter(), map(), and reduce().
- Stream pipeline: Source → Intermediate Operations → Terminal Operation.



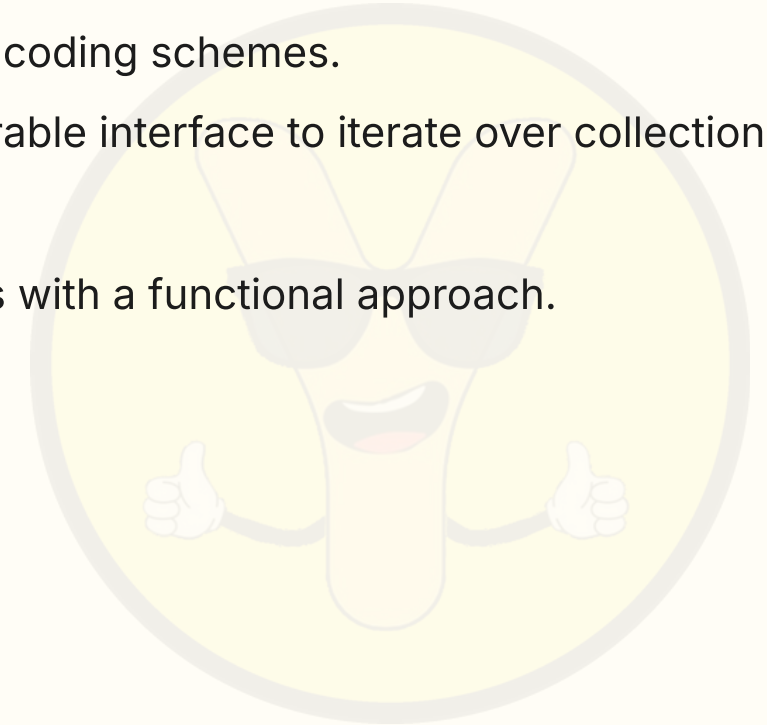
Default and Static Methods in Interfaces

- Default Methods: Allow adding new methods to interfaces without breaking implementing classes.
- Defined using the 'default' keyword; must have a method body.
- Static Methods: Utility methods defined inside an interface.
- Called using InterfaceName.methodName().
- Purpose: Provides backward compatibility and organizes helper methods.



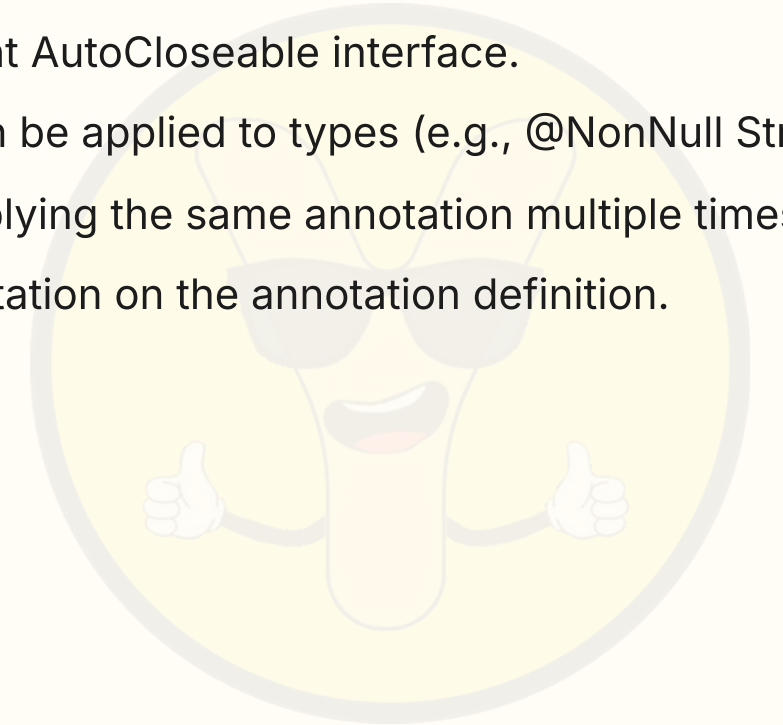
Base64 Encoding and ForEach Method

- Base64: Utility class (java.util.Base64) for encoding and decoding data.
- Supports Basic, URL, and MIME encoding schemes.
- ForEach Method: Introduced in Iterable interface to iterate over collections.
- Syntax: collection.forEach(action);
- Replaces traditional for-each loops with a functional approach.



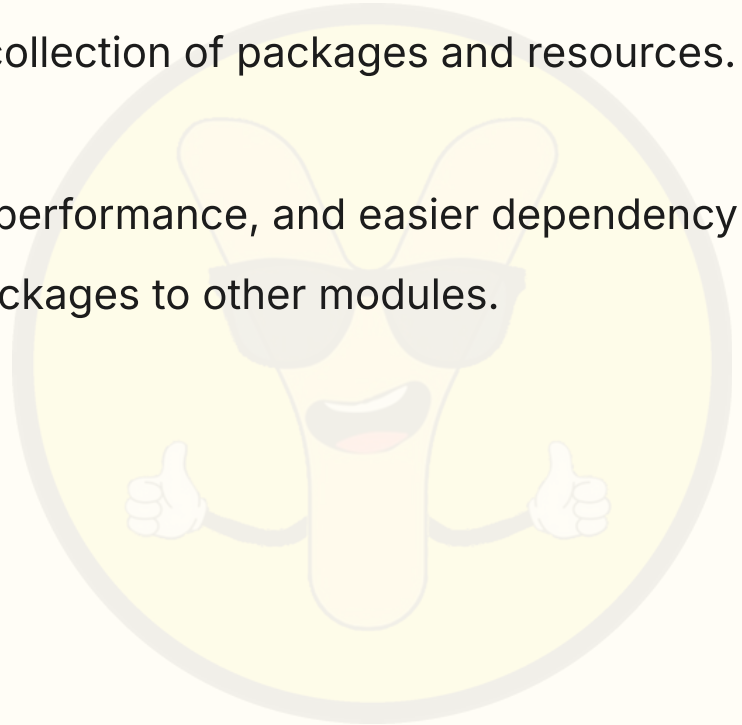
Try-with-resources and Annotations

- Try-with-resources: Ensures resources (like files) are closed automatically.
- Requires the resource to implement AutoCloseable interface.
- Type Annotations: Annotations can be applied to types (e.g., @NonNull String s).
- Repeating Annotations: Allows applying the same annotation multiple times to a declaration.
- Requires @Repeatable meta-annotation on the annotation definition.



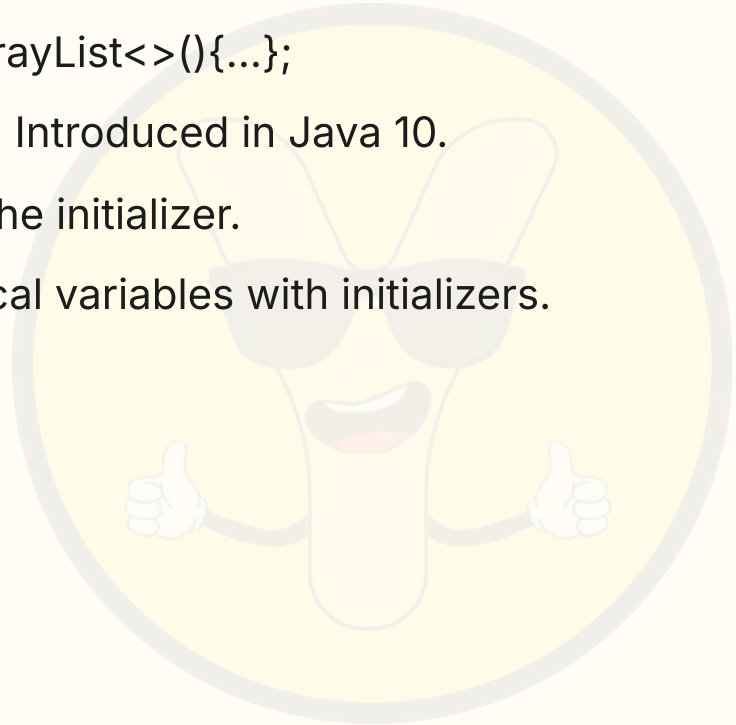
Java Module System (Project Jigsaw)

- Introduced in Java 9 to modularize the JDK and application code.
- Module: A named, self-describing collection of packages and resources.
- Defined using module-info.java file.
- Benefits: Improved security, better performance, and easier dependency management.
- Encapsulation: Explicitly exports packages to other modules.



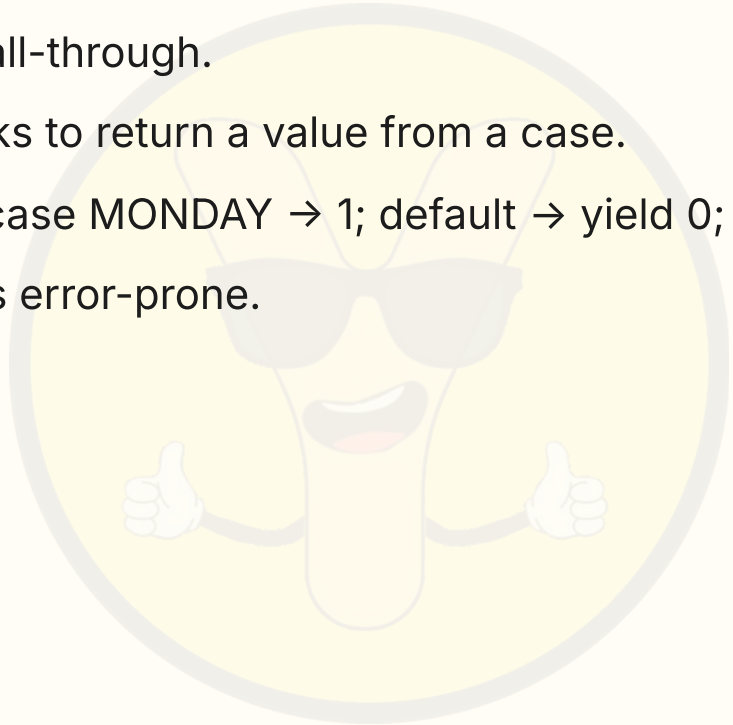
Diamond Syntax and Local Variable Inference

- Diamond Syntax: Allows omitting type arguments in anonymous inner class constructors.
- Example: List<String> list = new ArrayList<>(){...};
- Local Variable Type Inference (var): Introduced in Java 10.
- Compiler infers the type based on the initializer.
- Constraint: Can only be used for local variables with initializers.



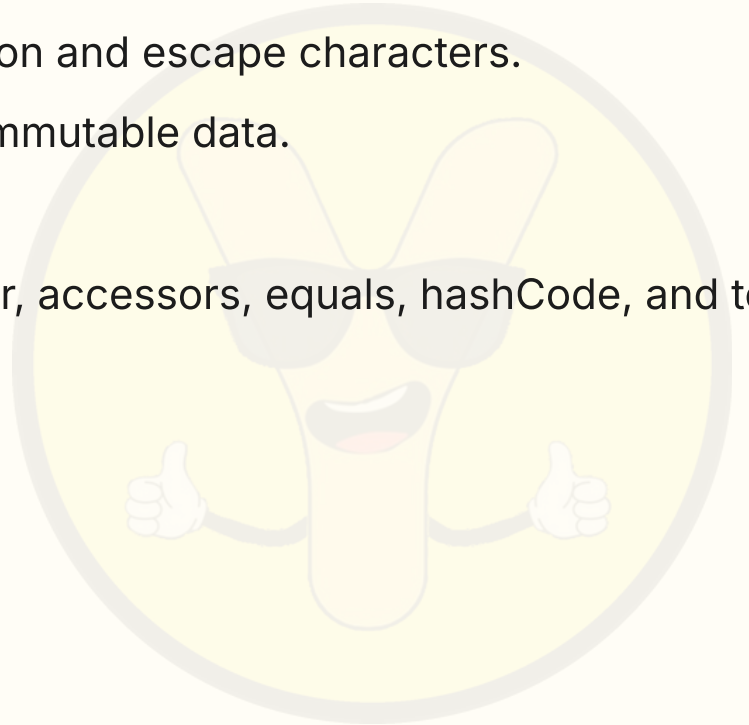
Switch Expressions and Yield

- Switch Expressions: Enhanced switch that returns a value.
- Uses arrow syntax (→) to prevent fall-through.
- Yield Keyword: Used in switch blocks to return a value from a case.
- Example: int result = switch(day) { case MONDAY → 1; default → yield 0; };
- Makes switch logic cleaner and less error-prone.



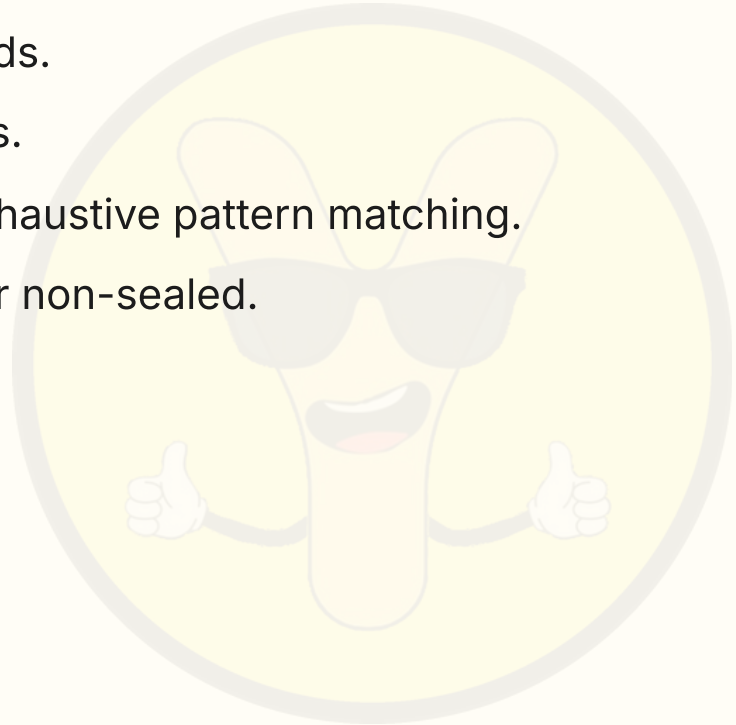
Text Blocks and Records

- Text Blocks: Multi-line string literals enclosed in triple quotes (""").
- Eliminates the need for concatenation and escape characters.
- Records: A transparent carrier for immutable data.
- Syntax: record Point(int x, int y) {}
- Automatically generates constructor, accessors, equals, hashCode, and toString.



Sealed Classes

- Sealed Classes: Restrict which classes can extend or implement them.
- Uses 'sealed' and 'permits' keywords.
- Ensures a fixed hierarchy of classes.
- Improves type safety and allows exhaustive pattern matching.
- Subclasses must be final, sealed, or non-sealed.



Stream API Pipeline



Data flows from source through intermediate operations to a terminal result.

Exam corner

- **Q1** Explain the role of Functional Interfaces with an example.
- **Q2** Differentiate between Lambda Expressions and Anonymous Inner Classes.
- **Q3** How does the Stream API improve collection processing?
- **Q4** Explain the use of 'var' and 'yield' keywords with examples.
- **Q5** What are Records? How do they differ from standard classes?
- **Q6** Describe the purpose of the Java Module System.
- **Q7** Explain the benefit of Try-with-resources over traditional try-catch-finally.
- **Q8** What are Sealed Classes and why are they used?

Unit 3 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 4 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

