The goal of this Project is to **complete a project from the ground up utilizing these principles**:

DRY

Don't
Repeat
Yourself

SOLID

Single Responsibility Principle: Each class has one responsibility.

Open Closed Principle: Classes are open for extension but closed for modification.

Liskov Substitution Principle: Subclasses can replace base classes without altering functionality.

Interface Segregation Principle: Avoid forcing classes to implement irrelevant methods.

Dependency Inversion Principle: Depend on abstractions, not concrete implementations.

Big O

Establish and Notate the Space and Time and Complexity for algorithmic operations, then refactor to optimize

Data Structure Planning and Implementation

Primitive Data Types

- Byte
- Short
- Int
- Long
- Float
- Double
- Char
- Boolean

Summary Table

Type	Size	Range/Values
`byte`	8 bits	-128 to 127
`short`	16 bits	-32,768 to 32,767
`int`	32 bits	-2,147,483,648 to 2,147,483,647
`long`	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`float`	32 bits	±3.40282347E+38F
`double`	64 bits	±1.79769313486231570E+308
`char`	16 bits	0 to 65,535
`boolean`	Not precisely defined	`true` or `false`

Reference Data Types

- Class
- Array
- Interface
- Enum

Type Conversion

- Implicit Casting (Widening)
- Explicit Casting (Narrowing)

Summary Table of Reference Data Types in Java

Type	Description	Examples
Class	A blueprint for creating objects, encapsulating data and methods.	`String`, `Scanner`, `File`
Interface	A contract that defines methods a class must implement, supporting multiple inheritance.	`Runnable`, `Comparable`, `List`
Array	A collection of elements of the same type, accessed by index.	`int[] arr = {1, 2, 3};`, `String[] names = new String[10];`
Enum	Represents a fixed set of constants, used for predefined values.	`enum Color { RED, GREEN, BLUE }`, `Color c = Color.RED;`
List	An ordered collection of elements that allows duplicates.	`ArrayList`, `LinkedList`, `Vector`
Set	An unordered collection of unique elements.	`HashSet`, `TreeSet`, `LinkedHashSet`
Queue	A collection that orders elements in a First-In-First-Out (FIFO) manner.	`PriorityQueue`, `LinkedList`
Deque	A double-ended queue that allows elements to be added/removed from both ends.	`ArrayDeque`, `LinkedList`

Map	A collection of key-value pairs with unique keys.	`HashMap`, `TreeMap`, `LinkedHashMap`
Graph	A collection of nodes (vertices) connected by edges, representing relationships.	Custom implementations, adjacency list, adjacency matrix
Tree	A hierarchical data structure where each node has a parent (except the root) and may have children.	Binary Tree, Binary Search Tree, `TreeMap`
HashMap	A hash table-based implementation of the `Map` interface for fast key-value lookups.	`HashMap<String, Integer>`
LinkedHashMap	A `Map` implementation that maintains insertion order.	`LinkedHashMap<String, Integer>`
TreeMap	A `Map` implementation sorted by keys, backed by a Red-Black Tree.	`TreeMap<Integer, String>`
Stack	A Last-In-First-Out (LIFO) collection for managing elements.	`Stack<Integer> stack = new Stack<>();`

Java Collection Framework

- Lists
  - ArrayList
  - LinkedList
  - Vector
  - Stack
- Trees
  - Binary Tree
  - Binary Search Tree
  - TreeMap
- Maps
  - HashMap
  - LinkedHashMap
  - TreeMap
- Graphs
  - Adjacency List
  - Adjacency Matrix
- HashMap
  - HashMap

Summary Table

Type	Use Case	Example
`List`	Ordered collection, allows duplicates	`ArrayList`, `LinkedList`
`Tree`	Hierarchical data structure	Binary Tree, `TreeMap`
`Map`	Key-value pairs	`HashMap`, `TreeMap`, `LinkedHashMap`
`Graph`	Represent connections between entities	Adjacency List, Adjacency Matrix
`HashMap`	Fast lookup and insertion	`HashMap`