Core Java Revision

Week-1

Prerequisite of Spring Boot

- ✓ Java Fundamentals
  - ✓ Variables
  - ✓ Control Statements
  - ✓ Data Types
  - ✓ Type Casting
  - ✓ Methods
  - ✓ Etc.
- ✓ Object Oriented Programming
  - ✓ Class and Object
  - ✓ Constructor
  - ✓ Overloading – CompileTime Polymorphism
  - ✓ Inheritance
  - ✓ Overriding - RunTime Polymorphism
  - ✓ Interfaces
  - ✓ Encapsulation

Prerequisite of Spring Boot

- ✓ Packages
- ✓ String Handling
- ✓ Exception Handling
- ✓ Collection Framework
  - ✓ List, Set, Map
  - ✓ Sorting Logics
    - ✓ Comparable and Comparator
  - ✓ Equality Logics
- ✓ Annotations
- ✓ Java 8 Features
- ✓ Java Database Connectivity[JDBC]
- ✓ Programming Skills
- ✓ IDE Eclipse/IntelliJ/Netbeans

Core Java Revision -1

- Object Oriented Programming
- OOP is a programming paradigm that organizes code around "objects,".
- Here are some key concepts of OOP:
  - **Objects**: Instances of a class that represent real-world entities or concepts. Object can have properties(attributes) and behaviors(method).
  - **Classes:** Blueprints for creating objects. They define the properties (data) and behavior (methods) that objects of that class will have.
  - **Encapsulation**: Bundling data and methods together within an object, protecting data from unauthorized access.
  - **Inheritance**: Creating new classes (subclasses) that inherit properties and behaviors from existing classes (superclasses).
  - **Polymorphism**: The ability of objects of different classes to respond differently to the same message.

Core Java Revision -1

- Object Oriented Programming
- OOP is a programming paradigm that organizes code around "objects,".
- Here are some key concepts of OOP:
  - **Objects**: Instances of a class that represent real-world entities or concepts. Object can have properties(attributes) and behaviors(method).
  - **Classes:** Blueprints for creating objects. They define the properties (data) and behavior (methods) that objects of that class will have.
  - **Encapsulation**: Bundling data and methods together within an object, protecting data from unauthorized access.
  - **Inheritance**: Creating new classes (subclasses) that inherit properties and behaviors from existing classes (superclasses).
  - **Polymorphism**: The ability of objects of different classes to respond differently to the same message.

Core Java Revision -1

- Constructor
  - The main purpose of a constructor is to initialize the object's state.
  - When you use **new**, memory is allocated for the object, and the constructor is invoked to perform any necessary setup for that object.
  - The constructor name must be identical to the class name.
  - **No return type**: Unlike methods, constructors don't have a return type (not even void).
  - **Implicit call**: Constructors are called implicitly when you use new. You don't explicitly call them like methods.
- **this** keyword
  - Call current class constructor
  - Refer current invoking object(use to call object data member and methods)

Core Java Revision -1

- Overloading
  - Constructor Overloading
    - Defining more than one constructor in same class is called constructor overloading.
    - Argument must be different.
  - Method Overloading
    - Defining more than one method with same name and different argument is called method overloading

Core Java Revision -1

- **Inheritance**
  - Acquiring properties and methods of one class by another class.
  - A class (**subclass**) can inherit attributes and methods from another class (**superclass** or **parent class**).
  - Core Reusability
  - Maintainability
- **Overriding**
  - When child class is not satisfied with parent methods then child class can redefine the method body .

Encapsulation

- Binding a data member and member methods in single unit class name.

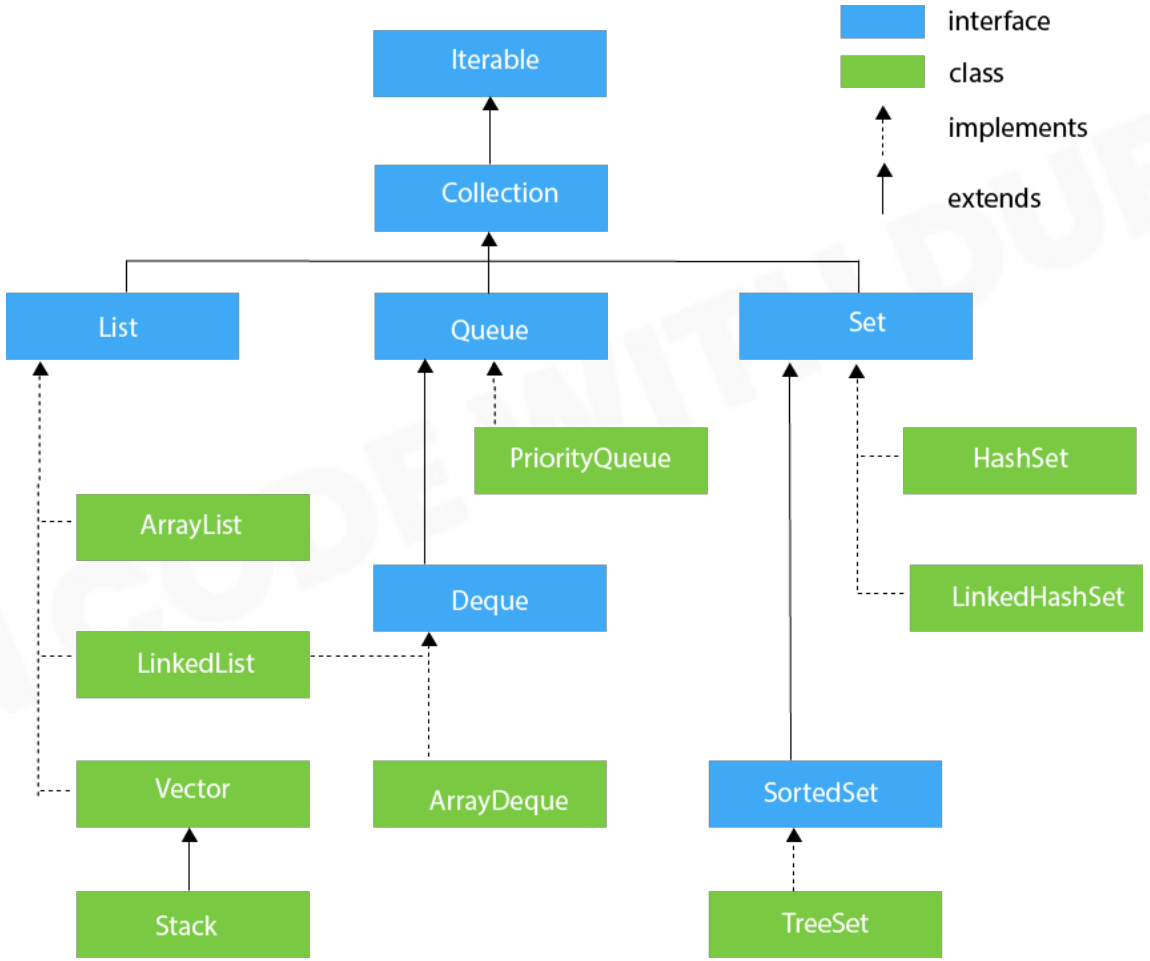
Abstraction

- Showing Features and Hiding Implementations.
- **Abstract Classes** and **Interfaces** are use to achieve abstraction in java.

Collection Framework

- Group of object represented as Single Unit is a **Collection**.
- **java.util** package provides all the classes and interfaces to work with group of objects(Collection)
- **Collection(Root Interface)** is a interface that resent group of objects as single unit.
- Collection Framework:
  - Interfaces and Its implementation
  - Algorithms

Collection Framework



Commonly used operation

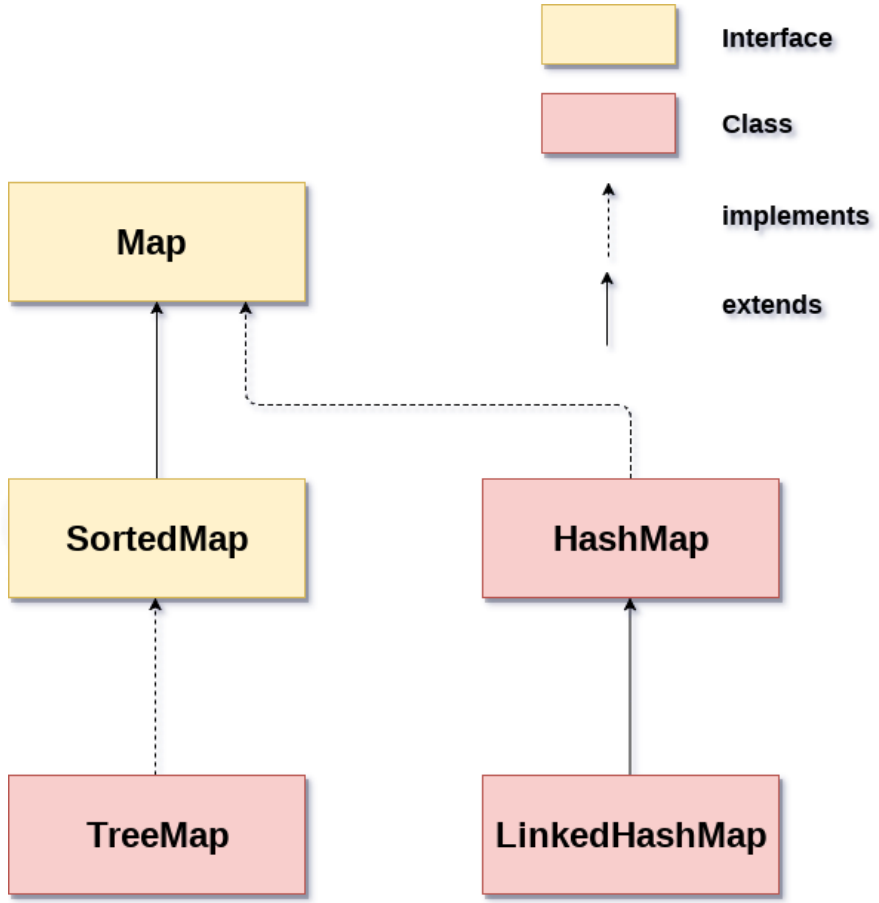
- Add element
- Get element
- Traverse Elements
  - Using Iterator
  - Using ListIterator
  - Using Enumeration
  - forEach
- Copy elements for other collection

Sorting Logic

- Comparable
  - Present in java.lang
  - Have to override compareTo method
  - Single sorting logic
- Comparator
  - Present in java.io
  - Have to override compre method
  - Multiple sorting logic

Collection Map API

- Map is used to represent collection of key value pair.



Common operations

- Add key-value pair
- Get value of key
- Traverse the key-value pair
- Remove key value
- Getting only keys
- How to get all the keyes

Important Java 8 Features

Lambda Expression

- Lambda Expression
- Short block of code which takes parameters and return value
- Lambda is use to implement Functional Interface.
- Syntax:
- parameter -> expression
- (parameter1, parameter2) -> expression
- (parameter1, parameter2) -> { code block }
- Parameters type : automatically refer
- Return Type : refer
- We can pass as parameter and we can return from any method.

Stream API

- The Stream API offers a declarative way to process collections of elements.
- It provides a chainable set of operations for **filtering**, **mapping**, and **reducing** data.
- **java.util.stream** package provides all the important classes and methods for stream api
- Create Stream: **Stream Interface**

Optional Class

- The Optional class introduced in Java 8 is a powerful tool for handling null references more safely and effectively.

Stream Operations

- ✓ **filter(Predicate):** Filters elements based on a condition.
- ✓ **map(Function):** Transforms each element using a function.
- ✓ **flatMap(Function):** Flattens a Stream of Streams into a single Stream.
- ✓ **sorted():** Sorts the elements in the Stream.
- ✓ **count():** Counts the number of elements in the Stream.
- ✓ **forEach(Consumer):** Performs an action on each element.
- ✓ **collect(Collector):** Collects the results of the Stream operations into a specific data structure.
- ✓ **reduce()**

Method references

static method ref
not-static method ref
constructor ref