Name: Gowtham Raja
Date: 22-11-2024

Department: Impact Training
Task: Java

1. Dictionary:

In Java, there is no direct data structure called Dictionary like in some other programming languages. However, Java provides several alternatives that function similarly to a dictionary (a collection of key-value pairs):

Interface:

The closest equivalent to a dictionary in Java is the Map interface. The Map interface represents a collection of key-value pairs, where each key is unique and maps to exactly one value. There are several common implementations of the Map interface:

a. **HashMap**:
   A widely used implementation of Map that stores keys and values with constant time complexity for lookups. It does not guarantee any specific order of the keys.
b. **TreeMap**:
   A Map implementation that keeps the keys in a sorted order, based on their natural ordering or a specified comparator.
c. **LinkedHashMap**:
   This implementation of Map maintains the insertion order of the keys.

So, while Java does not have a direct Dictionary class, you can use a Map (like HashMap or Hashtable) for similar functionality.

2. Difference b/w Interface and Collection:

Features	Interface	Collection
Definition	A blueprint for classes, defining method signatures without implementation.	A part of the Java Collections Framework that represents a group of objects.
Purpose	To define a contract that classes can implement.	To provide a group of objects and methods for managing collections of data.
Examples	Serializable, Cloneable, Comparable, Runnable, Iterable	Collection, Set, List, Queue.

Concrete Class	Cannot be instantiated directly, must be implemented by a class.	Can be implemented by concrete classes such as ArrayList, HashSet.
Methods	Defines method signatures; no implementation.	Defines methods for manipulating collections like add (), remove (), contains ().
Inheritance	Interfaces can be extended.	Collection is an interface itself and is extended by more specialized interfaces.

3.MAP

Map is an interface in the java.util package, and it maps keys to values, where each key is associated with exactly one value.

Key Characteristic of MAP:

- **Key-Value Pairs**: A Map stores data as key-value pairs. The key is used to access the corresponding value.
- **Unique Keys**: In a Map, each key must be unique, but multiple keys can map to the same value.
- **No Duplicate Keys**: If you try to add a duplicate key, the previous value for that key will be replaced with the new value.

❖ HashMap:

- The most used implementation of the Map interface.
- It allows null values and one null key.
- It does not guarantee the order of elements (i.e., no predictable order).

❖ TreeMap:

- A Map implementation that maintains its elements in a sorted order according to the natural ordering of its keys or a specified comparator.
- Does not allow null keys.

❖ LinkedHashMap:

- An implementation of Map that maintains the insertion order of elements.
- It preserves the order in which elements were added to the map.

- **Hashtable**:
  - An older implementation of the Map interface.
  - It is synchronized, which makes it thread-safe but slower than HashMap.
  - Does not allow null keys or values.

4.Throw and Throws:

Throw

The throw keyword is used to **explicitly throw an exception** from a method or a block of code. When you want to signal that something has gone wrong in your program, you can use throw to create and throw an exception.

Syntax:

```
{
throw new ExceptionType("Error message");
}
```

Throws

The throws keyword is used in a method signature to **declare** that the method **might throw** one or more exceptions during its execution. It tells the caller of the method that they need to handle (catch) or declare the exception further up the call stack.

Syntax:

```
public returnType methodName() throws ExceptionType1, ExceptionType2 {
}
```

Feature	Throw	Throws
Purpose	Used to throw an exception explicitly.	Used to declare that a method might throw one or more exceptions.
Used In	Method body or code block	Method signature (declaring the exception)

Syntax	throw new ExceptionType("message"); 	method() throws ExceptionType;
Control	Transfers control to the nearest matching catch block.	Notifies calling method that exceptions might be thrown, and it must handle or propagate them.

5.Interface names in JAVA

- **Runnable**
  Represents a task that can be executed by a thread.
- **Serializable**
  Indicates that a class can be serialized, i.e., converted into a byte stream for saving or transferring.
- **Cloneable**
  Allows an object to be cloned (i.e., a copy of the object can be made)
- **Comparable**
  Allows objects of a class to be compared to each other (often for sorting)
- **Iterable**
  Allows a class to be iterable using an iterator (e.g., for loops over collections)
- **Observer**
  Used in the Observer design pattern to define an object that gets notified of changes in another object
- **ActionListener**
  Used for handling action events (like button clicks in GUI applications)
- **EventListener**
  A generic interface for classes that want to receive events in an event-driven model
- **Readable**
  Describes objects that can be read (like a file or input stream)
- **Writable**
  Describes objects that can be written to (like output streams)

- **Lockable**
  Used for objects that can be locked, typically in concurrent programming

- **Flyable**
  Represents objects that can fly (like an aircraft)
- **Sortable**
  Allows objects to be sorted (often used for collections)
- **Persistable**
  Indicates that an object can be saved or persisted (e.g., to a database or file)
- **Comparable**
  Describes objects that can be compared to each other to establish an order

6. Difference b/w Interface and class

Feature	Class	Interface
Methods	Can have both abstract and concrete methods.	Can only have abstract methods (until Java 8) or default/static methods (Java 8+)
Fields	Can have instance fields (state)	Can only have constants (static final fields)
Constructor	Can have constructors	Cannot have constructors
Inheritance	Can extend only one class	Can extend multiple interfaces
Installation	Can be instantiated (e.g., new Dog()).	Cannot be instantiated directly.
Access Modifiers	Can have any access modifier (public, private, protected).	All methods and fields are implicitly public
Usage	Used to define the structure and behavior of objects	Used to define a contract or behavior that classes must implement

7. Vector:

- In Java, Vector is a class that implements the List interface and represents a growable array of objects.

- It is part of the java.util package and provides a way to store and manipulate a collection of objects.
- Although it was part of the original version of Java, Vector is now considered somewhat obsolete, and newer classes like ArrayList are preferred for most use cases.
- However, Vector is still used in legacy codebases or situations that require thread-safety.

8. Thread Safety:

- Thread safety in programming refers to the concept of ensuring that multiple threads can safely execute a piece of code concurrently without causing any inconsistencies, data corruption, or unexpected behavior.
- In Java, thread safety is particularly important in multi-threaded environments where multiple threads access shared resources, such as variables, data structures, or objects, concurrently.

9. Priority Queue

- A Priority Queue in Java is a special type of queue that allows elements to be processed based on their priority rather than their order of insertion.
- In a standard queue, elements are processed in a first-in, first-out (FIFO) order, but in a priority queue, elements are ordered by their priority. Elements with higher priority are dequeued before elements with lower priority, regardless of when they were added to the queue.
- Java provides a built-in Priority Queue class as part of the java.util package to implement this data structure. The elements in a Priority Queue are ordered either according to their natural ordering (for example, numbers or strings sorted in ascending order) or by a custom comparator that you can define.

10. Difference b/w Collection Class and Collection Interface:

Aspect	Collection Class	Collection Interface
Definition	An interface that defines the basic methods for working with collections.	Concrete classes that implement the Collection interface (e.g., ArrayList, HashSet, etc.).

Name: Gowtham Raja
Date: 22-11-2024

Department: Impact Training
Task: Java

Purpose	Provides the common contract (methods) for all collections (lists, sets, etc.).	Provides concrete implementations of collection types with actual behavior (storage and manipulation of elements)
Instantiable	Cannot be instantiated (since it's an interface	Can be instantiated (e.g., ArrayList, HashSet are concrete classes)
Implementation	Only method signatures; does not provide any implementation for methods.	Provides the actual implementation for methods defined by the Collection interface
Example	Collection, List, Set, Queue (these are all interfaces in the Collections Framework).	ArrayList, LinkedList, HashSet, TreeSet, PriorityQueue (these are concrete classes).