Software’s required

1. Java 17
2. PostgreSQL – zip
3. STS/Eclipse IDE
4. Node.js 16 or later
5. VS Code

OOPS concepts

1. Encapsulation – Java Beans (private data & public methods)
2. Inheritance - extends
3. Abstraction – abstract class & interface
4. Polymorphism – overloading & overriding

interface A {   
 void m1();  
}

class B implements A {   
 public void m1() { … }  
}  
class C implements A {   
 public void m1() { … }  
}  
  
Factory pattern: it abstracts object creation  
class Z {   
 public static A getInstance() { return new B(); //or return new C(); }  
}

//1st way

Client program

A a1 = new B();  
a1.m1();

2nd way

Client program

A a1 = Z.getInstance(); //   
a1.m1();

Collection Framework

It provides APIs to manage the data in various forms

* List: Index based – ArrayList, LinkedList
* Set: Stores only unique data – HashSet, TreeSet, LinkedHashSet
* Queue: Removes the data in FIFO / Sorted order – PriorityQueue, ArrayDeque

Map: It maintains the data in key value pairs

* HashMap
* LinkedHashMap
* TreeMap

Collection Framework – add(), remove(), size(), clear(), iterator(), forEach(), stream()

Collection: It is an interface at the top level, which provides all the methods

List, Queue & Set extends Collection – these are also interfaces

Comparable: It is used to provide default sorting technique to the complex objects, it has a method

* public int compareTo(T t);

compareTo returns an int value which could be 0 or +ve or -ve when two values are compared

Integer.compare(x, y): It compares two int values and returns 0 or +ve or -ve

implementing comparable

class User implements Comparable<User> {  
 userId, name, dob;  
   
 public int compareTo(User other) {   
 return Integer.compare(userId, other.userId);  
 }  
}

Comparator<T>: It is used to provide multiple sorting logic when you want to override the default sorting logic i.e., Comparable<T>, it is not implemented in the same class, instead implemented independently and its objects is supplied to the TreeSet, it has a method called

* public int compare(x, y)

Implementing Comparator

class SortById implements Comparator<User> {

public int compare(User u1, User u2) { ... }

}

class SortByName implements Comparator<User> {

public int compare(User u1, User u2) { ... }

}

Set s1 = new TreeSet(new SortByName());

Set s2= new TreeSet(new SortById());

// local anonymous classes

Comparator<User> c1 = new Comparator<User>() {

public int compare(User u1, User u2) { ... }

}

Comparator<User> c2 = new Comparator<User>() {

public int compare(User u1, User u2) { ... }

}

Set s1 = new TreeSet(c1);

Set s2= new TreeSet(c2);

// lambda expression : interface having only one abstract method

Comparator<User> c3 = ( u1, u2 ) -> { return Integer.compare(u1.id, u2.id) }

Comparator<User> c4 = ( u1, u2 ) -> Integer.compare(u1.id, u2.id) //

Comparator c5 = new Comparator<User> {

public int compare(User u1, User u2) {

return Integer.compare(u1.id, u2.id);

}

}

Set<User> set = new TreeSet(c4);

Set<User> set = new TreeSet( (u1, u2) -> Integer.compare(u1.id, u2.id) );

Streams & Lambda expressions

Streams: These take care of working with collections in a way so that you can process and manipulate it in an easier way

Suppose you have a collection & you want to apply some conditions

Functional Interface: Interfaces which have only one abstract methods, you can pass function itself as a parameter instead of passing an object

public void test(X x) {   
  
}

test( lambda expression );

List of functional interfaces

|  |  |  |
| --- | --- | --- |
| Functional Interface | Method name | Lambda expression |
| Comparator<T> | int compare(x, y) | (x, y) -> intValue; |
| Predicate<T> | boolean test(T t) | t -> booleanValue |
| Consumer<T> | void accept(T t) | t -> statement |
| Function<T> | T apply(T t) | t -> someValue |

Stream methods

1. filter(Predicate) : filter(t -> t == 2)
2. forEach(Consumer): forEach(t -> System.out.print(t) )
3. sorted(Comparator): sort( (x, y) -> intValue )
4. map(Function): map(t -> someValue)

Activity:

1. Create a class Item with id, name, price, ratings – generate 2 constructors, setters, getters, toString
2. Create some Item objects atleast 10 to 15 in the main method and add them in a List<Item>

ex: name: Mobile, price:20000, ratings: 4.5

Stream has two types of methods

1. Intermediate methods: This generates another stream where you can chain the methods, ex: filter, sort, map,
2. Terminal methods: This is the end of stream, which doesn’t generate another stream, here you can’t chain the methods, ex: forEach, sum, count, collect

Chaining the methods

filter(..).filter(..).filter(…)

filter(…).map(…).sort(..)

PostgreSQL

It is an RDBMS software that maintains the data, SQL is the language RDBMS understand.

Sub languages of SQL

1. DDL: create, alter, drop & truncate
2. DML: insert, update & delete
3. DQL: select
4. DCL: grant, revoke
5. TCL: commit, rollback

Create a folder named pgsql\_data in some drive like C or D

How to set username & password

initdb.exe -D pathOfpgsql\_data -U username -W -E UTF8 -A scram-sha-256

How to start the database

**pg\_ctl -D D:\Labs\pgsql\_data -l logfile start**

How to login to the database

psql.exe -U postgres

Datatypes

* int
* varchar
* bigint
* float
* date
* json

Constraint: primary key, not null, foreign key, unique, check

Functions: upper(), lower(), to\_date(), sum(), count(), avg()

Views: It is a virtual table that refers to the select query

create view view\_name as (sql query)

Index: It is used to increase search performance