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

Full Text Search

like operator is used to perform the text search, but it is not effective during the complex searches like searching from the dictionary words or the words which are derived from the base word

where text like ‘%a%’; # this will search the whatever the letters comes before or after a

where text like ‘satisf%’; # this will search satisfy, satisfied, satisfies

like operator doesn’t ignore stop words like from, to, where, and, a, and so on.

Full Text search can ignore lot of stop words and searches the text derived from the base words

ex: satisfy : it searches satisfies, satisfied, satisfy

ex: inspire: it searches inspires, inspired, inspiration

tsvector: it is a datatype, it would parse the text (document) & break the text into tokens and looks each token in the dictionary to find the lexemes(base words and their derived words), it also ignores the stop words like a, and, or, between, then, from, and so on.

tsquery: it is used to mention the word that you want to search

Example: using tsvector & tsquery

select \* from test where to\_tsvector(description) @@ to\_tsquery(‘friendly’);

the above query finds all the derived words from the base word friend like friends, friend, friendly

Searching the lexemes

select name, description from test;

select name, description from test where to\_tsvector(description) @@ to\_tsquery('satisfy');

select name, description from test where to\_tsvector(description) @@ to\_tsquery('friend');

Searching from the stop words

select name, description from test where to\_tsvector('simple',description) @@ to\_tsquery('simple','to');

Concatenating the documents to find the words in multiple columns

select name, description from test where to\_tsvector(name ||’ ‘||description) @@ to\_tsquery(‘word’);

Transaction isolation

When concurrently query is executed user might sometimes see the other user updates and sometimes they might not see other user updates

1. Read committed
2. Repeatable Read

Read Committed:

create table t1(id int, name varchar(10));

open second command prompt;

CMD1 : insert 2 records

CMD2 : select query – shows those two records, because by default its auto-commit & transaction isolation level is read commit

Repeatable Read: In this isolation level, a user under transaction can’t see another user changes even if the another user has done commit in the current transaction, they can see the changes in the next transaction

T1 -> begin transaction isolation level repeatable read;

T2 -> begin; -> update some data -> commit;

T1 -> select -> you don’t see the T2 changes

T1 -> commit/rollback -> you see the T2 changes

Activity:

T1 -> begin transaction isolation level repeatable read;

T2 -> begin; -> update some data

T1 -> update same data of T2 -> you will see T1 will be locked

T2 -> commit/rollback -> you will see the T1 will be unlocked

Performance Tips

1. You must not use \* while using select, instead you must use column names which you want to retrieve
2. Avoid using distinct because it selects common value & groups them as a single value
3. Use limit when you want to preview the result
4. Use index to increase search speed
5. Use like in the beginning of the phrase i.e., like ‘A%’ instead of ‘%A%’
6. Use full text search and index the vector(tsvector) to increase the search speed
7. Use prepared statement in the application that executes the query for different values without reparsing and recompiling the query each time

Note: In Postgresql you can use explain analyze command to find out how much time a query takes to execute

ex: explain analyze select name, description from test;

Query to create tsvector in the table

alter table test add document tsvector;

update test set document=to\_tsvector (name||' '||description);

Query without indexing

explain analyze select name, description from test where to\_tsvector (name||' '||description) @@ to\_tsquery('friend');

Output: Execution time will be shown

Query with indexing

explain analyze select name, description from test where document @@ to\_tsquery('friend');

Output: Execution time will be shown which will lesser to the previous query

JDBC: Java Database Connectivity that provides set of API’s to interact with any database

JDBC API’s >> Database Vendors provide the implementations for their database

i.e.,

MySQL has implemented JDBC API to interact with MySQL database

Postgres has implemented JDBC API to interact with Postgres database

These vendors provide the implementations in a jar file which you must use in your project.

How to interact with the database using JDBC

1. Loading the JDBC drivers (implementations) – optional in JDBC

Class.forName(driverClassName);

1. Connecting to the database

Connection connection = DriverManager.getConnection(url, un, pwd);

1. Creating the statements

PreparedStatement statement = connection.prepareStatement( sqlQuery );

sqlQuery = “insert into test values(?, ?, ?)”;

sqlQuery = “select name, phone from test where id = ?”;

// setting values to the ? positions  
statement.setInt(1, value);

statement.setString(2, value);  
statement.setDouble(3, value);  
statement.setString(4, value);

1. Execute statements

int count = statement.executeUpdate(); // for DML queries

ResultSet result = statement.executeQuery(); // for DQL queries

1. Closing resources

result.close();  
statement.close();  
connection.close();

Activity:

1. Create a Student class with 3 properties rollNo, name, dob, retrieve all the student records from the database and store each record’s value to the corresponding Student object properties and then add that student object to the List<Student> and return the List<Student>
2. Perform a store operation by storing rollNo, name & dob in the database by getting the values from the student object

ex: Student student = new Student(5, ‘Frank’, LocalDate.parse(‘2000-09-20’));

student object must be stored in student table where roll\_no = 5, name = Frank and dob = 2000-09-20

Note: Create a separate class like StudentDaoJdbcImpl for database operations & call them from the main method, the class must have two methods as below

1. public int save(Student student) { … } // stores student using insert query
2. public List<Student> findAll() { … } // retrieves all the students
3. public Student find(int rollNo) { … } // this returns student based on rollNo