ID: IT-22030

1. Protected Member Access in Inheritance

Same Package Access:

```
package com.example
class Parent {
    protected int protectedVar = 10;
}
class child extends Parent {
  void access Protected {
   System.out.println (protectedVar);
  }
}
```

Different Package Access:

```
package com.example.other;
```

```
import com.example.Parent;
class child extends Parent {
void access Proteeted() {

}

}
```

2. Abstract Classes vs Interfaces (Multiple Inheritance)

- Abstract Classes: Support single inheritance (can extend only one class)

- Interfaces: Support multiple inheritance (can implement multiple interfaces)

When to use:

- Use abstract classes when you need to share code among related classes.
- Use interfaces when you need to define a contract for unrelated classes.

3. Encapsulation with Bank Account Example:

```
public class BankAccount {
    private String accountNumber;
    private double balance;
```

```
public void setAccountNumber(String
accountNumber) {
  if (accountNumber == null accountNumber
  is Empty ()) {
  throw new IllegalArgumentException
  ("Account number cannot be null or empty");
  }
this.accountNumber = accountNumber;
}
public void setInitiaBalance(double balance){
  if (balance<0){
    throw new IllegalArgumentException
    ("Balance cannot be negative");
  }
```

```
        this.balance = balance
    }
}
```

4. Program Implementations (3 examples)

(i) kth smallest element in Arraylist:

Java

```
public static int findkthSmallest (Arraylist<
Integer> list int k) {
    Collections.sort (list);
    return list.get (k-1);
}
```

(ii) TreeMap for word frequencies:

```
public static TreeMap<String, Integer>
getWordFrequencies(String text) {
    TreeMap<String, Integer> freqMap
    = new TreeMap<>();
    for (String word : words) {
        freqMap.put(word, freqMap.get
        OrDefault(Word, 0)+1);
    }
    return freqMap;
}
```

(v) Check if two LinkedLists are equal:

```
public static boolean areLinkedListsEqual
(LinkedList<?> list1, LinkedList<?> list2) {
```

```
    return list1.equals(list2);
}
```

5. Singleton Pattern with Thread Safely

```
public class Singleton {
    private static volatile Singleton instance;
    private Singleton() { }

    public static Singleton getInstance(){
        if (instance == null) {
            instance = new Singleton();
        }
    }
}
```

```
    }
    return instance ;
  }
}
```

6. JDBC SELECT Query Example

```
try (Connection conn = DriverManager.get
Connection (DB URL, USER, PASS);

    statement stmt = conn. createStatement();
    ResultSet rs = stmt. executeQuery ("SELECT
    id, name From students ")) {

    while (rs.next ()) {
```

```
        int id = rs.getString("name");
        System.out.println("ID: " + id + ", Name:" +
        name );
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

7. Servlet Lifecycle

1. init(): Called once when servlet is first loaded

2. service(): Handles each request (calls doGet(), doPost(), etc.)

3. destroy(): Called when servlet is taken out of service

8. MVC in Web Applications

Student Registration Example:

- Model: Student.java (POJO with fields and business logic)
- View: studentForm.jsp (HTML form for user input)
- Controller: StudentServlet.java (processes requests, updates model, selects view)

9. Session Tracking Methods

- Cookies : Stored client-side, limited size, can be disabled
- URL Rewriting : Appends session ID to URLs, works without cookies but insecure
- HttpSession : Server-side storage, most secure but uses server memory

10. Spring MVC Request Flow

1. DispatcherServlet receives request
2. Handler Mapping finds appropriate @Controller
3. @Controller processes request, returns ModelAndView

4. View Resolver determines view to render

5. View renders response

11. Prepared statement Vs Statement

Java

```
String sql = "INSERT INTO employees
(name, age) VALUES (?, ?)";

try (PreparedStatement pstmt = conn.
prepare Statemen (sql)) {
    pstmt.setString (1, "John Doe");
    pstmt.setInt (2, 30);
    pstmt.executeUpdate ();
}
```

Advantages:

- Prevents SQL injection
- Better performance (pre-compiled)
- Easier to work with parameters

12. JPA Entity Mapping Example

```
@Entity
public class student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

Advantages:

- Prevents SQL injection
- Better performance (pre-compiled)
- Easier to work with parameters

12. JPA Entity Mapping Example

```
@Entity
public class student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    private string name;
}
```

Advantages over JDBC:

- Object-oriented approach
- Less boilerplate code
- Database-agnostic
- Built-in caching

13. Spring Boot REST Controller

```
@RestController
@RequestMapping("/api/books")
public class BookController {
```

```
@GetMapping
public List<Book> getAllBooks() {
    return bookService.findAll();
}

@PostMapping
public Book addBook(@RequestBody Book
book) {
    return bookService.save(book);
}
}
```

14. @RestController vs @Controller

- @Controller : For traditional MVC (returns view names)
- @RestController : For REST APIs (returns data directly, combines @Controller + @ResponseBody)

15. Maven pom.xml structure

```xml
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
```

<groupId> org.springframework.boot</groupId>

<artifactId> spring-boot-started-parent</artifactId>

<version> 2.5.0</version>

</parent>

</dependencies>

<dependency>

<groupId> org.springframework.book</groupId>

<artifactId> spring-boot-started-wed</artifactId>

</dependency>

</dependencies>

</dependencies>

</project>

16. Virtual DOM in React

Traditional DOM updates are slow because they directly manipulate the browser DOM. Virtual DOM:

1. Creates light weight copy of real DOM
2. Updates virtual DOM first
3. Uses diffing algorithm to find minimal changes
4. Updates only changes parts in real DOM

17. Event Delegation in JavaScript

Instead of adding event listeners to each element, add one to a parent:

```
javascript

document.getElementById('Paren').addEvent
Listener ('click', function(e){
```

1. Creates light weight copy of real DOM
2. Updates virtual DOM first
3. Uses diffing algorithm to find minimal changes
4. Updates only changes parts in real DOM

17. Event Delegation in JavaScript

Instead of adding event listeners to each element, add one to a parent:

javascript

```
document.getElementById('Paren').addEvent
Listener('click', function(e){
```

```
    if (e.target.classList.contains('dynamic
    -element')) {

    }

});
```

18. Java Regular Expressions

```
String emailRegex = "^[A-Za-z0-9+_.-]
@[A-Za-z0-9.-]+$";

Pattern pattern = Pattern.compile
(emailRegex);
```

```
Matcher matcher = pattern.matcher(input);
if (matcher.matches()) {

}
```

19. Custom Annotations

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface LogExecutionTime{}
public class Processor {
```

```
public static void processAnnotations
(Object obj) {
    for (Method method : obj.get
    class().getMethods())}
if (method.isAnnotationPresent
(LogExecutionTime.class)) {

}

}

}
```

20. DOM vs SAX Parsers

- DOM: Loads entire document into memory (good for small files, random access)
- SAX: Event-based, reads sequentially (good for large files, memory efficient)

21. Car Parking Simulation (Multithreading)

```
class ParkingPool {
    private Queue<RegistrarParking> queue
    = new LinkedList<>();
```

20. DOM vs SAX Parsers

- DOM: Loads entire document into memory (good for small files, random access)
- SAX: Event-based, reads sequentially (good for large files, memory efficient)

21. Car Parking Simulation (Multithreading)

```
class ParkingPool {
    private Queue<RegistrarParking> queue
    = new LinkedList<>();
```

```
public synchronized void addParking
Request(RegistrarParking request){
queue.add(request);
notifyAll();
}

public synchronized RegistrarParking
getNextRequest() throws Interrupted
Exception {
while (queue.isEmpty()){
wait();
}
return queue.remove();
}
}
```

```
class ParkingAgent extends Thread {
    public void run(){
        while (true){
            RegistarParking request = pool.getNext
            Request();
            system.out.println("Agent" + id + "parked
            car" + request.getCarId());
        }
    }
}
```

22. Servlet Concurrency Issues

Problem example:

```
Public class counterservelet extends HttpServelet{
    private int count = 0;
  Protected void doGet(HttpServletRequest
        req, HttpServeletResponse res){
    count ++;
  }
}
```

Solution:

```
Synchronized (this) {
    coont ++
}
```

23. Servlet to JSP Forwarding

```
protected void doPost(HttpServletRequest request,
                      HttpResponse response) {
String name = request.getParameter("name");
request.setAttribute("username", name);
request.getRequestDispatcher("welcome.jsp").
        forward(request, response);
```

24. HttpSession Management

Sessions:

- Created on first request with request.getSession()

- stored server-side with unique JSESSIONID cookie
- Timeout configured in web.xml (<session-timeout>30</session-timeout>)
- Invalidated with session.invalidate()

25. Spring Boot CRUD with mysql :-

```
@Repository
public interface studentRepository extends
    JpaRepository<student, Long>{ }

@Service
public class studentService {
public list<student> findAll() { return
    repository.findAll(); }
```

public student save (students) { return repository.save(s); }

public void delete(Long id) { repository.deleteById(id); }

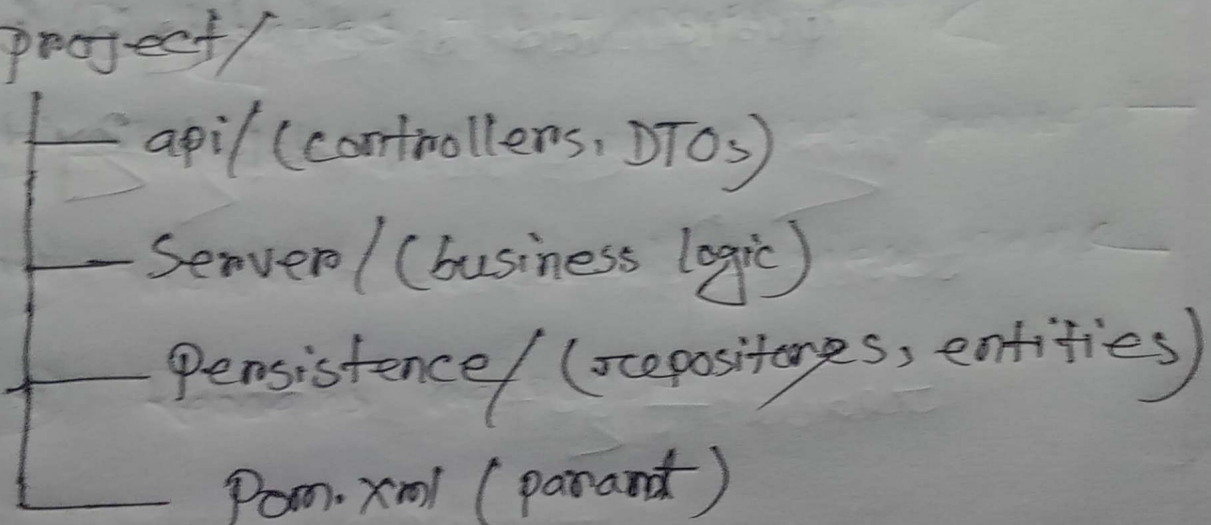
}

26. Entity Manager Operation

- persist(): Makes a transient instance persistent
- merge(): Updates a detached entity
- remove(): Marks an entity for deletion.

24. Multi-module Spring Boot project:

structure:



28. Resultset methods:

```
Resultset rs = stmt.executeQuery("SELECT name,
                age FROM people");
while (rs.next()) {
    String name = rs.getString("name");
    int age = rs.getInt("age");
    System.out.println(name + ":" + age);
}
```

29. Dispatcher Servlet workflow

i) Receives HTTP request

ii) Delegates to HandlerMapping

iii) HandlerAdapter processes request

iv) ViewResolver resolves view name

v) Renders response

30. Spring Boot starters:

Starters simplify dependency management by building common dependencies;

i) Spring-boot-starter-web (for web apps)

ii) Spring-boot-starter-data-jpa (for JPA)

iii) Spring-boot-starter-test (for testing)