Day 11

CallableStatement

- If we want to execute Stored procedure and function then we should use CallableStatement object.
- Syntax:

```
Statement statement = connection.createStatement();
PreparedStatement statement = connection.prepareStatement();
CallableStatement statement = connection.prepareCall();
```

- Stored Procedure to insert book in BookTable

```
DELIMITER $$
CREATE PROCEDURE sp_insert_book
(
    IN pBookId INT,
    IN pSubjectName VARCHAR(256),
    IN pBookName VARCHAR(256),
    IN pAuthorName VARCHAR(256),
    IN pPrice FLOAT
)
BEGIN
    INSERT INTO BookTable
    VALUES (pBookId, pSubjectName, pBookName, pAuthorName, pPrice);
END $$
DELIMITER ;
```

- Stored Procedure to update book in BookTable

```
DELIMITER $$
CREATE PROCEDURE sp_update_book
(
    IN pBookId INT,
    IN pPrice FLOAT
)
BEGIN
    UPDATE BookTable
    SET price=pPrice
    WHERE book_id=pBookId;
END $$
DELIMITER ;
```

- Stored Procedure to delete book from BookTable

```
DELIMITER $$
CREATE PROCEDURE sp_delete_book
(
    IN pBookId INT
)
BEGIN
    DELETE FROM BookTable
    WHERE book_id=pBookId;
END $$
DELIMITER ;
```

- Stored Procedure to get books from BookTable

```
DELIMITER $$
CREATE PROCEDURE sp_select_book( )
BEGIN
    SELECT * FROM BookTable;
END $$
DELIMITER ;
```

- Synatax to call stored procedure from java application. { call procedure_name(arg1, arg2, ...) }
- Synatax to call stored function from java application. {?= call procedure_name(arg1, arg2, ...)}

JDBC Transaction

- If we execute sql statement using JDBC then it is auto commited. If we want to change this behavior then we should use methods declared in java.sql.Connection interafce.
- Methods of Connection interface:

1. Statement createStatement() throws SQLException
2. PreparedStatement prepareStatement(String sql) throws SQLException
3. CallableStatement prepareCall(String sql) throws SQLException
4. void setAutoCommit(boolean autoCommit) throws SQLException
5. void commit()throws SQLException
6. void rollback()throws SQLException
7. Savepoint setSavepoint(String name)throws SQLException

```
CREATE TABLE accounts
(
    acc_number INT,
    name VARCHAR(256),
    balance FLOAT
);
INSERT INTO accounts VALUES( 101, 'Amol', 50000);
INSERT INTO accounts VALUES( 102, 'Rupesh', 90000);
SELECT * FROM accounts;
```

- If we create new connection then it execute every sql statement in auto commited mode.

```
Connection connection = null;
try
{
    connection = DBUtils.getConnection();
    connection.setAutoCommit(false);
    //TODO : DML
    connection.close();
}
catch( Exception ex )
{
    ex.printStackTrace();
    connection.rollback();
}
finally
{
    connection.close();
}
```

ResultSet Type

```
1. ResultSet.TYPE_FORWARD_ONLY
2. ResultSet.TYPE_SCROLL_INSENSITIVE
3. ResultSet.TYPE_SCROLL_SENSITIVE
```

ResultSet Concurrency

```
1. ResultSet.CONCUR_READ_ONLY
2. ResultSet.CONCUR_UPDATABLE
```

Multithreading

- Light weight process / sub process is called Thread.
- It is non java resource / OS resource.
- If operating system execute multiple processes simultaneously then it is called multitasking. In OOPs context it is called as concurrency.
- We can achieve multitasking using process as we all thread.
- If we use multiple threads for execution of application then it is called multithreaded application.
- Every Java application is multithread.

1. Thread is OS resource. To access OS thread, java application developer need not to do System programming. SUN/ORACLE has developed framework to access OS thread. In other words java supports multithreading.
2. When JVM starts execution of java application(.class) it also starts execution of main thread and garbage collector. Because of these two threads, every java application is multithreaded.

Main Thread

- It is user thread / non daemon thread.
- It is responsible for invoking main( ) method.
- Default priority of main thread is Thread.NORM_PRIORITY.

Garbage Collector / GC / Finalizer

- It is a daemon thread / background thread.
- It is responsible for releasing memory of unused objects/instances
- Default priority of main thread is Thread.NORM_PRIORITY + 3.

Types of thread

1. User Thread
    - It doesn't depend of its child thread. Once job is over it gets terminated.
2. Daemon Thread
    - It depends on its child thread. Once job is over it waits for child threads to be terminated.

- If we terminate daemon thread forcefully then all its child threads also gets terminated.
- If we create thread programatically then it is by default User Thread.
- isDaemon() method is used to check whether thread is Daemon / User thread. setDaemon() method is used to convert user thread into Daemon thread.

```
Thread thread = ....;
if( !thread.isDaemon())
    thread.setDaemon( true );
```

- If we want to manage OS thread from java then we should use types declared in java.lang package.
- Types
    1. Runnable : Interface
    2. Thread : Class
    3. ThreadGroup : Class
    4. ThreadLocal : Class
    5. Thread.State : Enum
    6. InterruptedException : Exception
    7. IllegalMonitorStateException : Exception
    8. IllegalThreadStateException : Exception

Runnable

- It is Functional interface declared in java.lang package.
- "void run()" is a abstract method of java.lang.Runnable interface
- run( ) method is called business logic method.
- If we want to create thread then we can take help of Runnable interface.

Thread.State(Nested Type)

- State is enum declared in java.lang.Thread class
- A thread can be in only one state at a given point in time.
- A thread can be in one of the following states:

```
NEW             :       0
RUNNABLE        :       1
BLOCKED         :       2
WAITING         :       3
TIMED_WAITING   :       4
TERMINATED      :       5
```

- These states are virtual machine states which do not reflect any operating system thread states.
- It is introduced in JDK 1.5

Thread

- It is a class declared in java.lang package.
- It implements Runnable interface.
- java.lang.Thread is a managed version of unmanaged thread. In other words, instance of Thread class is not a OS thread rather it represents operating System Thread. Mapping Thread instance to OS thread is a responsiblity of JVM.
- If we want to create thread then we can take help of Thread class
- Nested Type
    1. Thread.State
- Field
    1. public static final int MIN_PRIORITY; //1
    2. public static final int NORM_PRIORITY //5
    3. public static final int MAX_PRIORITY //10
- Constructor
    1. public Thread()

```
Thread thread = new Thread( );
```

```
2. public Thread(String name)
```

```
Thread thread = new Thread( "User Thread" );
```

```
3. public Thread(Runnable target)
```

```
Runnable target = new CThread( );
Thread thread = new Thread(  target );
```

```
4. public Thread(Runnable target, String name)
```

```
Runnable target = new CThread( );
Thread thread = new Thread(  target, "User Thread" );
```

```
5. public Thread(ThreadGroup group,Runnable target, String name)
```

```
ThreadGroup group = new ThreadGroup("User Thread Group");
Runnable target = new CThread( );
Thread thread = new Thread(  group, target, "User Thread" );
```

- Methods

1. public static Thread currentThread()
2. public final String getName()
3. public final void setName(String name)
4. public final int getPriority()
5. public final void setPriority(int newPriority)
6. public Thread.State getState()
7. public final ThreadGroup getThreadGroup()
8. public void interrupt()
9. public final boolean isAlive()
10. public final boolean isDaemon()
11. public final void setDaemon(boolean on)
12. public final void join() throws InterruptedException
13. public static void sleep(long millis) throws InterruptedException
14. public void start()
15. public static void yield()

```
Thread thread = Thread.currentThread();
System.out.println(thread.toString());
//Thread[main,5,main] //thread's name, priority, and thread group.
```

- Generally, we should avoid use of blocking calls in multithreaded application.
- Following are blocking calls:
    1. sleep()
    2. suspend()
    3. join()
    4. wait()
    5. Input operation