* Dirty reads : There are 2 tansactions, X and Y. If X reads an uncommited transaction by Y, then this is called dirty read. For example: If Y wants to update a value and X wants to read the same value.

① X begins read and Y begins write.
② Y changes value V_1 to V_2. X read value as V_2
③ some error occured in Y and it rolled back.
④ V_2 became V_1 as a result of rollback and X read value as V_2.
Thus a dirty read.

* Non Repeatable Reads :- this is about different data
● Almost same as dirty reads.
① X read some value V_1. Next Y comes and updates $V_1 \Rightarrow V_2$ + commits.
② X somehow wants to read value again, this time it get V_2.
getting different value for same ~~[illegible]~~ column ~~[illegible]~~ twice in same transaction is called non-repeatable reads.

* Phantom Reads :- Getting new ~~value~~ set of data ~~[illegible]~~ 2 time in diff same transaction. This is about more data
① X read all the rows where value is V_1. Gets 5 rows
② Y write 2 more rows with value V_1. commits
③ X ~~[illegible]~~ repeats ①. this time gets 7 rows. ⇒ Phantom Reads.

Shared Lock :- also ~~[illegible]~~ knowns as read locks. Use to prevent data modification. Multiple transaction can acquire ~~sh~~ shared lock on a single row.

Range Lock :- ~~[illegible]~~ Acquire lock on a range of rows. Prevent addition of new rows within this range or updation of any ~~[illegible]~~ row.

Isolation Levels :-
① Read Uncommited :- Allows txn to read uncommited changes.
DR, NRR, PR all possible.
② Read commited :- txn can only see commited changes.
prevents DR, but not NRR, PR.
③ Repeatable Read :- Txn acquire share lock. on all the rows it want read. Prevents DR, NRR but not PR
④ Serializeable :- Txn accquires range lock. on rows it is ~~[illegible]~~ reading. Prevents all.

Propogation in spring transaction :-

Required (default) :- if txn exists will execute ~~wn~~ within that txn, else will create a new txn.

Supports :- if exists will execute within that txn else will execute without the txn.

Mandatory :- If not txn is present the will throw an exception (basically a pre existing txn is mandatory.)

Requires new :- New txn is always created. If a pre existing txn is present then that will be ~~f~~ paused untill this txn completes.

Not supported :- Will execute without a txn even if a txn is present.

Never :- Will throw error if a txn is present (opposite of Mandatory)

Nested :- Will create new nested transaction if a transation exist If not will behave same as Required

---

** Transaction wont work on checked exception by default.
Need to use (roll back for = Exception.class)