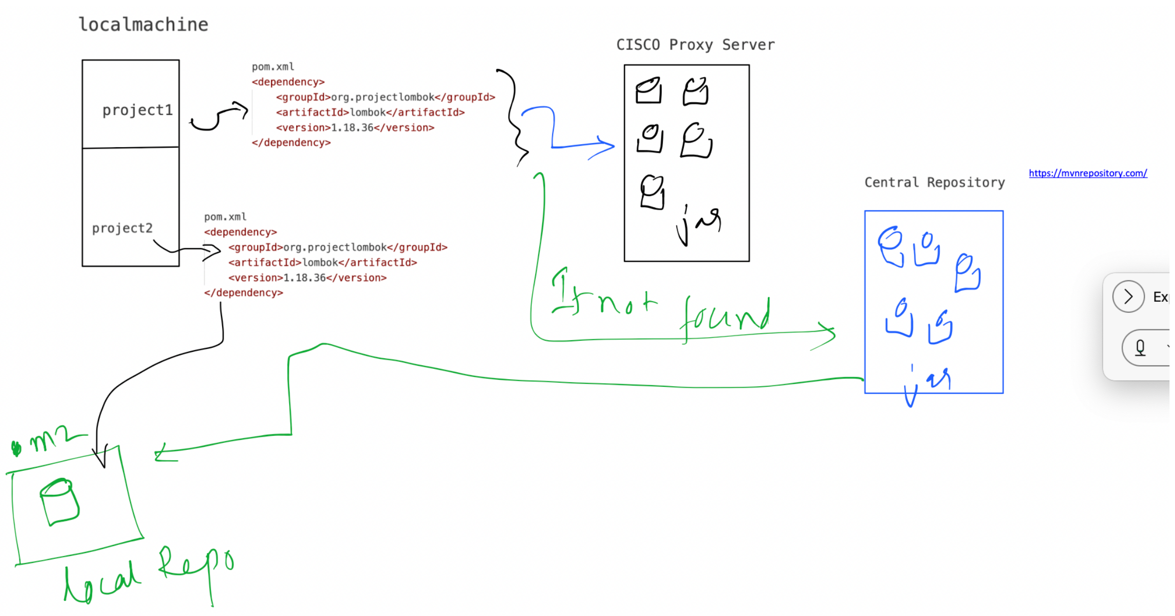
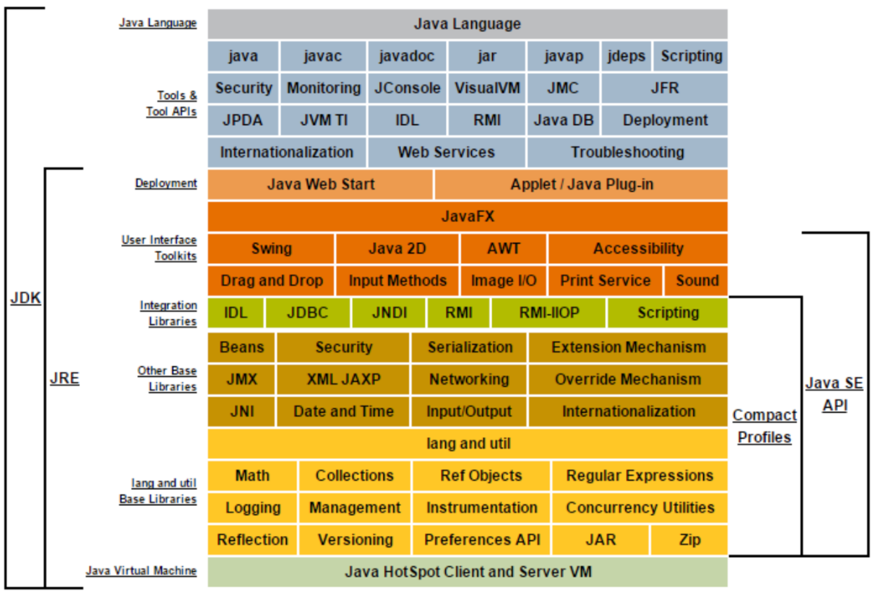
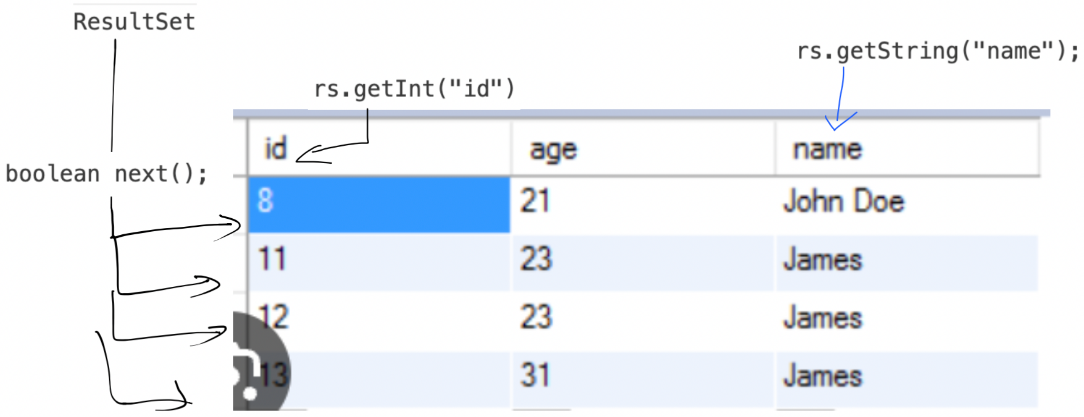


```
pom.xml
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.36</version>
</dependency>
```

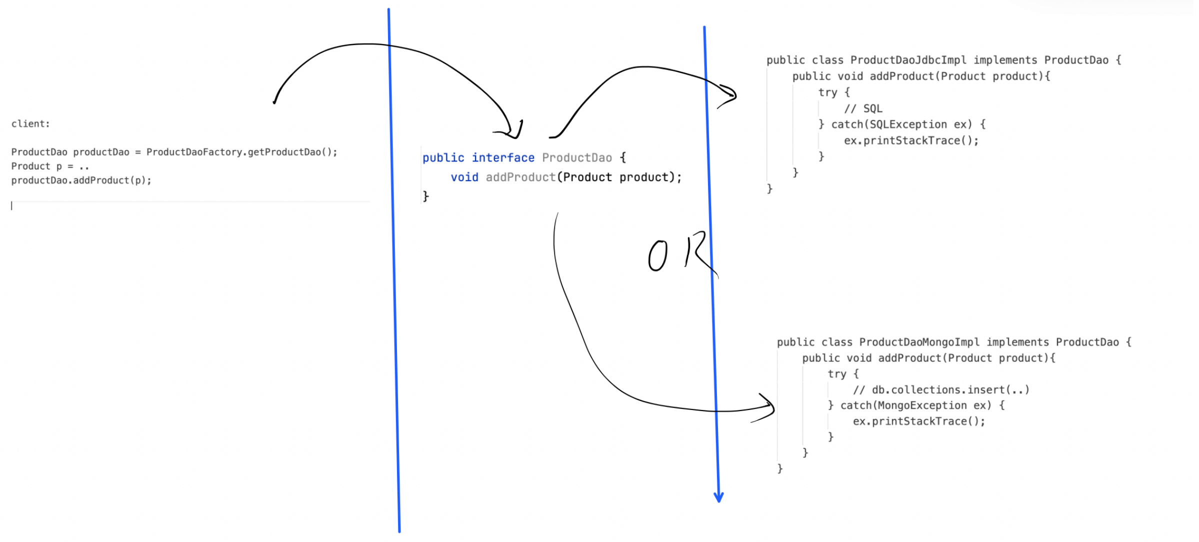
```
pom.xml
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.36</version>
</dependency>
```





id	age	name
8	21	John Doe
11	23	James
12	23	James
13	31	James

Scenario 1:
Exceptions are not propagated to client; client has no clue if exception occurred or not


```
client:

ProductDao productDao = ProductDaoFactory.getProductDao();
Product p = ..
productDao.addProduct(p);
```

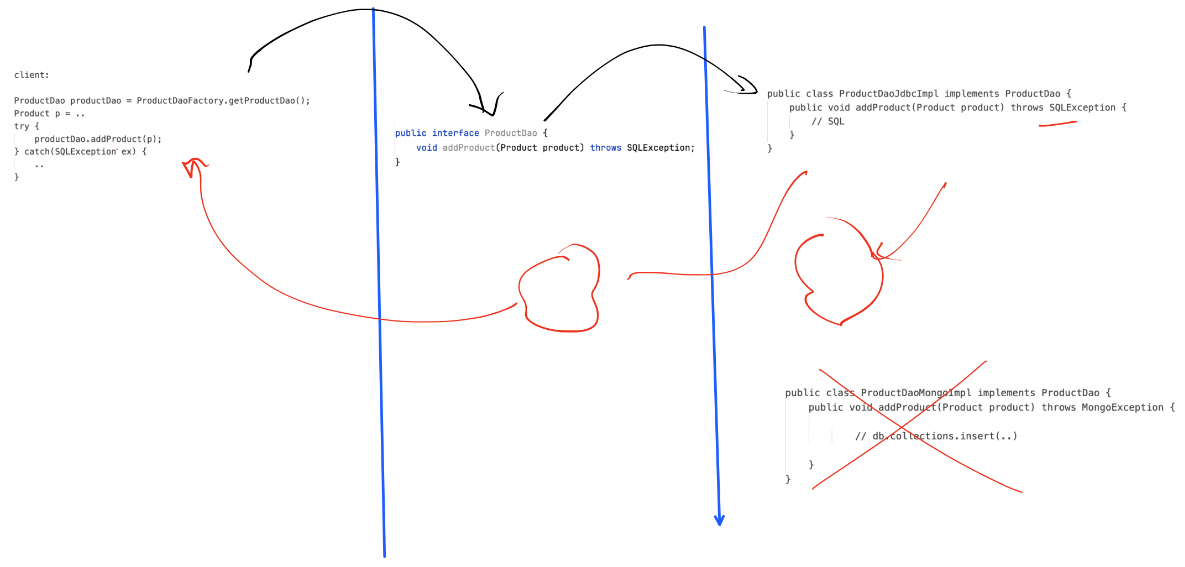
```
public interface ProductDao {
    void addProduct(Product product);
}
```

```
public class ProductDaoJdbcImpl implements ProductDao {
    public void addProduct(Product product){
        try {
            // SQL
        } catch(SQLException ex) {
            ex.printStackTrace();
        }
    }
}
```

```
public class ProductDaoMongoImpl implements ProductDao {
    public void addProduct(Product product){
        try {
            // db.collections.insert(..)
        } catch(MongoException ex) {
            ex.printStackTrace();
        }
    }
}
```

Scenario 2:
Exceptions are propagated to client.

- Tight coupling
- No abstraction; why should we expose the implementation type to client
- Exception messages are not understandable to client [ORA9001; unique constraint exception..]


```
client:

ProductDao productDao = ProductDaoFactory.getProductDao();
Product p = ..
try {
    productDao.addProduct(p);
} catch(SQLException ex) {
    ..
}
```

```
public interface ProductDao {
    void addProduct(Product product) throws SQLException;
}
```

```
public class ProductDaoJdbcImpl implements ProductDao {
    public void addProduct(Product product) throws SQLException {
        // SQL
    }
}
```

```
~~public class ProductDaoMongoImpl implements ProductDao {~~
~~    public void addProduct(Product product) throws MongoException {~~

~~            // db.collections.insert(..)~~

~~    }~~
~~}~~
```

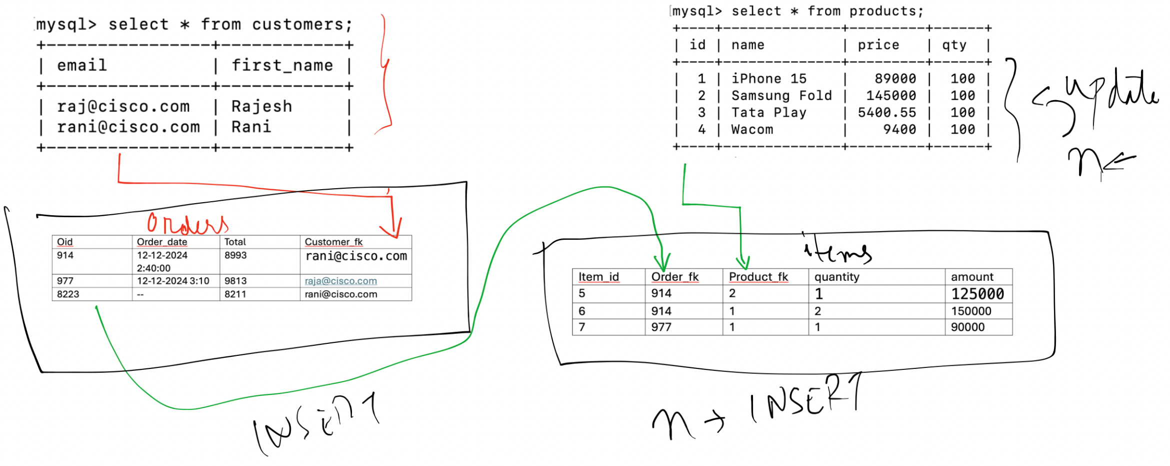
Scenario 3:
Using CustomExceptions

```
ProductDao productDao = ProductDaoFactory.getProductDao();
Product p = ..
try {
    productDao.addProduct(p);
} catch(PersistenceException ex) {
    ..
}
```

```
public interface ProductDao {
    void addProduct(Product product) throws PersistenceException;
}
```

```
public class ProductDaoJdbcImpl implements ProductDao {
    public void addProduct(Product product) throws PersistenceException {
        try {
        // SQL
        } catch(SQLException ex) {
            log
            throw new PersistenceException(message);
        }
    }
}


public class ProductDaoMongoImpl implements ProductDao {
    public void addProduct(Product product) throws PersistenceException {
        try {
            // db.collections.insert(..)
        } catch(MongoException ex) {
            log
            throw new PersistenceException(message);
        }
    }
}
```



```
mysql> select * from customers;
+-----------------+--------------+
| email           | first_name |
+-----------------+--------------+
| raj@cisco.com   | Rajesh     |
| rani@cisco.com  | Rani       |
+-----------------+--------------+
```

```
mysql> select * from products;
+----+----------------+----------+------+
| id | name           | price    | qty  |
+----+----------------+----------+------+
|  1 | iPhone 15      |    89000 |  100 |
|  2 | Samsung Fold   |   145000 |  100 |
|  3 | Tata Play      |  5400.55 |  100 |
|  4 | Wacom          |     9400 |  100 |
+----+----------------+----------+------+
```

Oid	Order_date	Total	Customer_fk
914	12-12-2024 2:40:00	8993	rani@cisco.com
977	12-12-2024 3:10	9813	raja@cisco.com
8223	--	8211	rani@cisco.com

Item_id	Order_fk	Product_fk	quantity	amount
5	914	2	1	125000
6	914	1	2	150000
7	977	1	1	90000