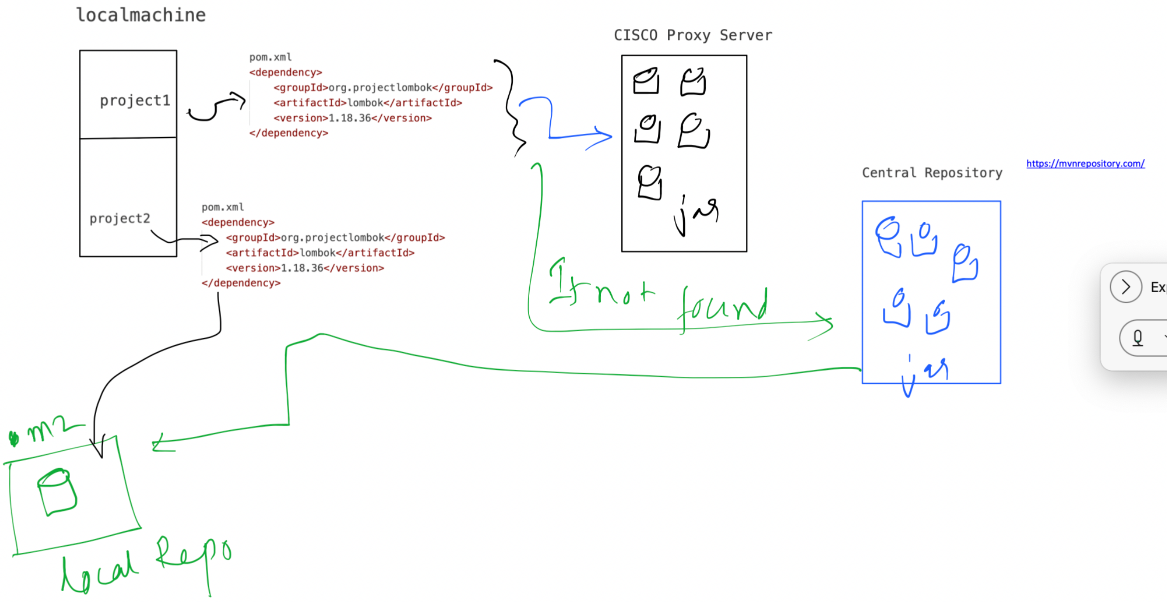
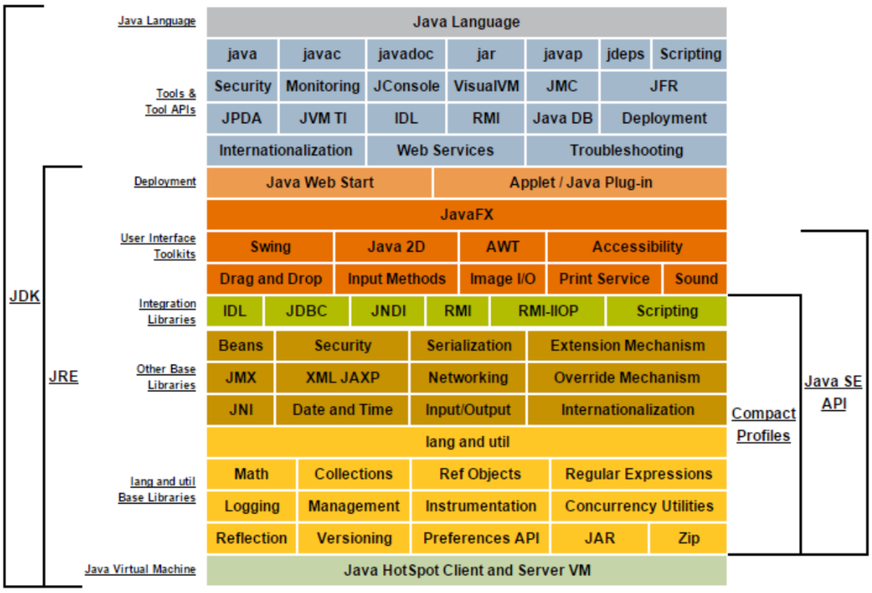
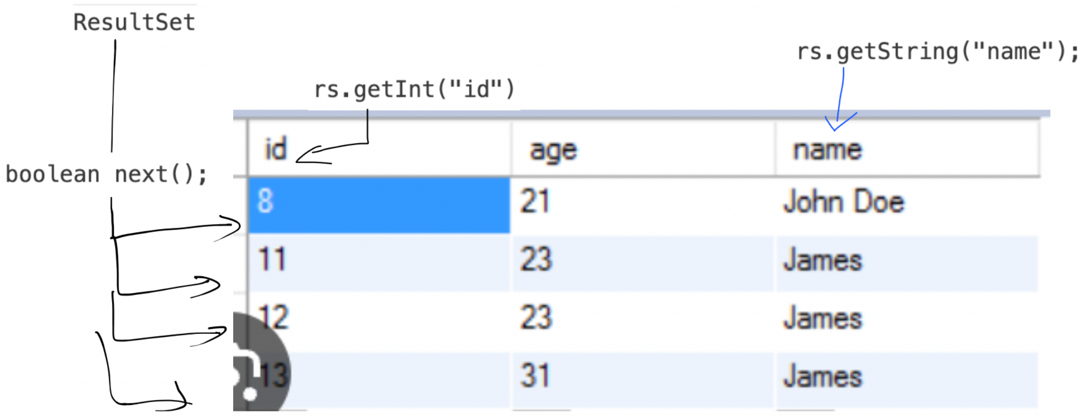


```
pom.xml
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.36</version>
</dependency>
```

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

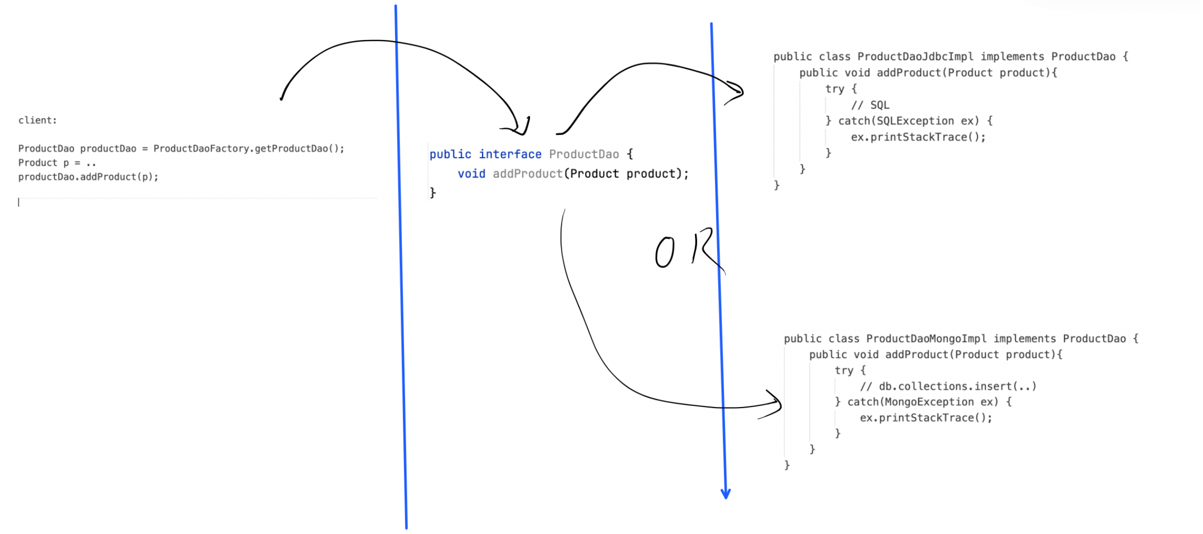


Java Language	Java Language						
Tools & Tool APIs	java	javac	javadoc	jar	javap	jdeps	Scripting
	Security	Monitoring	JConsole	VisualVM	JMC	JFR	
	JPDA	JVM TI	IDL	RMI	Java DB	Deployment	
	Internationalization	Web Services	Troubleshooting				
Deployment	Java Web Start	Applet / Java Plug-in					
User Interface Toolkits	JavaFX						
	Swing	Java 2D	AWT	Accessibility			
	Drag and Drop	Input Methods	Image I/O	Print Service	Sound		
Integration Libraries	IDL	JDBC	JNDI	RMI	RMI-IIOP	Scripting	
Other Base Libraries	Beans	Security	Serialization	Extension Mechanism			
	JMX	XML JAXP	Networking	Override Mechanism			
	JNI	Date and Time	Input/Output	Internationalization			
lang and util Base Libraries	lang and util						
	Math	Collections	Ref Objects	Regular Expressions			
	Logging	Management	Instrumentation	Concurrency Utilities			
	Reflection	Versioning	Preferences API	JAR	Zip		
Java Virtual Machine	Java HotSpot Client and Server VM						



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);
```

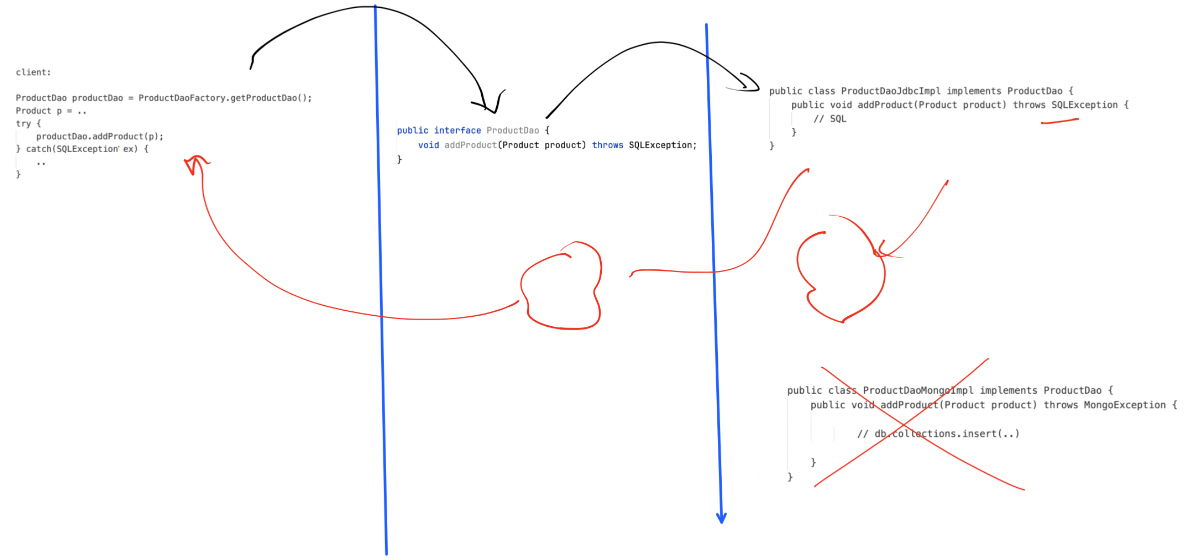
```
public interface ProductDao {
    void addProduct(Product product);
}
```

```
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..]



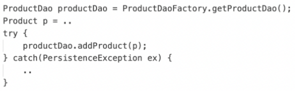
```
client:

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

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

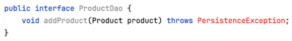
```
public class ProductDaoJdbcImpl implements ProductDao {
    public void addProduct(Product product) throws SQLException {
        // SQL
    }
}
```

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

            // db.collections.insert(..)

    }
}
```

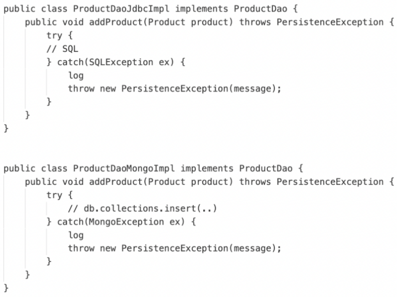
Scenario 3:
Using CustomExceptions



```
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);
        }
    }
}
```