

۲	فصل ۱) JDBC [اتصال پایگاه داده در جاوا]
۲	۱.۱) مفاهیم اولیه
۲	۱.۱.۱) دسته بندی کلی در دیتابیس ها:
۲	۱.۱.۲) انواع دیتابیس ها
۳	۱.۲) توضیحات JDBC
۳	۱.۲.۱) Auto Commit
۴	۱.۳) JDBC Drivers
۴	۱.۳.۱) JDBC Methods
۵	۱.۴) Prepared Statement
۶	۱.۵) MYSQL
۷	فصل ۲) Apache Maven [مدیریت پروژه استاندارد محور]
۷	۲.۱) استاندارد سازی دایرکتوری ها و فایل ها
۸	۲.۱.۱) نمونه اولیه یا Archetype
۸	۲.۲) مدیریت وابستگی ها
۹	۲.۲.۱) مشخصات یک وابستگی
۹	۲.۲.۲) Transitive Dependency
۹	۲.۲.۳) Dependency Scope
۱۰	۲.۳) ساخت پروژه (Build)
۱۰	۲.۳.۱) ایجاد پروژه تحت قواعد «maven»
۱۱	۲.۳.۲) فایل pom.xml
۱۱	۲.۴) Life Cycle And Phases
۱۲	۲.۵) افزونه ها (Plugins)
۱۳	۲.۶) دستورات maven
۱۳	فصل ۳) Internationalize یا Dynamic Data Code
۱۳	۳.۱) روش اول: کلاس Properties
۱۴	۳.۲) روش دوم: Resource Bundle
۱۴	فصل ۴) Design Pattern
۱۵	۴.۱) الگوی طراحی Singleton
۱۵	۴.۱.۱) طراحی
۱۶	۴.۱.۲) Singleton Class
۱۷	۴.۱.۳) Lazy singleton Class
۱۷	۴.۱.۴) Thread safe singleton Class
۱۹	۴.۲) الگوی طراحی Factory
۲۰	۴.۲.۱) طراحی
۲۰	۴.۲.۲) مثال

فصل ۱) JDBC [اتصال پایگاه داده در جاوا]

فریم‌ورک JDBC که مخفف عبارت Java Database Connectivity است، یکی از مکانیزم‌های اتصال به پایگاه داده در زبان جاوا است.

۱.۱) مفاهیم اولیه

Database: مجموعه دیتا

DBMS: مخفف عبارت Data Base Management System می‌باشد و به نرم‌افزاری می‌گویند که کار آن نگهداری و ساماندهی دیتا می‌باشد

SQL: مخفف عبارت Structured Query Language می‌باشد. یک زبان ارتباط با دیتای ذخیره شده در نرم‌افزار DBMS است

DBA: مخفف عبارت Data Base Admin است. شخصی که مسئول مدیریت یک DBMS می‌باشد

SQL Language: زبان SQL ذاتاً بدلیل افزایش Performance خیلی سبک است پس شامل مواردی نظیر if و دیگر موارد کلیدی نیست. به همین دلیل شرکت‌هایی که DBMS ارائه دادند ماژول Custom شده خود را برای افزایش امکانات بیشتر ارائه دادند که در ماکروسافت TSQL و در اوراکل PLSQL است

table: در برنامه DBMS برای هر پروژه یک دیتابیس ایجاد می‌کنیم و هر موجودیت برنامه خود را داخل یک جدول که حاوی سطر و ستون است، قرار می‌دهیم

Schema: اگر یک سیستم اداری متمرکز دارای چندین بخش باشد [مثلاً بانک: مشتریان، کارمندان، تراکنش‌ها، کارت‌ها، سپرده‌ها] برای تمییز بین جداول بخش‌ها، از نوعی دسته‌بندی تحت عنوان اسکیمای استفاده می‌شود. اسکیمای شامل: Table، View، Function، Store Procedure و غیره است

Catalog: جهت دسته‌بندی یا گروه‌بندی، می‌توان چند اسکیمای را در داخل یک کاتالوگ قرار داد [استاندارد اس کیوال ۹۲]

۱.۱.۱) دسته‌بندی کلی در دیتابیس‌ها:

- **Relational Database**: دیتابیس‌های رابطه‌ای مثل: MySQL، Oracle، PostgreSQL، MariaDB
- **NoSQL**: همانند مونگو که تقریباً تمام دیتابیس‌های از این نوع عنوان Transaction را حذف کردند
- **Emdbded**: حالت سروری نداشته و خیلی سبک است بطوری که اطلاعات روی فایل نگهداری می‌شود ولی همان حالت سروری را در ظاهر حفظ کرده است مثل SQLLight مثل برنامه کروم و فایرفاکس که از آن استفاده می‌کند
- **FileBase**: دیتابیس در فایل ذخیره می‌شود همانند SQLLight
- **MemoryBase**: دیتابیس در روی حافظه موقت ذخیره می‌شود همانند H۲ یا Hypersonic Database یا derby که برای جاوا است

۱.۱.۲) انواع دیتابیس‌ها

MySQL: در ابتدا برای شرکت سان بود که وقتی اوراکل آن را خرید، مقداری پولی شد و جامعه اپن سورس یک نسخه دیگر بنام MariaDB راه‌اندازی کرد که کاملاً اپن سورس است

SQLServer: دیتابیس ماکروسافت که زبان ساختاریافته خود را TSQL نامگذاری کرده‌است

MongoDB: نحوه نگهداری داده‌ها Collection است که براساس Document Base یعنی json است

SQLLight: دیتابیس بر پایه فایل است. برنامه Db Browser یک برنامه برای SQLLight است که کارهای آن را می‌توان انجام داد.

: Oracle

- دارای زبان مختص خود بنام PLSQL که باید همانند یک برنامه نصب شود
- نگهداری سخت و حتماً یک دیتابیس ادمین می‌خواهد که ماین‌تورینگ داشته باشد و بررسی شود

- نسخه Educational که تحت عنوان SE از آن نام برده می‌شود (مثلا نسخه ۱۱g)
- نسخه Enterprise که محیط کاملا واقعی مثل ایرانسل و بیمه ایران دارد از آن استفاده می‌کند
- **Table Space**: یک جدول به لحاظ حجم فیزیکی چقدر قابلیت استفاده از هارد را داشته باشد
- در اوراکل یوزر همان اسکیم است و جداول و غیره زیرمجموعه یوزر یا همان اسکیم است
- نکته: در هنگام نصب نرم‌افزار PLSQL اوراکل آن را به هیچ عنوان در مسیر ProgramFile X86 نصب نکنید زیرا عمل نمیکند

۱.۲ توضیحات JDBC

- **JDBC**: یک API برای اتصال از طریق جاوا به دیتابیس فارغ از نوع سیستم عامل
- یک تکنولوژی فراگیر در جاوا برای اتصال به پایگاه‌های داده از نوع Relational است
- نحوه اتصال توسط API ها تعیین شده است [API های آماده شده توسط شرکت‌های DBMS که در اختیار برنامه‌نویس‌ها قرار گرفته است]
 - API شامل موارد: ۱- اتصال به پایگاه داده ۲- واکنشی اطلاعات ۳- بروزرسانی اطلاعات ۴- فراخوانی توابع و Procedure ها ۵- استفاده از ODBC برای اتصال
- روش اتصال به هر نوع دیتابیس یکسان است ولی تمام ملاحظات توسط ماژول از پیش تعیین شده کنترل می‌شود.
- برای هر نوع دیتابیس (اوراکل یا مای‌اس‌کیوال و غیره) باید یک jar همراه داشته باشیم که باید بعنوان driver آن پایگاه داده در سیستم نصب شود

مراحل اتصال به پایگاه داده با جاوا

۱. بارگذاری درایور
۲. ایجاد یک اتصال (Connection): یک یوآرال شامل نام و آی پی و پورت نام کاربری و پسورد
۳. DML که می‌تواند یکی از موارد Select یا Update یا Delete باشد
 ۱. ارسال دستور واکنشی اطلاعات و بازیابی اطلاعات
 ۲. ارسال دستورات تغییر (ایجاد، حذف، بروزرسانی)
 ۳. فراخوانی دستورات و خواندن خروجی آن‌ها

۱.۲.۱ Auto Commit

- بطور پیش فرض نوع ایجاد کانکشن‌ها در هنگام اتصال از طریق JDBC، از نوع auto Commit است. یعنی به ازای هر DML بصورت خودکار آن را در دیتابیس کامیت می‌کند
- برای غیرفعال‌سازی از عبارت زیر را به بلاک تعریف Connection اضافه نمایید

```
connection.setAutoCommit(false)
```

- اگر خط بالا را به بلاک تعریف کانکشن اضافه نمایید، آنگاه باید به ازای هر عملیات از نوع Delete یا Update حتما کامیت یا رول‌بک انجام بگیرد وگرنه در دیتابیس منظور نخواهد شد.

```
dbconnection.getConnection().commit()
dbconnection.getConnection().rolBack()
```

JDBC Drivers (۱.۳)

پیرو ODBC که مخفف OpenDataBaseConnectivity است و امکان اتصال به پایگاه داده‌های مختلف را فراهم می‌کند و هدف آن استقلال اتصال فارغ از نوع سیستم عامل بود (توسط ماکروسافت ارائه شده بود) JDBC نیز همانند آن است و مخفف JavaDatabaseConnectivity است با این تفاوت که منحصر برای زبان برنامه نویسی جاوا است. با توجه به توضیحات قبل هرگاه بخواهیم به یک دیتابیس از نوع Relational متصل شویم باید فایل متفاوت JAR آن نوع دیتابیس را به پروژه بیافزاییم.

`Class.forName[DRIVER_NAME]`

MicrosoftODBC

DriverClass: sun.jdbc.odbc.jdbcOdbcDriver
URL: jdbc:odbc:

IDB

DriverClass: jdbc.idbDriver
URL: jdbc:idb:

Oracle

DriverClass: oracle.jdbc.Driver.OracleDriver
URL: jdbc:oracle:thin:@Server:Port:/dbname

Example1:

```
Class.forName("oracle.jdbc.Driver.OracleDriver");  
String url="jdbc:oracle:thin:@10.0.20.88:1521/XE"  
Connection conn=DriverManager.getConnection(url,"UserName","Password");
```

Postgre

DriverClass: postgresql.Driver
URL: jdbc:postgres://host/database

H2

DriverClass: org.h2.Driver
URL: jdbc:h2:mem:<DB Name>

Mysql

DriverClass: com.mysql.jdbc.Driver
URL: jdbc:mysql://host/database

JDBC Methods(۱.۳.۱)

JDBC method	SQL type	Java type
getBit()	BIT	boolean
getByte()	TINYINT	byte
getShort()	SMALLINT	short
getInt()	INTEGER	int
getLong()	BIGINT	long
getFloat()	REAL	float
getDouble()	DOUBLE	double
getString()	CHAR	String
getString()	VARCHAR	String
getString()	LONGVARCHAR	String
getDate()	DATE	java.sql.Date
getTimeStamp()	TIME	java.sql.Date
getObject()	BLOB	Object

Prepared Statement (۱.۴)

عنوان SQL Injection یکی از انواع باگهایی است که در مبحث پایگاه داده مطرح است و شرکتهای پشتیبان برای جلوگیری از این اتفاق Prepared Statement را معرفی کردند. در این روش دیتای متغیر در یک Query بطور مثال نام کاربری که قرار است در دیتابیس ذخیره شود، توسط API به Query اضافه می شود و در نتیجه نفوذپذیری پایگاه داده کاهش می یابد. در این روش هر مکانی در رشته کوئری که نیاز به تکمیل توسط دیتای متغیر باشد توسط یک علامت سؤال نگارش می شود و در انتهای کد، یک به یک علامت سوالها توسط توابع در اختیار قرار داده شده برای برنامه نویسان تکمیل خواهد شد.

در زیر یک نمونه از این کد در حالتی که نوع کوئری از جنس update است را خواهید دید:

```
java.sql.connection
connection=driverManager.getConnection("jdbc:mysql://host:۳۳۰۶/database","USERNAME","PASSWORD");
PreparedStatement statement=connection.prepareStatement("update table set email = ? where id= ?");
statement.setString(۱,"behroozmn@chmail.ir");
statement.setLong(۲,۱۰۰L);
```

اگر کوئری از نوع Select ساده باشد از دستور **executeUpdate** استفاده می کنیم

```
ResultSet resultSet=statement.executeUpdate("select * from TABLE");
```

و در انتها کانکشن را ایجاد می کنیم

```
private static DBConnection dbConnection=new DBConnection();
```

MYSQL (۱.۵)

```
java.sql.connection connection=driverManager.getConnection ( \
    "jdbc:mysql://host:3306/database", "USERNAME", "PASSWORD");
Statement statement=connection.createStatement();
ResultSet resultSet=statement.executeQuery("select * from TABLE");
while(resultSet.next()){
    System.Out.Println(resultSet.getString("FirstName"));
    # همچنین می‌شود از ایندکس استفاده کرد با شمارشگر شروع شونده یک یعنی بجای فیرست‌نیم از عدد یک استفاده کرد
    System.Out.Println(resultSet.getString("LastName"));
    System.Out.Println(resultSet.getString("StudentID"));
}
private static DBConnection dbConnection=new DBConnection();
```

همچنین

```
#execute
statement.executeQuery("select from ..."); # → For SELECT
statement.executeUpdate("update table set ..."); # → For UPDATE or INSERT
# → مقدار اینتیجر یعنی تعداد سطرهایی که در دیتابیس تأثیر پذیرفته‌اند را برمی‌گرداند
#get
resultSet.getInt("ستونی که مقدار اینتیجر برمی‌گرداند");
resultSet.getFloat("ستونی که مقدار اعشار برمی‌گرداند");
resultSet.getLong("ستونی که مقدار لانگ برمی‌گرداند");
resultSet.getDate("ستونی که مقدار زمان تاریخ برمی‌گرداند");
resultSet.getTimestamp("ستونی که مقدار زمان ساعت برمی‌گرداند");
```

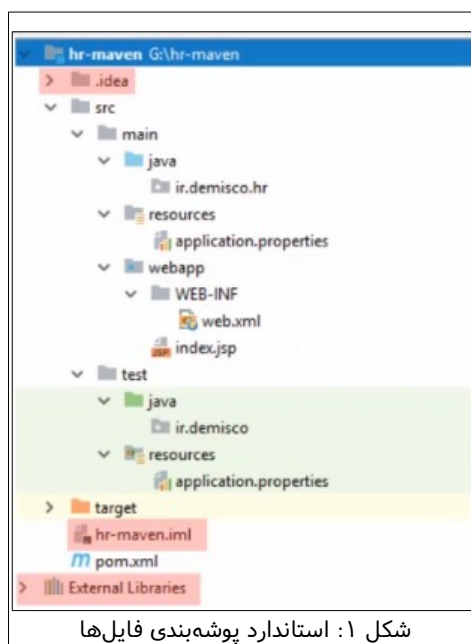
فصل ۲) Apache Maven [مدیریت پروژه استاندارد محور]

- کمک‌رسانی در ساخت و تست و گزارش‌گیری و پکیج‌کردن پروژه‌های جاوا
- ورژن اول آن در سال ۲۰۰۴ منتشر شد.
- فایل‌های با پسوند «IDE» و «IML» و بخش‌های «External Library» منحصرًا برای IDE می‌باشد و بعنوان فایل پروژه ذخیره نمی‌شود و اصلاً نیاز به کپی ندارد

۲.۱) استانداردسازی دایرکتوری‌ها و فایل‌ها

دارای ساختار استاندارد برای پوشه‌بندی و چیدمان فایل‌هایی نظیر: کدها، تست‌ها و فایل‌های تنظیماتی است. در هنگام ساخت پروژه جدید هر IDE، مطابق سلیقه خود عمل کرده و هر فایل را در مسیر دلخواه خویش قرار می‌دهد و این امر سبب اختلال هنگام مهاجرت از یک IDE به دیگری (مثلاً از IntelliJ به Eclipse) می‌شود. پس هنگام ساخت پروژه جدید باید آن را به Apache Maven بسپاریم (اثری از New project نیست) تا ساختاربندی پروژه، مطابق استاندارد شناخته شده توسط تمامی IDE‌های رایج (نظیر Eclipse یا IntelliJ یا NetBeans) انجام شود تا در هنگام Migrate از یک IDE به دیگری، پروژه دچار اختلال نشود.

- سبب یکسان شدن ساختار پروژه‌های متفاوت در IDE‌های متفاوت می‌شود (عدم سردرگمی برنامه‌نویسان در خروج و ورود به پروژه‌ها)
- در این مدل، مکان تمامی فایل‌ها و مسیرها استاندارد می‌شود.
- بطور مثال تعیین شده که فایل‌های سورس یا خروجی یا کلاس‌ها یا غیره در چه مسیری قرار بگیرند



در این شیوه مجموعه دیتا در دایرکتوری تحت عنوان «src» قرار دارد و داخل آن دو دایرکتوری تحت عناوین «main» و «test» وجود دارد

- دایرکتوری «src»:
 - دایرکتوری main: پروژه اصلی در این فولدر است
 - دایرکتوری «java»: سورس‌های کد جاوا داریم در این فولدر است. ممکن است سورس‌های برنامه توسط یک «JVM Language» دیگر نظیر «Groovy» در پوشه مربوط به خود نوشته شود. پس کلمه جاوا در این مسیر گنجانده شده است. البته در زمان «Run Time» این موارد با یکدیگر یکپارچه شده و نکته خاصی در پی نخواهد داشت
 - دایرکتوری «Resources»: ریسورس از نوع فایل و عکس و غیره داریم در اینجا قرار می‌دهیم
 - دایرکتوری «webapp»: پوشه‌های وب در این مسیر قرار خواهند گرفت
 - دایرکتوری test: تمامی تست‌های واحد «Unit Test» در اینجا قرار دارد
 - دایرکتوری «java»: سورس‌های کد تست‌های جاوا داریم در این فولدر است
 - دایرکتوری «Resources»: ریسورس تستها از نوع فایل و عکس و غیره داریم در اینجا قرار می‌دهیم
- دایرکتوری «Target»: شامل فایل‌های خروجی و کامپایل شده‌ها است. خروجی فایل‌ها یعنی پسوند class در این مسیر قرار دارد و دیگر خبری از دایرکتوری out وجود ندارد. همچنین package ها نیز در این مسیر قرار خواهند گرفت. مثلاً خروجی war را در این فولدر قرار خواهد داد.

۲.۱.۱) نمونه اولیه یا Archetype

- سری تمپلیت از قبل تعریف شده برای نوع فریم‌ورک‌های مختلف با تنظیمات متفاوت که بتواند توسط این الگوها ساختارهای رایج را متمایز نماید.
 - مثلاً پروژه‌های وب نیاز به دایرکتوری «webapp» می‌باشند
- برای پروژه‌های نوع متفاوت باید از archetype های متفاوت استفاده نمود.
- هنگام ایجاد پروژه ساده از نوع maven بصورت پیش فرض از archetype ساده اولیه خود استفاده می‌کند که شامل دایرکتوری‌های «src» و «main» و «java» است. ولی این ساختار با ورود فریم‌ورک‌های متفاوت کمی پیچیده تر شده و می‌بایست دایرکتوری‌های تنظیماتی و غیره به پروژه ملحق شود

۲.۲) مدیریت وابستگی‌ها

- اکثر پروژه‌ها برای کامپایل و اجرا به فریم‌ورک‌ها یا کتابخانه‌های دیگر احتیاج دارند و دانلود کردن Jar فایل‌ها بصورت دستی و نگهداری آن و همچنین کنترل ورژن‌های آن بصورت دستی دشوار بوده و نیاز به مکانیزمی برای خودکارسازی این امر می‌باشد
- برای مدیریت ورژن‌های برنامه و وابستگی‌ها، آن‌ها را درون فایل بنام pom.xml قرار می‌دهیم
 - «maven» برای مدیریت وابستگی‌های از Repository استفاده می‌کند.
 - کتابخانه‌هایی که در مسیر «pom.xml» معرفی می‌شوند از یک «Remote Repository» دانلود می‌شوند و درون یک «Local Repository» درون سیستم خود برنامه‌نویس ذخیره می‌شوند.
 - درمگانیزم «maven» وابستگی‌های dependency ها را نیز بصورت خودکار دانلود خواهند شد.
 - ریپوزیتوری «maven central» شامل همه فایل‌های استاندارد و تأیید شده است و در دسترس همگان قرار دارد

- ریپوزیتوری‌های ریموت زیادی وجود دارند که مثلاً فریم‌ورک spring بصورت مستقل «Repository» مربوط به خودش را دارد
- امکان این وجود دارد که در فایل «pom.xml» چندین ریپوزیتوری از نوع «mirror» تعریف نماییم. یعنی بگوییم پیش‌فرض فلان ریپوزیتوری باشد و اگر داخل آن نبود سرآغ ریپوزیتوری دیگر
- توصیه می‌شود یک «maven» مرکزی داخل شرکت قرار داده شود که به آن می‌گویند «Dependency Management» که ابتدا برنامه اول در ریپوزیتوری داخل شرکت جستجو کرده و اگر نبود سرآغ ریپوزیتوری ریموت اینترنتی می‌رود.
- توسط برنامه «Nexus» قابلیت ایجاد یک ریپوزیتوری لوکال در شرکت فراهم می‌شود.

۲.۲.۱) مشخصات یک وابستگی

- Group ID: نمایانگر این است که این پروژه محصول چه شرکتی یا گروهی یا سازمان مسئول پروژه مثل: log4j یا org.hibernate یا org.springframework
- artifact ID: مشخصه پایین‌تر از group Id است و نمایانگر اسم محصول یا چیزی که قرار است نوشته شود مثل: hibernate-tools یا spring-core
- Version: ورژن محصول
 - به ورژن‌های پایین‌تر که دستخوش تغییرات زیاد است «Major Version» گفته می‌شود
 - به ورژن‌های بالاتر که تغییرات کمتری صورت می‌گیرد و در آن باگ‌ها فیکس می‌شود «Minor» گفته می‌شود
 - به ورژن‌های بعدی که تغییرات ناچیز است «Patch» گفته می‌شود
- type: چه Package از محصول مورد نیاز است مثل: EAR یا WAR یا JAR
 - گاهی پیش می‌آید که به یک War وابستگی پیش بیاید و در این حالت پیچیدگی پیش می‌آید. در این صورت محتوای وب با تمامی جزئیات در پروژه کپی خواهد شد

۲.۲.۲) Transitive Dependency

- ممکن است یک dependency خود به artifact های دیگری dependency داشته باشد به این حالت Transitive Dependency گفته می‌شود.
- مثلاً hibernate به jbossLogging و dom4j و javaassist و غیره وابستگی دارد.
 - «maven» می‌تواند وابستگی‌های بین اجزا را تا آخرین سطح مدیریت نماید.
 - هر وابستگی که در فولدر مخفی m2 موجود در مسیر `home\username` است حاوی یک فایل pom.xml است که داخل آن هم می‌توان برخی وابستگی‌های بسته دانلود شده را قرار داد.

۲.۲.۳) Dependency Scope

- می‌توان به «maven» توضیح داد که یک وابستگی در چه زمانی مورد نیاز باشد.
- «Compile» (پیش‌فرض): در تمامی مراحل یعنی کامپایل، تست، ساخت و اجرا این وابستگی در Class Path وجود داشته باشد.

- «Provided»: فقط در زمان کامپایل و تست این بسته در اختیار قرار بگیرد و اضافه شود
 - در مرحله ساخت محصول و اجرا نیاز به در اختیار قرار گرفتن و اضافه شدن نیست
- «Runtime»: فقط در زمان اجرا استفاده شود و در زمان کامپایل نیاز نیست
- «Test»: فقط برای اجرای تست‌ها مثل JUnit زیرا فقط زمان تست استفاده می‌شود
- «System» (منسوخ شده است): شبیه Provided عمل می‌کند با این تفاوت که آن را از File System لود می‌کند
- «Import»: صرفاً برای استفاده از تنظیمات یک pom فایل دیگر استفاده می‌شود
 - اضافه شده از «maven» ورژن ۳ به بعد
- بعدی

۲.۳ ساخت پروژه (Build)

- در گذشته Apache Ant توسط یک فایل XML تمام کارهایی که باید انجام شود را بصورت خودکار انجام می‌داد و خودکار پروژه Build می‌شد
- طریقه‌ای یکسان برای «Build» پروژه‌ها را فراهم می‌آورد تا برنامه‌نویسان به آسانی بتوانند فارغ از نوع پروژه و در هر زمان و مکان توسط مجموعه دستورات اقدام به کامپایل و تولید خروجی در پروژه نمایند
- مکانیزم کامپایل در IDE ها و همچنین Apache Ant استفاده از Javac است.
- کار Apache Maven به مراتب فراگیرتر از apache Ant است. و تنها کار Build کردن نیست و کارهای بیشتر انجام می‌دهد
- در پروژه‌های عادی IDE ها (بعنوان مثال IntelliJ) امر Build کردن را برعهده می‌گرفتند ولی در این روش build کردن را به IDE نخواهیم سپرد.
- فریم‌ورک maven توسط لایه‌ای که بر JAVAC نوشته است کارهای بیشتری انجام می‌دهد.

۲.۳.۱ ایجاد پروژه تحت قواعد «maven»

- هر پروژه maven باید حاوی سه ویژگی باشد
 - اول: group ID که نمایانگر این است که این پروژه محصول چه شرکتی یا گروهی است

```
DgroupId=com.demisco
DgroupId=com.Behrooz
```

- دوم: Artifact ID که نمایانگر اسم محصول یا چیزی که قرار است نوشته شود

```
DartifactId=UI
DartifactId=Backend
```

- سوم: ورژن پروژه
 - معمولاً کلمه اسنپ‌شات گذاشته می‌شود به معنی این است که هنوز Release نشده است

0.0.1-snapshot

- نمونه Command کامل برای ایجاد یک پروژه ساده:

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetype -DgroupId=com.behroozco -DartifactId=Behrooz
```

- معمولاً IDE ها نیاز به نصب «maven» ندارند و نسخه «maven» را درون خود دارند. پس نیازی به زدن دستور بالا نیست

۲.۳.۲ فایل pom.xml

فایل «pom.xml» یکی از فایل‌های مهم پروژه است که حاوی توضیحات و تنظیمات Build پروژه و نام و ورژن و غیره است

- هرکجا pom.xml را مشاهده کردید یعنی این پروژه توسط maven ایجاد شده است
- برخی موارد موجود در فایل به شرح زیر است:
 - model Version : این عدد مخصوص خود «maven» است
 - group ID : نمایانگر این است که این پروژه محصول چه شرکتی یا گروهی است
 - artifactID : نمایانگر اسم محصول یا چیزی که قرار است نوشته شود [هم برای پروژه مورد استفاده است] [هم برای کسی که بخواهد از پروژه ما بعنوان dependency استفاده نماید که در مبحث Transitive Dependency از آن صحبت شده است]
 - packaging : نوع خروجی که در دایکتوری Target ایجاد خواهد شد
 - version : ورژن نسخه که قرار است Build شود
 - name : نام خاص که توسط برنامه نویس داده شده است
 - URL : آدرس سایت خودمان یا داکيومنت‌های پروژه
 - dependencies و dependency : شامل group id و artifact id و ورژن و scope برای هر وابستگی

۲.۴ Life Cycle And Phases

این نام گذاری برای مراحل است که طی آن یک پروژه Build می‌شود. در حالت کلی و در نظر عامه یک برنامه شامل سه بخش است:

- 1-Develop: به فاز برنامه نویسی پروژه توسط کدنویس می‌گویند
- 2-Test: به فاز تست کدهای نوشته شده توسط برنامه نویس و تیم تست می‌گویند
- 3-Deploy: به استقرار یک کد نوشته شده (محصول) در محیط علمياتی یا آزمایشی می‌گویند

چرخه حیات یک کد نوشته شده ممکن است شامل Goal های متفاوت باشد (مثلاً packaging ممکن است شامل ۲۰ Goal باشد که هرکدام به تبع کار خاص باید انجام شود) مجموعه Goal هایی که پشت سر هم اجرا شوند یک life Cycle را بوجود می‌آورند اما ممکن است یک goal خاص را فقط بخواهیم اجرا بگیریم

بصورت پیش‌فرض در «maven» سه فاز قرار داده شده است که شامل موارد زیر است:

- Clean : پاک کردن فایل‌های خروجی و موقت (نظیر پسوندهای Class) و غیره برای تولید فایل‌های خروجی جدید
 - Default : شامل مراحل زیر است
 - Validate: چک شدن صحت پروژه از نظر وابستگی‌های مد نظر و فایل‌های تنظیماتی
 - Compile : کامپایل (تبدیل فایل‌های باینری) کدهای نوشته شده
 - Test : تست کدهای نوشته شده توسط ابزارهای تست
 - packaging : ایجاد فایل خروجی Jar یا war در مسیر صحیح (مسیر target)
 - Install: نصب پکیج در Local Repository برای اینکه همه بتوانند کدهای نوشته شده را ببینند
 - Deploy : ارسال پکیج در remote Repository
 - Site : ساخت Documentation ها و مستقر کردن پروژه
- نکته: اگر یک چرخه را نام ببریم یعنی همه Goal های آن چرخه را انجام بده ولی اگر تنها نام goal خاص را استفاده نماییم یعنی تنها از یک کار خاص می‌خواهیم استفاده نماییم. آدرس [URL](#) لیست این موارد را نمایش خواهد داد

۲.۵ افزونه‌ها (Plugins)

ساختار maven از یک معماری plugin محور تبعیت می‌کند یعنی این پلاگین‌ها می‌توانند وظایف متفاوتی را انجام دهند. tools برای کارهایی مثل:

- کامپایل کردن
- پاک کردن خروجی
- ساخت نوع فایل‌های war یا ear یا jar
- تولید گزارش
- که لیست این پلاگین‌ها را در مسیر [URL](#) می‌توان مشاهده کرد
- طریقه استفاده از یک پلاگین به این صورت است که نام پلاگین به همراه goal آن را ذکر می‌کنیم (goal: هر پلاگین ممکن است در چرخه حیات ساخت پروژه چند کار انجام دهد و هر کاری که انجام می‌دهد را تحت عنوان یک goal اسم می‌بریم)

```
mvn PluginIdentifier:GoalIdentifier
mvn clean:clean
mvn compile:compile //Goal: compile
mvn compile:testCompile //Goal: test-compile
mvn resources:resources //Goal: process-resources
mvn resources:testResources //Goal: process-test-resources
mvn surefire:test //Goal: test
mvn jar:jar //Goal: package
mvn install:install //Goal: install
mvn deploy:deploy //Goal: deploy
```

۲.۶ دستورات maven

- دستور mvn کامند «maven» است که برای استفاده آن را در PATH تنظیم نمایید.
- فریم‌ورک «maven» شامل برخی دستورات است که در زیر به بیان برخی می‌پردازیم
- دستورات «Build» یک پروژه توسط زیر انجام می‌شود. کارهای بیلد را انجام می‌دهد: ۱- کامپایل کردن ۲- اجرای تست‌ها ۳- ساخت ایل خروجی نظیر «JAR» فایل

```
mvn package
```

- دستور پاک کردن دایرکتوری «Target»

```
mvn clean:clean
```

فصل ۳ Internationalize یا Dynamic Data Code

بهتر است که برخی رشته‌هایی که بصورت شفاف در کد استفاده می‌شود را بصورت داینامیک در کد استفاده نماییم که برای این کار از Resource Bundle یا کلاس Properties استفاده می‌شود.

۳.۱ روش اول: کلاس Properties

- این روش بگونه ای استفاده از نوع map است زیرا یک فرزند از کلاس map است. ولی یک نوع map که هم مقدار و هم کلید آن از نوع رشته است
- یک فایل بنام setting.txt (نام مهم نیست) در مسیر Resources (مسیر مهم نیست) می‌سازیم و محتوی آن را با عبارات متناظر پر می‌کنیم

```
db.url=jdbc:oracle:thin@127.0.0.1/XE
db.username=behrooz
db.password=salamhavij

private Properties proper=new Properties();
InputStream settingFileInputStream =
this.getClass().getClassLoader().getResource("com/setting/setting.txt").openStream();
proper.load(settingFileInputStream);

System.out.println(proper.getProperty("مقدار کلید"));
```

- بهروز: نیازمند تکمیل شدن است

۳.۲ روش دوم: Resource Bundle

برای استفاده از کلاس ResourceBundle استفاده می‌شود. مثلاً برای یک کلید دو نام قرار بدهید بطوری که در آن تایم بهش بگید hellow را بخواند یا salam را بخواند و برای نام آن کلید در نظر بگیرید. یک فریم‌ورک است که کدهای نوشته شده به زبان جاوا را Internationalize می‌کند.

فصل ۴ Design Pattern

The Design Patterns are descriptions of communicating objects and class that are customized to solve a general design problem in a particular context.

در شی‌گرایی، استانداردهایی برای تنظیم روابط و رفتار مطلوب‌تر بین آبجکت‌ها با یکدیگر وجود دارد که پیروی از آن‌ها سبب بهینگی کد می‌شود و پس از اعمال آن، در ادامه همگان ملزم به استفاده از این شیوه رفتاری استاندارد در همه‌جای کد خواهند بود. همچنین استفاده از هر الگو منوط به وجود مشکلی است که توسط این الگوی استاندارد قصد مرتفع نمودن آن را خواهیم داشت.

- مبحث الگوهای طراحی یا Design Pattern، پیرو عنوان Object Oriented می‌باشد.
- برخی الگوهای طراحی در برخی زبان‌های برنامه‌نویسی بدون کاربرد است و استفاده نمی‌شود زیرا هدف خاص در زبان برنامه‌نویسی خاص مرتفع گردیده است.
- قابلیت مرتفع شدن برخی مشکلاتی که در طراحی برنامه وجود دارد توسط عنوان Aspect Oriented وجود دارد و لزوماً نیاز به استفاده از Design Pattern نیست
- برخی مواقع اصلاً نیاز به استفاده از یک الگوی طراحی نیست ولی گاهی به اشتباه مورد استفاده قرار می‌گیرد

طبقه‌بندی الگوهای طراحی:

- Creational Patterns : الگوهای طراحی بر مبنای ایجاد و ساخت «آبجکت»
 - Singleton pattern : تنهای یک شیء از یک کلاس ساخته بشود و هر بار شیء ساخته شده را مورد استفاده قرار دهد
 - Factory Method pattern :
 - Abstract Factory pattern :
 - Builder Pattern :
 - Prototype pattern :
- Structural Patterns : الگوهای طراحی بر مبنای «تنظیم روابط آبجکت‌ها» از نوع ترکیب‌سازی آبجکت‌ها با یکدیگر
 - Adapter pattern :
 - Bridge pattern :
 - Composite pattern :
 - Decorator pattern :
 - Facade pattern :
 - Flyweight pattern :
 - Proxy pattern :
- Behavioral Patterns : الگوهای طراحی بر مبنای «تنظیم روابط آبجکت‌ها» از نوع استفاده یک آبجکت در آبجکت دیگر (رفع پیچیدگی)

- : Chain of responsibility
- : Command
- : Interpreter
- : Iterator
- : Mediator
- : Memento
- : Observer
- : State
- : Strategy
- : template
- : Visitor

۴.۱) الگوی طراحی Singleton

- ساخت تنها و تنها یک نمونه از یک شیء: هر بار که یک کلاس را New می‌کنیم آنگاه یک شیء جدید از آن کلاس بوجود می‌آید. در صورتی که بخواهیم با هر بار New کردن شیء جدید ساخته نشود از این روش استفاده می‌کنیم.
- تضمین کنترل منابع: زمانی که محدودیت منابع وجود دارد مثلاً دیتابیس یا پرینتر یا فایل یا هر چیز دیگر که می‌خواهیم مطمئن بشویم که ارتباط با آن ریسورس تماماً با این کلاس صورت می‌گیرد
- این مدل طراحی دارای قاعده ثابت است (عدم انعطاف‌پذیری)
- مثال‌ها:

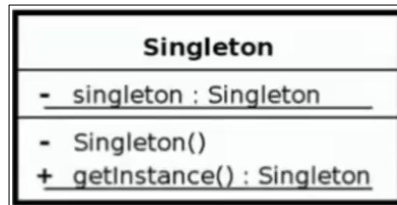
- نمونه‌های رایج در جاوا: Runtime ، Logger
- Spring Beans: در اسپرینگ تمامی Bean هایی که ساخته می‌شوند بصورت پیش فرض از نوع سینگلتون است
- وجود تنها یک پرینتر
- کانکشن به دیتابیس

نکته: هرگاه به این تفکر برخوردید که نیاز به State های متفاوت و Data های متفاوت است باید فکر سینگلتون را از ذهن خارج کرد

۴.۱.۱) طراحی

- تنها با یک آبجکت راه‌اندازی می‌شود و نیاز به چندین آبجکت برای پیاده‌سازی ندارد.
- کلاس سینگلتون به این صورت است که یک Instance از خود کلاس Singleton درون سینگلتون وجود دارد یعنی خود کلاس، خودش را New کرده است. تک آبجکتی که قرار است همه به آن دسترسی پیدا کنند را داخل خود کلاس سینگلتون می‌گذاریم
- تک آبجکت را از نوع Private قرار می‌دهیم: وگرنه ممکن است در خلال کدنویسی Null یا دستکاری کند
- تک آبجکت را از نوع Static قرار می‌دهیم
- دارای تابع Constructor بدون پارامتر است

- برای اینکه کسی نتواند این کلاس را New نماید و آبجکت جدید بسازد:
- متد Constructor آن را در حالت Private قرار می‌دهیم تا به هیچ عنوان قابلیت ساخت آبجکت جدید نداشته باشد، مگر از طریق ساخته شدن داخل خود کلاس آن
- در آخر یک متد نهایی داریم که از نوع Public هست و همه از طریق آن به سینگلتون دسترسی دارند با نامی مثلاً GetInstance



- در قطعه کد زیر اگر هش کد دو نمونه مساوی باشند آنگاه در خروجی تصریح خواهد شد:

```

Runtime firstInstance = Runtime.getRuntime();
firstInstance.gc(); //Garbage Collector //شود اجرا می‌شود
System.out.println(firstInstance);

Runtime anotherInstance = Runtime.getRuntime();
System.out.println(anotherInstance);

if (firstInstance == anotherInstance){
    System.out.println("Two instance are equal");
}

```

Singleton Class (۴.۱.۲)

- قطعه کد زیر یک نمونه از اتصال دیتابیس از نوع سینگلتون است:

```

public class DBConnection {
    private static DBConnection dbconnection = new DBConnection();
    private DBConnection(){}
    public static DBConnection getInstance(){
        return dbConnection;
    }
}

```

توضیحات:

• **خط دوم:**

- متغیر از نوع Static است زیرا باید از کلاس یک نمونه شیء بیشتر ساخته نشود و هرکس خواست از این نمونه استفاده نماید
- متغیر از نوع private است زیرا کسی نتواند این مقدار رو تغییر و مستقیماً از آن استفاده نماید

خط سوم:

- متد Constructor این کلاس ساخته شده و private تعریف شده است تا هیچ کس خارج کلاس نتواند از این تابع سازنده استفاده نماید. عبارتی تابع سازنده آن قابل فراخوانی نیست و چون این کلاس تنها یک تابع سازنده دارد پس کسی نمی تواند از روی آن شی جدید بسازد

خط چهارم:

- متد getInstance تا هرکسی بخواهد نمونه ای از کلاس رو بگیرد شیء که یکبار ساخته شده است را فراخوانی نماید. خروجی بازگشتی از نوع singleton است

در صورت استفاده از سینگلتون باید با دستور زیر شیء تولید کرد

```
Singleton s = Singleton.getInstance();
```

در صورت استفاده از Singleton خط زیر غلط خواهد بود:

```
Singleton s = new Singleton(); // غلط است
```

Lazy singleton Class (4.1.3)

- طبق قاعده جاوا (در بحث Class Loading) اولین Touch از یک کلاس (حتی Import در Junit) سبب Instantiate از تمامی مقادیر استاتیک آن کلاس می شود. پس کلاس سینگلتون حتماً دارای یک نمونه آبجکت می باشد. حالا اگر برنامه بصورت سینگلتون باشد و حتی یک ارتباط با دیتابیس نداشته باشد آنگاه اتقلاf منابع خواهیم داشت (این مثال در برخی منابع ممکن است دارای Cost زیاد باشد) پس می توان قطعه کد بالا بصورت Lazy نگارش شود یعنی هرگاه به شیء نیاز شد آنگاه آبجکت تولید گردد

```
public class DBConnection {
    private static DBConnection dbconnection = null;
    private DBConnection(){}
    public static DBConnection getInstance(){
        if(dbconnection == null){
            dbconnection = new DBConnection();
        }
        return dbConnection;
    }
}
```

Thread safe singleton Class (4.1.4)

- این شرایط وجود دارد که کلاس را بعداً Thread safe نماییم: یعنی اگر برنامه در محیط Concurrent اجرا می شود و چندین Thread همزمان چندین کلاس را Instance نمایند آنگاه سبب بروز مشکل خواهد شد

روش اول

```

public class DBConnection {
    private static DBConnection dbconnection = null();
    private DBConnection(){}
    public synchronized static DBConnection getInstance(){
        if(dbconnection == null){
            dbconnection = new DBConnection();
        }
        return dbConnection;
    }
}

```

روش دوم

```

public class DBConnection {
    private static DBConnection dbconnection = null();
    private DBConnection(){}
    public static DBConnection getInstance(){
        if(dbconnection == null)
            synchronized (DBConnection.class){
                if(dbconnection == null){
                    dbconnection = new DBConnection();
                }
            }
        return dbConnection;
    }
}

```

توضیحات: اگر یک thread داخل محدوده بلوک **Synchronized** قرار داشته باشد آنگاه اگر thread دوم به این بلاک برسد، صبر می کند تا thread اول از این بلاک عبور کند و سپس Thread دوم وارد این بلاک می شود. (پردازه برای دومی قفل می شود و با خروج اولی قفل آن باز می شود)

۴.۲ الگوی طراحی Factory

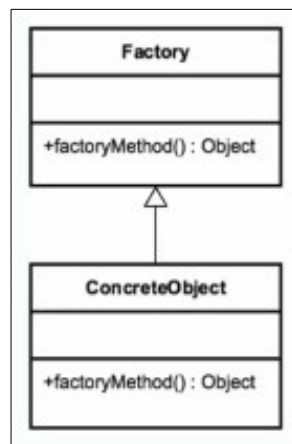
- هدف ایجاد: پنهان سازی پیچیدگی های ساخت شیء (برنامه نویسی درگیر پیچیدگی های آبجکت ها نشود و به سهولت نمونه بسازد)
- این الگوی طراحی بر پایه اصل وراثت بنا نهاده شده (Inheritance)
- یک از کاربردهای این الگوی طراحی برای زمانی است که از سیستم cache استفاده می شود. خصوصاً زمانی که تولید نمونه ها پرهزینه خواهد بود (ارتباط با دیتابیس، پرینتر، اسکنر، دیوایس های External و غیره). در این هنگام می توان نمونه ها را در فضای استاتیک نگهداری کرد و در هر بار فراخوانی فقط از آن استفاده نمود.
- الگوی طراحی **Factory method** نقطه مقابل **Singleton** می باشد
- پیچیدگی ساخت زیر کلاس ها را رفع می کند
- پیاده سازی آن منوط به نوشتن کد بیشتری است (پیچیدگی همراه می آورد ولی در نگاه کلی سبب سهولت است)

مثال

- تبدیل فرمت عکس به فرمت های گوناگون (JPG , PNG , GIF , SVG , غیره) که در اینصورت هر فرمت صاحب کلاس مستقل به همراه پارامترهای منحصر به خود است که ممکن است در فرمت دیگری به چنین پارامتری نیاز نباشد. یعنی کلاس های متفاوت برای هر نوع فرمت تصویر با ورودی های متفاوت به همراه پیچیدگی های آنها
- Number Format: نوع اعداد فارسی یا عربی یا فرمت انگلیسی باشد
- Resource Bundle: ایجاد نمونه متفاوت بر حسب تنظیمات
- Calendar: نوع تقویم جلالی یا میلادی یا هجری قمری یا عبری یا پهلوی یا غیره باشد (قطعه کد زیر که بر حسب منطقه خاص می تواند Locale بپذیرد)

```
import java.util.Calendar;
Calendar x = Calendar.getInstance(Locale.English); #getInstance is Factory
System.out.println(x);
System.out.println(x.get(Calendar.SECOND));
```

- این الگوی طراحی به کد قابلیت گسترش می دهد (در مثال تغییر فرمت تصاویر به یکدیگر می توانیم به سهولت یک فرمت جدید بیافزاییم)



۴.۲.۱ طراحی

- **اصل اول:** «مخفی سازی منطق ساخت اشیاء»
 - عدم ارتباط کاربر با Subclass ها
 - ایجاد یک کلاس جدید بنام «Factory» و پیاده سازی تمام تعاملات کاربر در این کلاس [معمولا انتهای آن کلمه Factory قرار می دهند. ممکن است به دلخواه برنامه نویس این کلمه آورده نشود]
- **اصل دوم:** ساماندهی و بهینه سازی ارتباطات «FactoryClass» با «کلاس های فرزند»
 - پایه گذاری پیچیدگی های برنامه در قالب کلاس های متفاوت
 - مبنای ارتباطات، بر پایه شیوه «ارثبری»
 - تمرکز پیچیدگی های «کلاس های فرزند» در کلاس «FactoryClass»
- **اصل سوم:** «Share Interface» برای استاندارد سازی ارتباطات
 - استفاده از یک **اینترفیس مشترک** برای مدیریت «کلاس های فرزند» با «FactoryClass»
 - متدهای مهم در در «اینترفیس مشترک» پایه گذاری می شود و تمامی «کلاس های فرزند» موظف به پیاده سازی آن متدها خواهند بود
 - مثال: در مسأله تبدیل فرمت تصاویر یک اینترفیس بنام «ImageConvertor» خواهیم داشت که همه «کلاس های فرزند» با این اینترفیس تعامل برقرار خواهند کرد و این اینترفیس یک متد بنام Convert خواهد داشت که اطلاعات را میگیرد و خروجی را برمی گرداند

۴.۲.۲ مثال

- به کلاس زیر و نیز پیاده سازی های آن توجه نمایید

```
public abstract class Calculation {
    protected int amountPerMonth;
    protected int taxPercent;
    protected string product;

    public Calculation(int amountPerMonth, int taxPercent, string product) {
        this.amountPerMonth = amountPerMonth;
        this.taxPercent = taxPercent;
        this.product = product;
    }

    public abstract int calculate();
}
```

پیاده سازی از نوع Calculation :

```
public class Calculation extends Calculation {
    public Calculation(int amountPerMonth, int taxPercent, string product) {
        super(amountPerMonth, taxPercent, product);
    }
    @Override
    public abstract int calculate() {
        return amountPerMonth * taxPercent + ۲۰۰;
    }
}
```

پیاده‌سازی از نوع **Calculationy** :

```
public class Calculationy extends Calculation {  
    public Calculationy(int amountPerMonth, int taxPercent, string product) {  
        super(amountPerMonth, taxPercent, product);  
    }  
    @Override  
    public abstract int calculate() {  
        return amountPerMonth * taxPercent;  
    }  
}
```

پیاده‌سازی **FactoryClass** :

```
public class CalculationFactory {  
    public Calculation createCalculation(int amountPerMonth, int taxPercent, string product, boolean flag) {  
        if (flag) {  
            return new Calculationy(amountPerMonth, taxPercent, product);  
        } else {  
            return new Calculationy(amountPerMonth, taxPercent, product);  
        }  
    }  
}
```