فهرست

فصل ۱) JDBC[اتصال‌پایگاه‌داده درجاوا]

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 DataBase 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 DataBase 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 دراوراکل 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[مثلا بانک: مشتریان، کارمندان، تراکنش‌ها،کارت‌ها،سپرده‌ها] برای تمییز بین جداول بخش‌ها، از نوعی دسته‌بندی تحت عنوان اسکیما استفاده می‌شود. اسکیما شامل: Table، View، Function، Store Procedure و غیره است
Catalog: جهت دسته‌بندی یا گروه‌بندی، می‌توان چند اسکیما را در داخل یک کاتالوگ قرار داد[استاندارد اس کیو ال ۹۲]

۱.۱.۱)دسته‌بندی کلی در دیتابیس‌ها:

- **Relational Database**:
  - دیتابیس‌های رابطه‌ای که بر پایه جدول بنا شده‌اند
  - همانند دیتابیس‌های(نرم‌افزارهای) MySQL، Oracle، PostgreSQL، MariaDB
- **NoSQL** : بر پایه متن دیتا ذخیره می‌کنند(کالکشن)
  - برپایه key و value هستند(همانند دیکشنری‌های پایتون اما با تفاوت عدم استفاده از دابل کوتیشن)
  - تقریباً تمام دیتابیس‌های از این نوع عنوان Transaction را حذف کردند
  - دیتابیس‌های(نرم‌افزارهای) معروف: مونگو و کاساندرا و ...
- **Embded**: 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
  - **RRD**(**B**ound **R**obin **D**atabase): برپایه خلاصه‌سازی (Sumarize) که مثلاً در برنامه Netdata استفاده شده است
- **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 : درابتدا 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 کاملا 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[نیاز اساسی به حافظه رم]
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 حتما یک دیتابیس ادمین می‌خواهد که ماینتورینگ داشته باشد و بررسی شود
- نسخه Educational که تحت عنوانSE از آن نام برده می‌شود(مثلا نسخه g۱۱)
- نسخه Enterprise که محیط کاملا واقعی مثل ایرانسل و بیمه‌ایران 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 اسکیما است و جداول و غیره زیرمجموعه یوزر یا همان اسکیما است
- نکته: درهنگام نصب نرم‌افزار PLSQL اوراکل آن را به هیچ عنوان در مسیر ProgramFile X۸۶ 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 دراختیار برنامه‌نویس‌ها قرار گرفته است]
  - API شامل موارد: ۱-اتصال به پایگاه‌داده ۲-واکشی اطلاعات ۳-بروزرسانی اطلاعات ۴-فراخوانی توابع و Procedureها ۵-استفاده از ODBC برای اتصال
- روش اتصال به هرنوع 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. بارگزاری 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 مواردSelect یا Update یا Delete باشد
    ۱. ارسال دستور واکشی اطلاعات و بازیابی اطلاعات
    ۲. ارسال دستورات تغییر(ایجاد، حذف، بروزرسانی)
    ۳. فراخوانی دستورات و خواند خروجی آن‌ها

۱.۲.۱) Auto Commit

- بطور پیش‌فرض 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 مخففOpenDataBaseConnectivity است و امکان اتصال به پایگاه دادهای مختلف را فراهم می‌کرد و هدف آن استقلال اتصال فارغ از نوع سیستم عامل بود(وتوسط ماکروسافت ارائه شده بود) JDBC نیز همانند آن است و مخفف JavaDatabaseConnectivity است با این تفاوت که منحصرا برای زبان برنامه نویسی جاوا است
باتوجه 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

۱.۴ Prepared Statement

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 می‌شود و درنتیجه نفوذپذیری پایگاه داده کاهش می‌یابد. در این روش هر مکانی در رشته کوئری که نیاز به تکمیل توسط دیتای متغیر باشد توسط یک علامت سؤال نگارش می‌شود و در انتهای کد، یک به یک علامت سوال‌ها توسط توابع دراختیار 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=driverManaager.getConnection("jdbc:mysql://host:۳۳۰۶/database","USERNAME","PASWORD");
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 می‌کنیم

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

و در انتها کانکشن را ایجاد می‌کنیم

```
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 فیرست‌نِیم 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 تاثیرپذیرفته‌اند 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 زمان‌تاریخ برمی‌گرداند");
    resultSet.getTimeStamp("ستونی که مقدار زمان‌ساعت برمی‌گرداند");
```

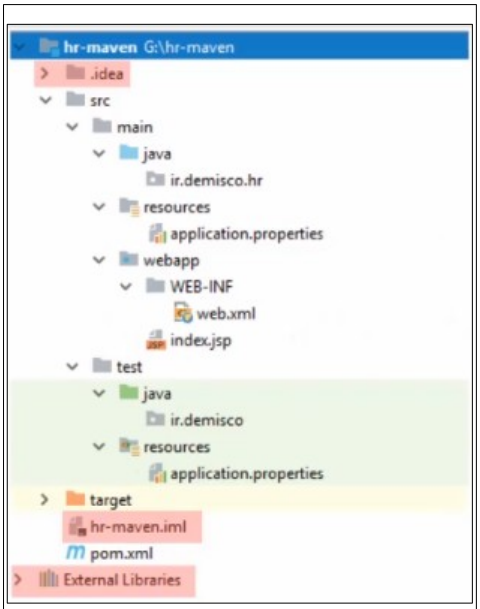
فصل ۲) Apache Maven [مدیریت پروژه استاندارد محور]

- 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» منحصرا 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 هرفایل 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 IDE های رایج(نظیر Eclipse یا IntelliJ یا NetBeans) انجام شود تا درهنگام Migrate از یک IDE به دیگری، پروژه دچار اختلال نشود.

- سبب یکسان شدن ساختار پروژه‌های متفاوت در IDE های 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



شکل ۱: استاندارد پوشه‌بندی فایل‌ها

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)نمونه‌اولیه 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 پروژه های نوع متفاوت باید از archetype های متفاوت استفاده نمود.
- هنگام ایجاد پروژه ساده از نوع maven بصورت پیش‌فرض از archetype ساده اولیه خود استفاده می‌کند که شامل دایرکتوری‌های « src» و «main» و «java » است. ولی این ساختار با ورود فریم‌ورک‌های متفاوت کمی پیچیده‌تر 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 فایلی بنام pom.xml قرار می‌دهیم
- «maven» برای مدیریت وابستگی‌های از Repository استفاده می‌کند.
- کتابخانه‌هایی که درمسیر «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 برود سراغ رپوزیتوری 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 سراغ 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: log۴j یا org.springframework یا org.hibernate
- artifact ID: مشخه 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 ورژن های بالاتر که تغییرات کمتری صورت می‌گیرد و در آن باگ‌ها فیکس می‌شود «Minor» گفته می‌شود
  - به ورژن های 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 جزییات 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 dom۴j و javaassist و غیره وابستگی دارد.
- «maven» می‌تواند وابستگی‌های بین اجزا را تا آخرین سطح مدیریت نماید.
- هر وابستگی که در فولدر مخفی m۲ 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 دراختیار قرار بگیرد و اضافه شود

- در مرحله ساخت محصول و اجرا نیاز به دراختیار 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» ورژن۳ 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 داکیومنت‌های 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 مراحلی است که طی آن یک پروژه Build می‌شود. در حالت کلی و در نظر عامه یک برنامه شامل سه بخش است:
۱-Develop: به فاز برنامه نویسی پروژه توسط کدنویس می‌گویند
۲-Test: به فاز تست کدهای نوشته شده توسط برنامه نویس و تیم تست می‌گویند
۳-Deploy: به استقرار یک کد نوشته شده(محصول)در محیط علمیاتی یا آزمایشی می‌گویند

چرخه حیات یک کد نوشته شده ممکن است شامل Goal های متفاوت باشد(مثلا 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 کارخاص 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:

- کامپایل‌کردن
- پاک‌کردن 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 دستور است که در زیر به بیان برخی می‌پردازیم

- دستورات «Build» یک پروژه توسط زیر انجام می‌شود. کارهای بیلد را انجام می‌دهد: ۱-کامپایل‌کردن 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) روش‌اول: 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.getPropertiey("مقدار کلید"));
```

- بهروز: نیازمند تکمیل شدن است

۳.۲) روش‌دوم: Resource Bundle

برای استفاده از کلاس ResourceBundle استفاده می‌شود. مثلا برای یک کلید دو نام قرار بدهید بطوری که در ران تایم بهش بگید hellow را بخواند یا salam را بخواند و برای نام آن کلید در نظر بگیرد.
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 رفتارمطلوب‌تر 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 برنامه نویسی بدون کاربرد است و استفاده نمی‌شود زیرا هدف خاص در زبان برنامه نویسی 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 برمبنای ایجاد و ساخت «آبجکت»
  - Singleton pattern : تنهای یک شیء از یک کلاس ساخته بشود و هربار شیء ساخته شده را مورد استفاده قرار دهد
  - Factory Method pattern : پنهان‌سازی پیچیدگی‌های ساخت شیء برپایه وراثت(البته نیاز به نوشتن کد بیشتری دارد)
  - Abstract Factory pattern : همانند FactoryMethodpattern بگونه Factory والد و Factory فرزند(داینامیک‌سازی کلاس فرزند) پیچیدگی زیاد کلاس‌ها را هنگام ایجاد شیء تسهیل می‌دهد. مناسب FrameWork نویسی زیرا پیچیدگی‌ها مرتفع می‌گردد
  - Builder Pattern : هنگام تولید آبجکت با تعددا پارامتر زیاد کاربرد دارد تا کارها و اقدام‌ها کاهش یابد
  - Prototype pattern : اشیاء جدید توسط کپی از شیء موجود[بجای ایجاد شیء جدید از طریق توابع سازنده (Constructor)]
- **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** : الگوهای‌طراحی برمبنای «تنظیم روابط‌آبجکت‌ها» 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) الگوی‌طراحی Singleton

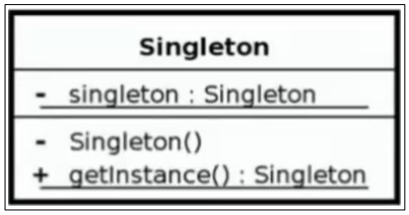
- **ساخت تنها و تنها یک نمونه از یک شیء**: هربار که یک‌کلاس را New می‌کنیم آنگاه یک شیء جدید از آن کلاس بوجود می‌آید. درصورتی که بخواهیم با هربار New کردن شیء جدید ساخته نشود از این روش استفاده می‌کنیم.
- **تضمین کنترل منابع**: زمانی که محدودیت منابع وجود دارد مثلاً دیتابیس یا پرینتر یا فایل یا هرچیز دیگر که می‌خواهیم مطمئن بشویم که ارتباط با آن ریسورس تماماً با این کلاس صورت می‌گیرد
- این مدل طراحی دارای قاعده ثابت است(عدم انعطاف‌پذیر)
- مثال‌ها:
  - نمونه‌های رایج در جاوا: Logger ، Runtime
  - Spring Beans:در اسپرینگ تمامیBean هایی که ساخته می‌شوند بصورت پیش‌فرض از نوع سینگلتون است
  - وجود تنها یک پرینتر
  - کانکشن به دیتابیس

نکته: هرگاه به این تفکر برخوردید که نیاز به Stateهای متفاوت و Dataهای 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 هش‌کد دو نمونه مساوی باشند آنگاه در خروجی تصریح خواهد شد:

```
Runtime firstInstance = Runtime.getRuntime();
firstinstance.gc(); //Garbage Collector // این موضوع فقط نمایش است وگرنه بصورت پیش‌فرض توسط جاوا 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:

- **خط‌دوم:**
  - 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 هیچ‌کس خارج کلاس نتواند از این تابع سازنده استفاده نماید. بعبارتی تابع سازنده آن قابل فراخوانی نیست و چون این کلاس تنها یک تابع سازنده دارد پس کسی نمی‌تواند از روی آن شی جدید بسازد
- **خط چهارم:**
  - متد getInstance تا هرکسی بخواهد نمونه‌ای 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
```

۴.۱.۳ Lazy singleton Class

- طبق قاعده جاوا(در بحث Class Loading) اولین Touch از یک کلاس(حتی Import در Junit) سبب Instantiate از تمامی مقادیر استاتیک آن کلاس می‌شود. پس کلاس سینگلتون حتماً دارای یک نمونه آبجکت می‌باشد.حالا اگر برنامه بصورت سینگلتون باشد و حتی یک ارتباط با دیتابیس نداشته باشد آنگاه اتقلاف منابع خواهیم داشت(این مثال در برخی منابع ممکن است دارای Cost زیاد باشد) پس می‌توان 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;
   }
}
```

۴.۱.۴ Thread safe singleton Class

- این شرایط وجود دارد که کلاس را بعداً Thread safe نماییم: یعنی اگر برنامه در محیط Concurrent اجرا می‌شودوچندین 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;
   }
}
```

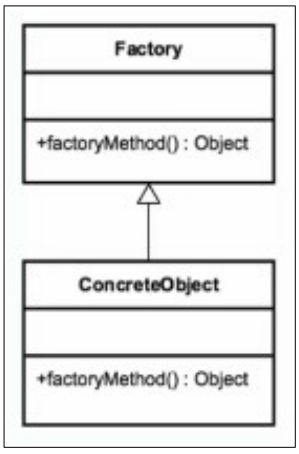
توضیحات: اگر یک thread داخل محدوده بلوک Synchronized قرار داشته باشد آنگاه اگر thread دوم به این بلاک برسد، صبر می‌کند تا thread اول از این بلاک عبور کند و سپس Thread دوم وارد این بلاک می‌شود.(پردازه برای دومی قفل می‌شود و با خروج اولی قفل آن باز می‌شود)

۴.۲) الگوی‌طراحی Factory

- **هدف‌ایجاد:**پنهان‌سازی پیچیدگی‌های ساخت شیء(برنامه‌نویس درگیر پیچیدگی‌های آبجکت‌ها نشود و به سهولت نمونه بسازد)
- این الگوی طراحی برپایه اصل وراثت بنا نهاده شده(Inheritance)
- یک از کاربردهای این الگوی طراحی برای زمانی است که از سیستم cache استفاده می‌شود.خصوصا زمانی که تولید نمونه‌ها پرهزینه خواهد بود(ارتباط با دیتابیس، پرینتر، اسکنر،دیوایس‌های External وغیره). در این هنگام می‌توان نمونه‌ها را در فضای استاتیک نگهداری کرد و در هربار فراخوانی فقط از آن استفاده نمود.
- الگوی طراحی **Factory method** نقطه مقابل **Singleton** می‌باشد
- پیچیدگی ساخت زیرکلاس‌ها را رفع می‌کند
- پیاده‌سازی آن منوط به نوشتن کد بیشتری است(پیچیدگی همراه می‌آورد ولی در نگاه کلی سبب سهولت است)
  - **مثال**
    - تبدیل فرمت عکس به فرمت‌های گوناگون(SVG , GIF , PNG , JPG ، غیره) که در اینصورت هر فرمت صاحب کلاس مستقل به‌همراه پارامترهای منحصر به خود است که ممکن است در فرمت دیگری به چنین پارامتری نیاز نباشد. یعنی کلاس‌های متفاوت برای هر نوع فرمت تصویر با ورودی‌های متفاوت به همراه پیچیدگی‌های آن‌ها
    - Number Format: نوع اعداد فارسی یا عربی یا فرمت انگلیسی باشد
    - Resource Bundle: ایجاد نمونه متفاوت برحسب تنظیمات
    - Calendar: نوع تقویم جلالی یا میلادی یا هجری‌قمری یا عبری یا پهلوی یا غیره باشد(قطعه‌کد زیر که برحسب منطقه خاص می‌تواند 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 می‌دهد(در مثال تغییر فرمت تصاویر به یکدیگر می‌توانیم به سهولت یک فرمت جدید بیافزاییم)



۴.۲.۱)طراحی

- اصل‌اول: «مخفی‌سازی منطق ساخت اشیاء»
  - عدم ارتباط کاربر با Subclass ها
  - ایجاد یک کلاس جدید بنام «Factory» و پیاده‌سازی تمام تعاملات کاربر در این کلاس[معمولا انتهای آن کلمه Factory قرار می‌دهند. ممکن است به دلخواه برنامه‌نویس این کلمه آورده نشود]
- اصل‌دوم: ساماندهی و بهینه‌سازی ارتباطات «FactoryClass» با «کلاس‌های فرزند»
  - پایه‌گذاری پیچیدگی‌های برنامه در قالب کلاس‌های متفاوت
  - مبنای ارتباطات، برپایه شیوه «ارث‌بری»
  - تمرکز پیچیدگی‌های «کلاس‌های فرزند» در کلاس «FactoryClass»
- اصل‌سوم: «share Interface» برای استاندارد سازی ارتباطات
  - استفاده از یک اینترفیس مشترک برای مدیریت «کلاس‌های فرزند» با «FactoryClass»
  - متدهای مهم در در «اینترفیس‌مشترک»پایه‌گذاری می‌شود و تمامی «کلاس‌های فرزند» موظف به پیاده‌سازی آن متدها خواهند بود
  - مثال: در مسأله تبدیل فرمت‌تصاویر یک اینترفیس بنام «ImageConvertor» خواهیم داشت که همه «کلاس‌های فرزند» با این اینترفیس تعامل برقرار خواهند کرد و این اینترفیس یک متد بنام Convert خواهد داشت که اطلاعات را میگیرد و خروجی را برمی‌گرداند

۴.۲.۲)مثال

- به کلاس زیر و نیز پیاده‌سازی‌های 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();
}
```

پیاده‌سازی از نوع Calculation۱ :

```
public class Calculation۱ extends Calculation {
  public Calculation۱(int amountPerMonth, int taxPercent, string product) {
    super(amountPerMonth, taxPercent, product);
  }
  @Override
  public abstract int calculate() {
    return amountPerMonth * taxPercent + ۲۰۰;
  }
}
```

پیاده‌سازی از نوع Calculation۲ :

```
public class Calculation۲ extends Calculation {
  public Calculation۲(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 Calculation۱(amountPerMonth, taxPercent, product);
    } else {
      return new Calculation۲(amountPerMonth, taxPercent, product);
    }
  }
}
```

۴.۳) الگوی‌طراحی Abstract Factory

- کارخانه‌ای که خودش کارخانه تولید می‌کند. یعنی Factory والد و Factory فرزند که این به خودی خود دارای پیچیدگی خواهد شد.
- کاربرد در سیستم‌های بزرگ و آبجکت‌های سنگین که بخواهند ساخت کلاس فرزند را dynamic کنند.
- وجود interfaceهای مشترک از ۲ گروه **الف:** به ازای هر Factory **ب:**به ازای هر کلاس‌هایی که داخل Factory است
  - مثال کلاس **Document Builder**: برای Parse کردن فایلXML که یک آبجکت Node بصورت درختی برمی‌گرداند که می‌توان به تمامی المنت‌هایxml مورد نظر دسترسی پیدا کرد

۴.۳.۱)طراحی

- گروهی از factory ها دارای interface مشترک خواهند بودو با هم استفاده می‌شوند
- abstractFactoryها گروهی از Factory ها هستند که همواره برای ساخته شدن آن‌ها باید ابتدا از یک Factory شروع کرد و سپس به abstractFactory رسید.
- پیچیدگی در پیاده‌سازی
- نیاز به abstraction های زیاد

- الگویی مناسب برای framework ها محسوب می‌شود.(Framework نویس‌ها)

۴.۳.۲)مثال اول DocumentBuilderFactory

```
DocumentBuilderFactory abstractFactory = DocumentBuilderFActory.newInstance();
DocumentBuilder documentBuilder = abstractFactory.newDocumentBuilder();
Document document = documentBuilder.pars(new ByteArrayInputStream(
           “<person><firstName>Behrooz</firstName><lastName>MohammadiNasab</lastName></person>”
             .getBytes(“UTF-8”)));
document.normalizeDocument();
System.out.println(documentBuilder.getClass());
System.out.println(document.getClass());
```

۴.۳.۳)مثال دوم MediaConverterFactory

```
MediaConverterFactory abstraactFactory =
          MediaConverterAbstractFactory.createFactory(Converter.Type.IMAGE);
try{
      Converter converter = abstractFactory.createConverter
               (new File(“/FileName/Directions/pic1.bmp”),Converter.CodeTypes.JPG);
      System.out.println(abstractFactory.getClass());
      System.out.println(converter.getClass());
      byte[] bytes = converter.doConvert();
} catch (FileNotFountExeption | ConvertionExeption e) { e.printStachTrace(); }
```

```
public interface MediaConverterAbstractFactory {
   static MediaConverterFactory createFactory(Converter.Type type) {
      switch (type) {
         case AUDIO:
            return new MusicConverterFactory();
         case VIDEO:
            return new VideoConverterFactory();
         case Image:
            return new ImageConverterFactory();
      }
      throw new OllegalArgumentsException(“Wrong Converter Type”);
   }
}
```

در ادامه

```
public class ImageConverterFactory implements MediaConverterFactory {
   public Converter createConverter(File file, Converter.CodecTypes toImageType)
      throws FileNotFoundException {
         String name = file.getName().toLowerCase();
         if (name.endWith(“.bmp”)){
            switch (toImageType){
               case JPG:
                  return new BmpToJpgConverter(file);
               // AND MORE
            }
         }
         throw new IllegalArgumentException(“No Converter Found”);
   }
}
```

که کار آن این است که در حالت‌های موسیقی و ویدئو و عکس بتواند فرمت‌های متفاوت را تبدیل نماید.

۴.۴) الگوی‌طراحی Builder

این «الگوی‌طراحی» هنگام **تولید آبجکت با تعدادا پارامتر زیاد** کاربرد دارد. همچنین ساخت آبجکت cost زیاد دارد

هدف‌ایجاد: تسهیل مقداردهی پارامترهای زیاد هنگام ساخت کلاس بصورت یکجا

مثال: هنگام ایجاد یک کلاس QueryBuilder برای SQL که نیازمند تعدا پارامترهای زیاد نظیر موارد زیر می‌باشد:

۱. تعداد اجزای Selection

۲. دریافت تک تک عبارت‌های شرطی که بعنوان where استفاده خواهد شد یا همانWhere clause ها

۳. GroupBy ها

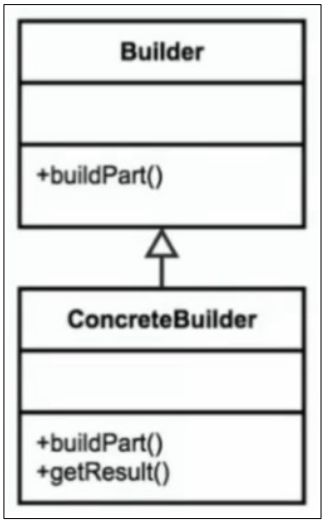
۴. OrderBy ها و ...

در این حالت تعداد پارامترهای زیاد را می‌توان به روش سازنده کلاس یا Constructor دریافت نماییم یا به‌ازای هرپارامتر new کرده و در ادامه آن مقدار دهی نماییم.

در جاوا کلاس‌هایی از جمله DocumentBuilder وStringBuilder و Locale.Builder یا JsonBuilder وجود دارد که در آن از این شیوه استفاده شده است

۴.۴.۱)طراحی

- انعطاف‌پذیری در مقابل constructor ها
- استفاده از innerclass ها
- متدهایی برای جایگزینی Setter ها
- ایجاد کلاس برای Build کردن
- متدهایی تحت عناوین مثلاً build یا getResult ایجاد نماییم تا بعنوان ارائه دهنده خروجی نهایی یا آبجکت نهایی عمل نماید
- معمولاً اسم Builder را به انتهای کلاس می‌افزایند
- پیشنهاد می‌شود کلاس اصلی را بصورت innerClass درون کلاس Builder تعریف نمود تا پیچیدگی کاهش یابد



۴.۴.۲)مثال StringBuilder در جاوا

کلاس StringBuilder (موجود در Java.lang) قابلیت افزودن دیتا به یک رشته را به گونه‌ای دارد که بعنوان رشته اصلی عمل کرده و هر بار دیتای جدید مستقیماً با آن اضافه می‌شود و نیازبه ساخت شیء string جدید بعنوان subString نیست تا آن شیء را به رشته اصلی(شیء اصلی) append نماییم

```
StringBuilder builder = new StringBuilder();
string result = builder.append("Hello, I am").append(33).append("years old").toString();
```

۴.۴.۳)مثال H۲۶۴PropertiesBuilder

۴.۴.۳.۱)بدون Builder

فرض کنید کلاس decoder فرمت H۲۶۲ را بخواهیم پیاده‌سازی نماییم آنگاه بدلیل وجود پارامترهای زیاد، در حالت بدون Builder به شکل زیر می‌باشد(۱-کلاس سازنده با پارامتر زیاد ۲-getter برای هرکدام ۳-setter برای هرکدام)

```
public class H263Properties{
    int keyInt;
    int minKeyInt;
    int sceneCut;
    int bFrames
    int bAdabt
    int qp
```

```
    int bitrate
    boolean bFrameBias
    int crf
    int qpstep
    int pbRatio
    int chromaOffset
    float rateTol
    byte pass
    boolean state
    int direct
    int meRange
    boolean weightB
    boolean noFastPSkip

    تابع سازنده#
    public  H263Properties(int keyInt, int minKeyInt, int sceneCut,int bFrames,int bAdabt, int qp, int
bitrate, boolean bFrameBias, int crf, int qpstep, int pbRatio, int chromaOffset, float rateTol, byte
pass, boolean state, int direct, int meRange, boolean weightB, boolean noFastPSkip){}

    ایجاد کلاس سِتِر برای همه پارامترهای این کلاس#
    public void setMinKeyInt(int minKeyInt){
        this.minKeyInt = minKeyInt;
        return this.minKeyInt = minKeyInt;
    }

    ایجاد کلاس گِتِر برای همه پارامترهای این کلاس#
    public int getMinKeyInt(){
        return minKeyInt;
    }

}
```

کلاس main شیوه بدون Builder به شکل زیر خواهد شد

```
public class Main {
    public static void main(String[] args){
        H263Properties  decoder – new  H263Properties();
        decoder.setbAdapt(12) ;
        decoder.setbweghtB(yes) ;
        decoder.setrateTol (1.5) ;
        …
        تعداد بسیار زیاد باید تنظیم نماید
    }
}
```

۴.۴.۳.۲)همراه Builder

باید کلاسH۲۶۴Properties بدون Builder را همانند بخش قبل داشته باشیم و همراه آن کلاس در وضعیت Builder نیز به شکل زیر تولید شود

```
public class H263PropertiesBuilder{
```

```
    int keyInt;
    int minKeyInt;
    int sceneCut;
    int bFrames
    int bAdabt
    int qp
    int bitrate
    boolean bFrameBias
    int crf
    int qpstep
    int pbRatio
    int chromaOffset
    float rateTol
    byte pass
    boolean state
    int direct
    int meRange
    boolean weightB
    boolean noFastPSkip

    نوشت تابع مشابه ستِر اما بدون کلمه ستِر برای همه پارامترهای این کلاس به شکل زیر#
    public  H263PropertiesBuilder MinKeyInt(int minKeyInt){ نکته: کلمه ستِ از نام تابع حذف شده است#
        this.minKeyInt = minKeyInt;
        return this;
    }

    public H263Properties build() {
        H263Properties decoder = new  H263Properties();
        decoder.setbAdapt(badapt) ;
        decoder.setbweghtB(weightB) ;
        decoder.setrateTol (rateTol) ;
        ...
        //یک بار برای تک تک پارامترها مقدارها را قرار می‌دهیم
    }
}
```

کلاس main شیوه Builder به شکل زیر خواهد شد

```
public class Main {
    public static void main(String[] args){
        H263Properties decoderBuilder = new  H263PropertiesBuilder();
        H263Properties decoder = builderDecoder.frame(12).KeyInt(17).rateTol(1.5).minKeyInt(2)
                                        .sceneCut(9)…….build() ;
    }
}
```

نکته‌ها

۱. در کلاس Builder مقدار بازگشتی تابع Setter هرکدام از پارامترها باید بجای نوع(مثلا int یا string یا ...) به نام کلاس تغییر پیدا کند

۲. در کلاس Builder کلمه set از نام تابع بازگشتی حذف می‌شود

۳. در کلاس Builder توابع getter همانند وضعیت بدون Builder خواهند بود

۴. در کلاس Builder تابع build را ایجاد نماییم که قرار است خروجی نهایی رو برگرداند

۴.۵) الگوی‌طراحی Prototype

در برنامه‌نویسی و طراحی نرم‌افزار، الگوی **Prototype** (پروتوتایپ) یکی از الگوهای طراحی (Design Patterns) است که به شما اجازه می‌دهد تا اشیاء را با کپی کردن از یک شیء موجود به جای ایجاد یک شیء جدید از طریق سازنده (Constructor) ایجاد کنید. این الگو به ویژه در مواقعی مفید است که ایجاد یک شیء جدید از طریق سازنده هزینه‌بر یا پیچیده باشد.

هنگامی‌که ساخت شیء با Cost زیاد همراه باشد(متدهای زیادیcall می‌شوند و دیتای زیادی در کلاس موجود است که باید تک تک آن‌ها در حافظه بارگزاری و مقداردهی شوند) آنگاه، بجای ساخت شیء جدید، نمونه شیء قبل از کلاس را Clone کنیم. یعنی بخشی از حافظه را duplicate نماییم تا نیاز به بارگزاری تک تک اجزا در حافظه نباشد

- توسط متد clone می‌توان یک آبجکت را کاملاً کپی کرد
- جلوگیری از تولید اشیاء پر هزینه توسط new[عدم استفاه از کلیدواژه new]
- توابع سازنده با «Constructor» ها فراخوانی نمی‌شوند، پس ممکن است در صورت وابستگی به توابع سازنده آنگاه منطق برنامه گاهی دچار اختلال شود(مواردی که تابع سازنده با منابعی مستقل و بیرون از کلاس کارداشته باشند تا هر زمان وضعیت جدیدی ممکن است وقوع پیوندد)
- هر نمونه ایجاد شده یک instance مستقل و منحصر بفرد است
- گاهی اوقات از یک interface استفاده می‌کند که مثلاً اسم آن را cloneble قرار بدهند
- معمولا همراه با «الگوی طراحی» Registry استفاده می‌شود
- تضمین اشیاء یک شکل
- نمونه‌ها:
  - تابع clone که در java.lang.Object موجود است

۴.۵.۱)انواع کپی کردن شیء

۴.۵.۱.۱)Shallow Copy (کم عمق)

- فقط خود شیء کپی می‌شود، اما مراجع به اشیاء داخلی (اگر وجود داشته باشند) به همان مراجع قبلی اشاره می‌کنند.
- اگر شیء کپی شده دارای مراجع به اشیاء دیگر باشد، این مراجع در کپی جدید به همان اشیاء اشاره خواهند کرد.
- فقط خود شیء کپی می‌شود و مراجع به اشیاء داخلی به همان اشیاء اشاره می‌کنند.
- متد clone موجود در کلاس object از این نوع است.

مثال برای shallow copy

```
class Person {
    String name;
    Address address; // Address یک شیء دیگر است
    Person(String name, Address address) {
        this.name = name;
        this.address = address;
```

```
    }
}
class Address {
    String city;
    Address(String city) {
        this.city = city;
    }
}
Person original = new Person("Alice", new Address("New York"));
Person shallowCopy = original;
```

۴.۵.۱.۲) Deep copy

- کپی کردن شیء به گونه‌ای که نه تنها خود شیء کپی می‌شود، بلکه تمام اشیاء داخلی آن نیز به طور مستقل کپی می‌شوند.
- به این ترتیب، تغییرات در کپی جدید بر روی اشیاء داخلی تأثیری بر روی شیء اصلی نخواهد داشت.
- خود شیء و تمام اشیاء داخلی به طور مستقل کپی می‌شوند.

مثال برای deep copy :

```
class Person {
    String name;
    Address address;
    Person(String name, Address address) {
        this.name = name;
        this.address = address;
    }
    // متد برای دیپ کپی
    Person deepCopy() {
        return new Person(this.name, new Address(this.address.city));
    }
}
Person original = new Person("Alice", new Address("New York"));
Person deepCopy = original.deepCopy(); // این یک دیپ کپی است
```

۴.۵.۱.۳) استفاده از روش serialize کردن Object ها

- کلاس‌هایی که serializable باشند قابلیت تبدیل به بایت شدن دارند. که این بایت‌ها قابلیت نگهداری در داخل پایگاه داده یا فایل دارند
- برای این کلاس‌ها object writer تعریف می‌شود
- به این ترتیب که شیء را serialize می‌کنند در داخل مموری می‌ریزند و همان آرایه رو مجددا به شیء تبدیل می‌کنند. که این روش مقداری Cost زیاد دارد

۴.۵.۲)مثال۱

```
interface Prototype {
    Prototype clone();
}

class ConcretePrototype implements Prototype {
    private String name;

    public ConcretePrototype(String name) {//سازنده
        this.name = name;
    }

    @Override
    public Prototype clone() { // پیاده‌سازی متد کلون
        return new ConcretePrototype(this.name);
    }

    @Override
    public String toString() {
        return "ConcretePrototype{name='" + name + "'}";
    }
}

public class PrototypePatternExample { // کلاس اصلی
    public static void main(String[] args) {
        ConcretePrototype original = new ConcretePrototype("Original"); // ایجاد یک شیء اصلی
        ConcretePrototype cloned = (ConcretePrototype) original.clone();// کپی کردن شیء اصلی

        // نمایش اشیاء
        System.out.println(original);
        System.out.println(cloned);
    }
}
```

۴.۵.۳)مثال۲

```
Statement statement = new Statement()
statement.setRecord(new Record());
statement.setProjection(“select firesName , lastName”);
statement.setForm(“from Employee”);
statement.setWhere(“where id = 12”);

System.out.println(statement.hashCode());//326573597
System.out.println(statement.getRecord().hashCode());//1735600054

Statement clone = statement.clone();
System.out.println(clone.hashCode());//21685669
System.out.println(clone.getRecord().hashCode());//1735600054
```

یک‌نمونه:

```
public class Registry {
    private static Map<String, Item> cacheData =new HashMap<>();
    static {
        cacheData.put(“1” , “First  Item” , “IT1” , “http://ad.com/IT1”));
        cacheData.put(“2” , “Second Item” , “IT2” , “http://ad.com/IT2”));
    }
    public static Item getItem(String code) {
        if (cacheFata.containKey(code)) {
            Item item = cacheData.get(code);
            return item.clone();
        } else {
            return null;
        }
    }
}
```