הכנת סביבת העבודה

כדי למנוע תקלות של הרשאות כתיבה וכדומה C יצירת תיקייה בשם שלכם בכונן
c:/eldar

.מותקן כבר על המחשבים של הכיתה – JDK התקנת

.אבל מה שנוח לכם. להוריד לתיקייה שיצרתם (ללא התקנה) Eclipse אני עובד עם – IDE הורדת

workspace יצירת תיקיית

c:/eldar/workspace

מי שעובדים עם Eclipse להוריד ZIP ולא קובץ התקנה:
Eclipse IDE for Java Developers

הכנת סביבת העבודה

כדי למנוע תקלות של הרשאות כתיבה וכדומה C יצירת תיקייה בשם שלכם בכונן
c:/eldar

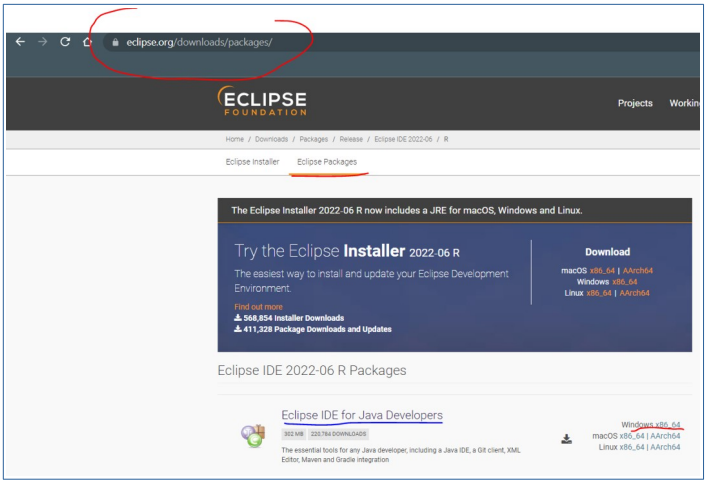
.מותקן כבר על המחשבים של הכיתה – JDK התקנת

.אבל מה שנוח לכם. להוריד לתיקייה שיצרתם (ללא התקנה) Eclipse אני עובד עם – IDE הורדת

workspace יצירת תיקיית

c:/eldar/workspace

מי שעובדים עם Eclipse להוריד ZIP ולא קובץ התקנה:
Eclipse IDE for Java Developers



אפשרויות עם בנאים:

אפשר להגדיר כמה בנאים שרוצים @Component בכל מחלקה עם

- (תשתמש בו ותזריק spring) אוטומטית autowired בנאי אחד עם פרמטרים: מקבל
- ספרינג תשתמש בבנאי - autowired בנאי אחד ללא פרמטרים ועוד בנאים עם פרמטרים אבל ללא שלא מקבל פרטמרים.
- ספרינג תשתמש בבנאי שמקבל - autowired בנאי אחד ללא פרמטרים ובנאי עם פרמטרים ועם (על השדות autowired מותר בנוסף לתת) .פרמטרים ותזריק
- ספרינג תשתמש בבנאים עם - autowired מספר בנאים שמקבלים פרמטרים אבל רק אחד עם ותזריק autowired.
- ספרינג תזרוק שגיאה שאין בנאי ברירת - autowired מספר בנאים שמקבלים פרמטרים אבל ללא מחדל.
- אסור יותר מבנאי אחד עם autowired.

אסטרטגיה לביצוע Autowired

- אז מוזרק (type לפי) אם יש התאמה אחת בלבד
- אם יש Primary אם יש יותר מאחת אז לפי
- Qualifier אז לפי Primary אם יש כמה אפשרויות ואין
- לפי שם השדה

שתי דרכים להגדיר bean

1. להגדיר class עם אנוטציה Component – כשמבקשים bean ה container מפעיל בנאי
2. להגדיר מתודה עם אנוטציה Bean – כשמבקשים bean ה container מפעיל את המתודה

Annotations

@ComponentScan

שמים על class למטרת קונפיגורציה, כדי להגדיר סריקת מחלקות כדי לאתר הגדרות של beans

@Configuration

שמים על class למטרת קונפיגורציה, כדי להגדיר שה class יכול להכיל bean methods

@PropertySource

שמים על class למטרת קונפיגורציה, כדי להגדיר מיקום לקובץ properties

@Component

שמים על מחלקה כדי להגדיר bean מהסוג של המחלקה

@Bean

שמים על מתודה כדי להגדיר bean מהסוג שהמתודה מחזירה (בתוך class שמכיל אנוטציית Configuration)

@Scope("prototype")
@Scope("singleton")

קונפיגורציה להגדרת bean – האם יהיה סינגלטון או פרוטוטייפ

@Autowired

הוראה להזריק bean שנדרש ל bean הנוכחי:

1. CTOR
2. field
3. setter

@Qualifer

מצורף בהקשר של Autowired כשיש יותר מהתאמה אחת ורוצים לציין איזה להזריק

@Primary

מצרפים להגדרה של bean כדי לקבוע שהוא ההתאמה העיקרית במקרה שיש כמה אפשרויות.

@Value

מצרפים לשדה או פרמטר בבנאי או מתודה כדי להזריק ערך מתוך קובץ properties

@Lazy

שמים על הגדרה של bean שהוגדר כ singleton על מנת לשנת את ההתנהגות ברירת מחדל של טעינה Eager. כלומר ה bean לא ייווצר עד שמישהו מבקש אותו. לא רלוונטי ל prototype (תמיד lazy)

Spring AOP – Aspect Oriented Programming

AOP – פרדיגמה בתכנות שמתמקדת ב aspects (היבטים) – שהם נושאים לטיפול שמשתרעים על יותר ממודול אחד של האפליקציה.

לכל מודול באפליקציה יש עניין ליבתי core concern – למשל פעולות מול database

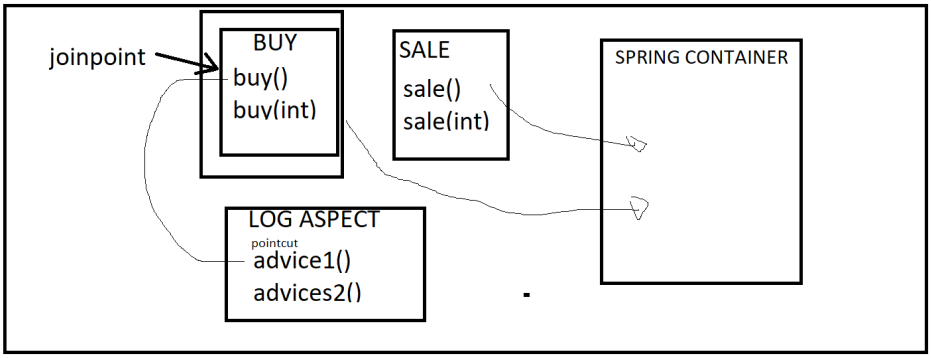
aspect הוא עניין שיש לטפל בו אבל הוא חוצה את האפליקציה – cross cutting concern

למשל:

Aspect are modules for cross cutting concerns of the application
mainly for:

- logging
- security
- statistic information

<u>Terminology</u>
aspect – a class with advice methods
advice – a method in aspect class to be run with joinpoints
joinpoint – a businesses method intercepted by advice
pointcut – a text expression for matching advice to jpoinpoints



Advice types:

1. Before
2. After
3. After Returning
4. after Throwing
5. Around

<u>Annotations</u>
@Aspect – on aspect class
@Before – on advice method
@EnableAspectJAutoProxy – on configuration class

pointcut expression

execution(
modifers
return type
declaring type pattern
method name pattern
parameters pattern
throws pattern
)

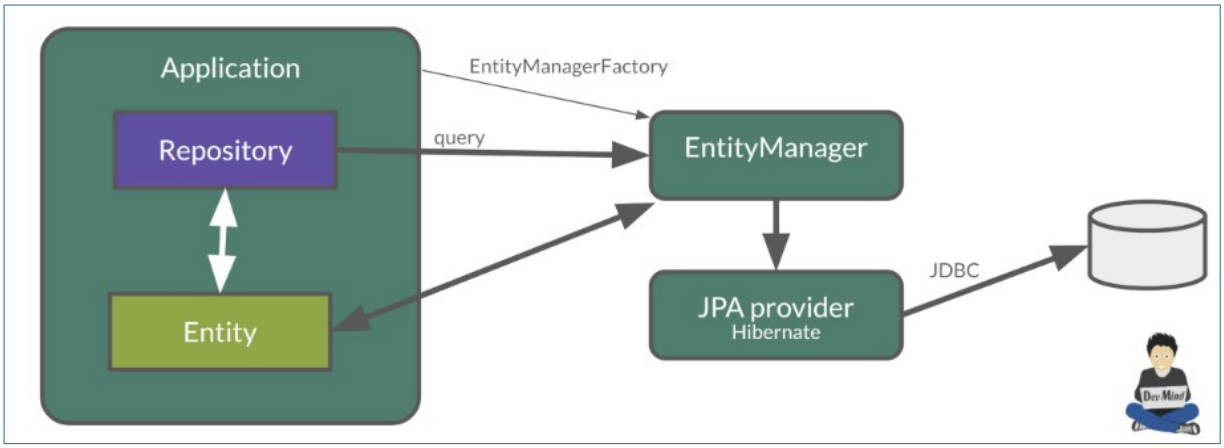
// 1. modifiers - any if not specified
// 2. return type - * = any [mandatory]
// 3. declaring type (package.class.) - any if not specified
// 4. method pattern [mandatory]
// 5. parameter pattern [mandatory]
// 6. throw pattern

parameters
() - no parameters
(type1, type1) – specific parameters
(*) – one parameter of any type
(..) – 0 or more parameters of any type

```java
// POINTCUTS

// no annotations in business
@Pointcut("execution(java.lang.String divide(Integer, Integer))")
public void div() {
}

// for annotations on methods
@Pointcut("@annotation(app.core.aspects.annotations.MyLogAnnotation)")
public void annotatedMethod() {
}

// for annotations on class
@Pointcut("@target(app.core.aspects.annotations.MyLogAnnotation)")
public void annotatedClass() {
}
```

How to Define Pointcuts

JPA


Application
Repository
Entity
EntityManagerFactory
query
EntityManager
JPA provider
Hibernate
JDBC

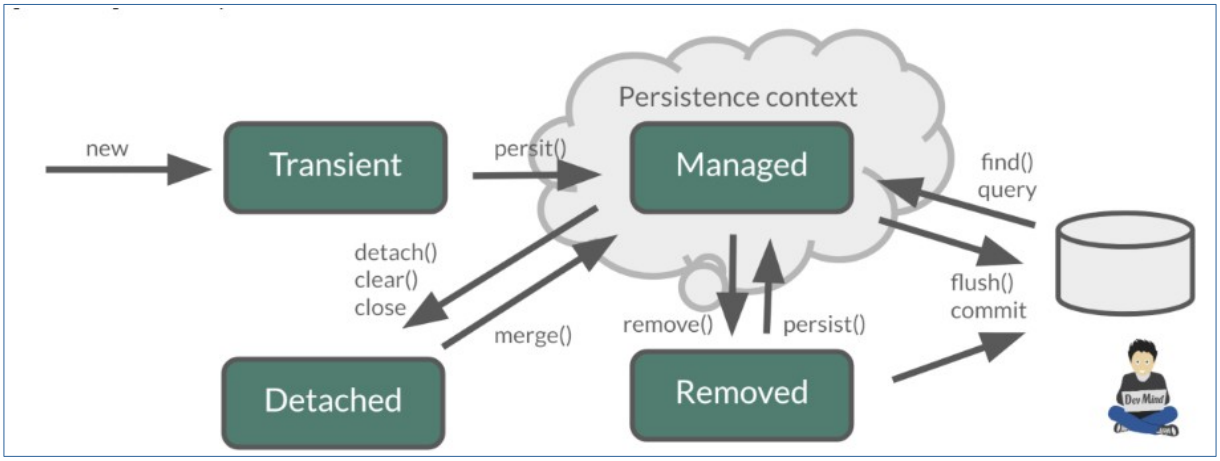
מחזור חיים של entities

- ניתן ליצור אובייקט חדש ולשמור אותו ל database
- ניתן למשוך אובייקט מתוך ה database

ברגע שיש entity יש שלבים שונים במחזור החיים שלו

Entity Manager

משמש להרצת queries



- **Transient Objects**: Transient objects are non transactional and in fact Hibernate has no knowledge of these objects
  - לא נמצאים בטרנסאקציה ו hibernate בכלל לא יודעת עליהם
- **Persistent Objects**: Persistent entity has a valid database identity associated with.
  - ל
- **Removed Object**: An object scheduled for deletion either by calling delete or because of orphan deletion of entities.
- **Detached Object**: The object in persistent state go into detached state after the persistent context is closed. Detached objects can be brought into other persistent context by reattachment or merging. Detached object still has a valid primary key attribute but it is no longer managed by Hibernate.

We have different operations to several stages in the life-cycle.

- **persist**() makes a persistent entity. It will be written in the database at the next commit of the transaction we are in..
- **remove**(): inverse of persist(). It will be erased from the database at the next commit of the transaction we are in.
- **refresh**(): synchronizes the state of an entity to its database state. If the fields of an entity have been updated in the current transaction, these changes will be canceled. This operation only applies to persistent entities (otherwise we have an IllegalArgumentException)
- **detach**(): detaches an entity from entity manager. This entity will not be taken into account during the next commit of the transaction in which we are
- **merge**(): attach an entity to the current entity manager. This is used to associate an entity with another entity manager than the one that was used to create or read it.

https://dev-mind.fr/training/spring/spring-data.html