Clase 26/3

#PAW

Para el tp: **JAVA 8**

Siempre que devuelvas colecciones nunca devuelvas NULL, devolve lista vacía ➜ *least surprise*

NULL es mala palabra desde Java 8 ➜ **Optional**User [igual que *Maybe*!!! en Haskell :)))))]

userList.stream().findFirst() retorna un Optional<User> ➜ null safe

orElseThrow método que devuelve Optional y te pide excepción (factory).

Definir nueva dependencia:

```
<groupId>org.postgresql</groupId>
<artifactId?postgresql</artifactId>
<version>${postgresql.version}</version> ..



# agrego property en pom.xml y persistence/pom.xml
<postgre ..
```

y después mvn eclipse:eclipse

WebConfig

```
@Bean
 ... viewResolver() {
}

@Bean
```

```
public DataSource dataSource() {
  final SimpleDriverDataSource ds = new SimpleDriverDataSource();

ds.setDriverClass(org.postgresql.Driver.class)
ds.setUrl("jdbc:postgresql://localhost/paw")
ds.setUsername()
ds.setPassword()
```

agrego dependencia de postgresql a webapp pom.

@Autowired: spring se mete y te completa todo

Crear database

psql paw -U root -W

Evitar escribir query a mano:

```
@Autowired
public UserJdbcDao(final DataSource ds)    {
jdbcTemplate = new JdbcTemplate(ds);
jdbcTemplate.execute(“CREATE TABLE IF NOT EXISTS user (“ +
  “id SERIAL PRIMARY KEY,” +
  “username varchar(100) UNIQUE” +
  “)”);
}
```

en UserJdbcDao:

```
@Override
public User create(String username) {
  final Map<String, Object> = args
```

```
  args.put("username", username);
  final Number userId = jdbcInsert.executeAndReturnKey(args);
  return new User(userId.longValue(), username);
}
```

Me garantiza pleno control en un solo lugar que las entidades de modelo se reflejan en persistencia (no creo instancias del modelo que no existan).

user tiene el id auto generado y *username*.

En HelloWorldController

No hay ningún flujo normal donde mi aplicación no pueda retornar un user.

```
@RequestMapping("/create")
public ModelAndView create(@RequestParam(value = "name", required = true) f….
  final User u = us.create(username);
  return new ModelAndView("redirect:/user/" + u.getId());
}
```

Test unitarios

¿Cómo testeo DAO?

¿Qué condiciones debo pedir? que mis tests sean reproducibles. Siempre los mismo resultados no importa el orden. (No Heisen bugs) ➜ motor

HSQLDB

agrego dependencia solo para scope *test* de HSQLDB

agrego dependencia spring-test con scope test

en persistencia defino dependencia con groupie y artifact el resto lo hereda

agrego dependencia de junit

agrego property junit

coding en src*main*java ➜ tests en src*test*java
en el mismo package (persistence, etc.)

Test

Test algo específico.

UserJdbcDao
➜ método *create* ➜ que devuelva el username y lo inserte en la base de datos
➜ método findById ➜ si debe devolver que lo haga

Test correcto: valida un único comportamiento

1. **Setup**: precondiciones
2. **Ejercitar** mi clase
3. **Meaningful asserts**: solo cosas que tengan sentido, un buen testing tiene poco asserts

En una sola corrida tengo qué está bien y qué mal.
Para el tp, nos exigen que sean rápidos de ejecutar.
Si quiero probar findById, en setup create este usuario. Pero si el create está roto, el resto puede fallar. El comportamiento puede ser errático.

Mocks ➜ libs generan implementación adhoc de un proxy y puede hacer que devuelva lo que quiero.

Para el tp vamos a usar: **Mockito**

Tests de caja negra (solo conozco interfaz y contrato que expone).
Tests de caja blanca busca si tengo un if que entre por las dos ramas, casos bordes en fors... Conozo implementación.

@RunWith(SpringJunit4ClassRunner.class)
@ContextConfiguration(classes = TestConfig.class) ➜ pars autowireds
UserJdbcDaoTest {

@Autowired (ver alpha)
private DataSource ds;

@Autowired
private UserJdbcDao userDao;

private JdbcTemplate jdbcTemplate;

alpha: hacer clase TestConfig con @ComponentScan (....persistance...) con @Configuration como WebConfig con dataSource() con setUrl “jdbc:hsqldb:mem:paw”

@Before
P v Setup() {

jdbcTemplate = new JdbcTemplate(null);
UserDao = new userDao(null);
JdbcTestUtils.deleteFromTables(jdbcTemplate, “users”);

}

@Test a veces puedo decir que nullexception es expected acá
public void testCreate() {
final User user = userDao.create (USERNAME);

assertEquals(USERNAME, user.getUsername());
}

Errores comunes: assertTrue assertEquals, !assertTrue, orden de parametros de assert, no poner métodos deprecated