1. How to use Java code during table creation set the column size on the table on varchar2 on the DB

Use the following approach:
@Size(min = 3, max = 15)
private String name;

Link: https://thoughts-on-java.org/hibernate-tips-whats-the-difference-between-columnlength50-and-sizemax50/

2. Discussion on the awareness of inheritance strategy in Spring and how it is represented on the

DB layer and Java layer

3. JPQL Join on multicolumn is it possible and how to retrieve it, Provide an Exercise for students to achieve the same to ensure they understand it

Link: https://www.tutorialspoint.com/jpa/jpa_jpql.htm

JPA Tutorial :

https://mkyong.com/spring-boot/spring-boot-spring-data-jpa/

https://dzone.com/articles/spring-boot-with-spring-data-jpa

https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa

3. Explanation on how do you explain dependency injection to the students in easy way even when the technical way is easy to put it across.

Dependency Injection is basically a way of creating singleton, and injecting those singleton as parameters to other methods, without explicitly calling it.

```
@Configuration
public class AppConfig {

    @Value("${spring.datasource.url}")
    private String url;

    @Value("${spring.datasource.username}")
    private String username;

    @Value("${spring.datasource.password}")
    private String pswd;

    @Value("${spring.datasource.driver-class-name}")
    private String drivername;

    @Value("${spring.datasource.schema-username}")
    private String schema;

    @Bean
    public JdbcTemplate getJdbcTemplate(DataSource dataSource) {
        return new JdbcTemplate(dataSource);
    }

    @Bean
    public DataSource configureDataSource() {

        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(this.drivername);
        dataSource.setUrl(this.url);
        dataSource.setUsername(this.username);
        dataSource.setPassword(this.pswd);
        dataSource.setSchema(this.schema);
        return dataSource;
    }


}
```

4. class triangle has 3 points needed and using auto wiring can you set the object, what challenges you will face and how to get it resolved, have an example exercise for students to do the same.

Refer to `AppConfig Class example`…

Note : @Autowired can only work in @Component or similarly annoted classes only.

5. Using integer values can be used to set the value on to the spring object, Provide and show an example to the students on the same

6. Create one example for the students to convert String to date Object via xml file.

I am hoping the following approach could achieve that , Any better approach can be discussed

```
<bean id="dateFormat" class="java.text.SimpleDateFormat">
<constructor-arg value="yyyy-MM-dd" />
</bean>

<bean id="customer" class="com.mkyong.common.Customer">
<property name="date">
<bean factory-bean="dateFormat" factory-method="parse">
<constructor-arg value="2010-01-31" />
</bean>
</property>
</bean>
```

7. Reading Spring property file reading capability

```
@Configuration
@PropertySource("classpath:foo.properties")
public class PropertiesWithJavaConfig {
  //...
}
```

==========

```
@Configuration
public class Query {

    @Value("${prop.varname.value}")
    private String VarName;
```

In “application.yml”

```
prop:
  varname:
    value:
```

In “application.properties”

```
prop.varname.value: scpay-deterministic Application
```

8. Can I switch between @component and @repository is possible.

Link: https://stackoverflow.com/questions/6827752/whats-the-difference-between-component-repository-service-annotations-in

9. Use an employee JSON object coming via a angular code, The object should receive it as a employee object instead of String and not try to convert later to JSON

sending and receiving employee object as request and response exercise for students

Link: https://www.baeldung.com/spring-boot-bean-validation

```
@RestController
public class UserController {

    @PostMapping("/users")
    ResponseEntity<String> addUser(@Valid @RequestBody User user) {
        // persisting the user
        return ResponseEntity.ok("User is valid");
    }

    // standard constructors / other methods

}
```