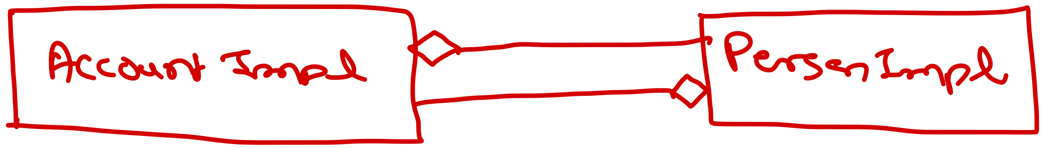
Trainer: Nilesh Ghule

Wake up from Hibernate, Spring up!!!



accHolder

account

↓

when using @Autowired on ctor level,
circular dependency error.

calling param ctor to create the object

① recheck design.

①

Spring Expression Language

spring reference doc ① core ② mvc

- SpEL is a powerful expression language that supports querying and manipulating an bean object graph at runtime. Syntactically it is similar to EL. → JSP EL ${ — }
- SpEL can be used in all spring framework components/products.
- SpEL supports Literal expressions, Regular expressions., Class expressions, Accessing properties, Collections, Method invocation, Relational operators, Assignment, Bean references, Inline lists/maps, Ternary operator, etc.
- SpEL expressions are internally evaluated using SpELExpressionParser.
  - ExpressionParser parser = new SpELExpressionParser();
  - value = parser.parseExpression("'Hello World'.concat('!')");
  - value = parser.parseExpression("new String('Hello World').toUpperCase()");
  - value = parser.parseExpression("bean.list[0]");
- SpEL expressions are slower in execution due parsing. Spring 4.1 added SpEL compiler to speed-up execution by creating a class for expression behaviour at runtime.

Bean scopes

```
<bean id="a1" class="AccImpl"/>        ← singleton
<bean id="a2" class="AccImpl"/>          beans
    r1 = getBean("a1") | r2 = getBean("a1")
    r3 = getBean("a2");
```

- singleton bean
  - class is not singleton.
  - multiple bean objs for class is possible.
  - only one bean obj for given id.
- singleton class
  - only one obj for the class.
  - if tried to create another obj, return same obj.
  - typically created using factory method e.g. getInstance().

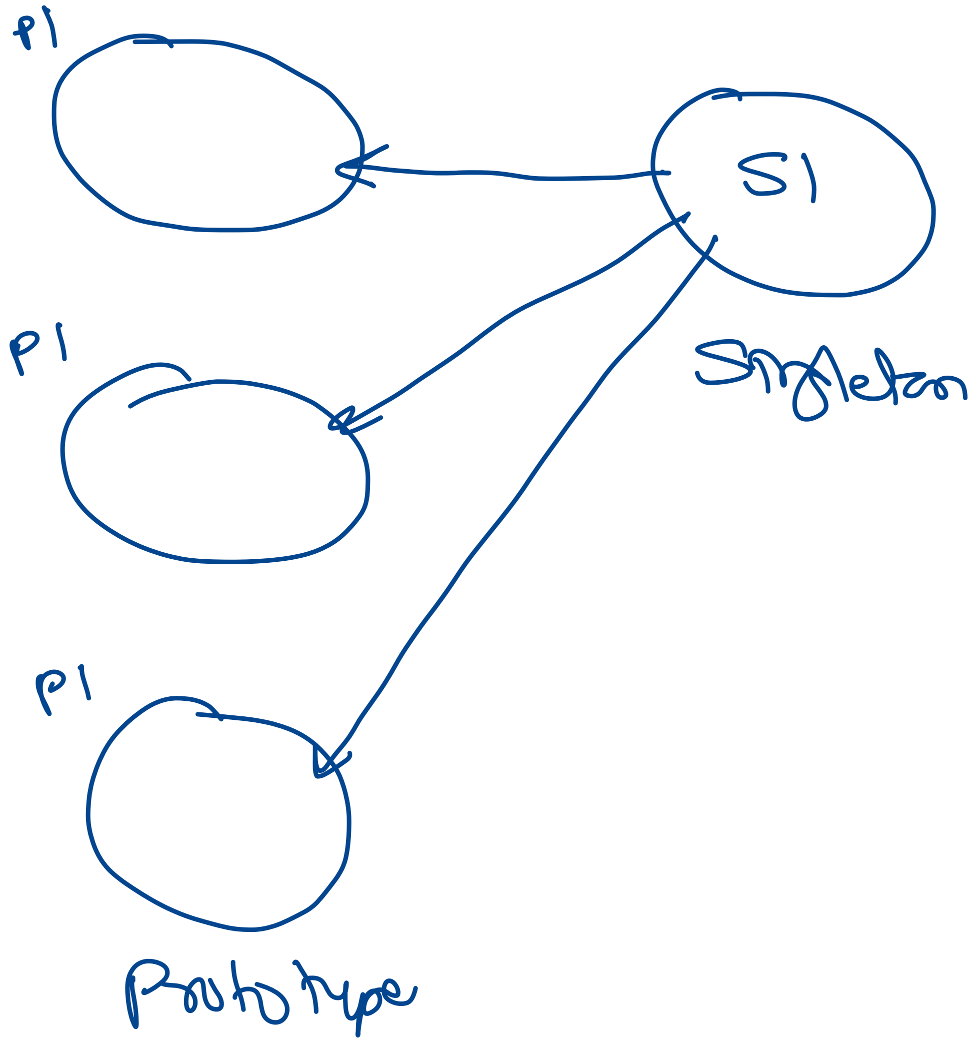
- Bean scope can be set in XML or annotation.
  - <bean id="___" class="___" scope="singleton|prototype|request|session" />
  - @Scope("singleton|prototype|request|session")
- singleton default - stateless beans e.g. service, dao, ...
  - Single bean object is created and accessed throughout the application.
  - XMLBeanFactory creates object when getBean() is called for first time for that bean. (lazy)
  - ApplicationContext creates object when ApplicationContext is created. (eager).
  - For each sub-sequent call to getBean() returns same object reference.
  - Reference of all singleton beans is managed by spring container. → never gc.
  - During shutdown, all singleton beans are destroyed (@PreDestroy will be called).

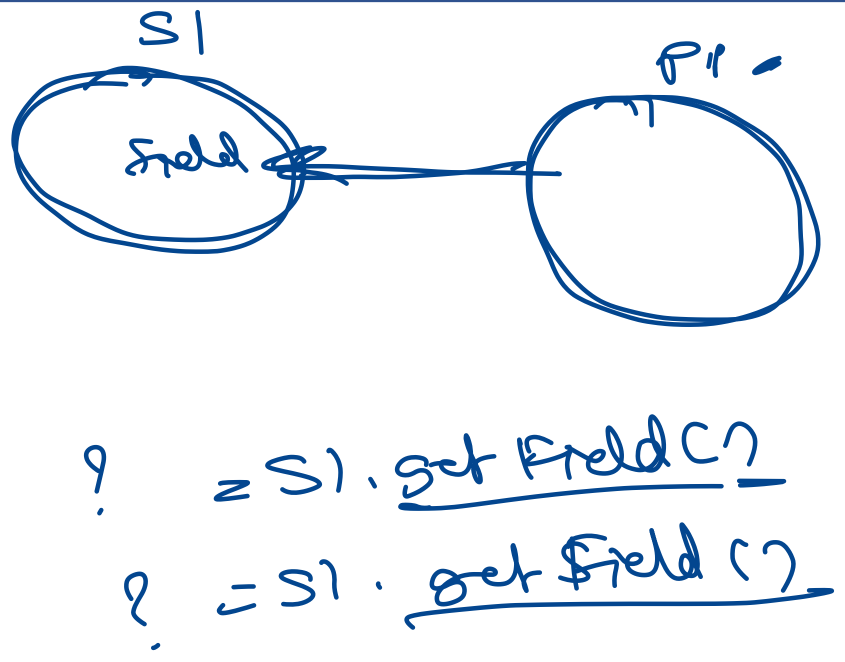
    ctx.close()
- prototype - stateful.
  - No bean is created during startup.
  - Reference of bean is not maintained by ApplicationContext. → gc
  - Beans are not destroyed automatically during shutdown.
  - Bean object is created each time getBean() is called.
- request and session: scope limited to current request and session. → web applns,

PersonImpl → prototype.

```
p1 = new PersonImpl();
initialize(p1);
```

Bean scopes


P1
P1
P1
S1
Singleton
Prototype


S1
P1
field
? = S1.getField()
? = S1.getField()

Bean scopes

- ✓ Singleton bean inside prototype bean
  - Single singleton bean object is created.
  - Each call to getBean() create new prototype bean. But same singleton bean is autowired with them.
- ✓ Prototype bean inside singleton bean
  - Single singleton bean object is created.
  - While auto-wiring singleton bean, prototype bean is created and is injected in singleton bean.
  - Since there is single singleton bean, there is a single prototype bean.
- Need multiple prototype beans from singleton bean (solution 1)
  - Using ApplicationContextAware
  - The singleton bean class can be inherited from ApplicationContextAware interface.
  - When its object is created, container call its setApplicationContext() method and give current ApplicationContext object. This object can be used to create new prototype bean each time (as per requirement).
- Need multiple prototype beans from singleton bean (solution 2)
  - using @Lookup method
  - The singleton bean class contains method returning prototype bean.
  - If method is annotated with @Lookup, each call to the method will internally call ctx.getBean(). Hence for prototype beans, it returns new bean each time.

Stereo type annotations

- Auto-detecting beans (avoid manual config of beans).
  - ✓ @Component – generic bean
  - ✓ @Service – business logic & tx mgmt
  - ✓ @Repository – dao beans.
  - ✓ @Controller and @RestController – web mvc – navigation.
    – web mvc rest services.
- In XML config file
  - <context:component-scan basePackages="___"/>
- Annotation based config
  - @ComponentScan(basePackages = "pkg")
  - includeFilters and excludeFilters can be used to control bean detection.

BeanPostProcessor may run specific to annotation

auto search bean classes with stereo type anns in packages & sub-packages.

Spring JDBC Integration

- Spring DI simplifies JDBC programming.
- Using JDBC we can avoid overheads of ORM tools. This is helpful in small applications, report generation tools or running ad-hoc SQL queries.
- Steps:
  - In pom.xml, add spring-jdbc and mysql-connector-java
  - Create dataSource bean (XML or annotation config) $\rightarrow$ con info
  - Create JdbcTemplate bean (XML or annotation config) and attach dataSource,
  - Implement RowMapper interface in a class (for dealing with SELECT queries)
    - mapRow() convert resultset row to Java object.
  - Create Spring @Repository bean and auto-wire JdbcTemplate in it.
  - Invoke JdbcTemplate query() and/or update() for appropriate operations.
  - To use transaction management, create TransactionManager bean and use @Transactional on service layer (common practice).

JDBC steps
① register driver
② create connection
③ create statement
④ executeQuery()
executeUpdate()
& process result,
⑤ close all.

SUNBEAM

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>