Spring Container (also known as ApplicationContext) -> Inversion of control + Dependency Injection.

Inversion of control – Outsourcing of creation and maintenance of object.

Dependency injection – Injection the dependency object where needed implementing loose coupling.

Configuration of Spring Container:

1. XML Configuration file (legacy)
2. Java Annotations (modern)
3. Java Source Code (modern)

Dependency Injection:

1. Constructor injection
2. Setter Injection
3. Method Injection (not in xml) //simply use @Autowired
4. Field Injection //Instead of above injection use this for automatic injection

Bean Scope:

1. Singleton
2. Prototype
3. Request
4. Session
5. Global-Session

Bean LifeCycle/Hooks:

1. init-method
2. destroy-method

NOTE: FOR PROTOTYPE SCOPED BEAN, SPRING DOES NOT CALL DESTROY METHOD

**Aspect Oriented Programming (AOP):**

Types of Advice: Before, After finally, After throwing, After returning, Around

AOP Project Setup:

* Download and put aspectJ jar (dependency) in class path
* Add @EnableAspectJAutoProxy above the main class
* Add @Aspect above the Component class
* Add @Before(“execution(public void addAccount())”) above the method you will write to be executed before addAccount() method
* Pointcut is execution(…..) //format = “execution(modified? returnType fullyQualifiedClasName? methodName() throws?)”
* Use \*(any) to fill anything. Eg – add\* method name will look for all methods starting with add
* Parameter match – () with 0 parameter, (\*) with 1 parameter, (..) with 0 or more parameters
* @PointCut(“execution(\* add\*(..))”) //serves reusability

Private void forAnyAddMethod(){}

@Before(“forAnyAddMethod()”)

Public void addNums(int a,int b){}

* PointCut expression can be combined using &&, || and ! as a single pointcut declaration
* Order the advices by using @Order(n) using above aspect class. //put advices in different aspects for ordering.
* JoinPoint class (inbuilt) is used to get method signature and parameters of pointcut method
* @Before(“pointcut expression method name()”)

Public void anyMethod(JoinPoint theJoinPoint){

MethodSignature methodSig = (MethodSignature) theJoinPoint.getSignature();

Sysout(methodSig);

Object[] args = theJoinPoint.getArgs();

For(Object arg:args){

Sysout(arg);

}

}

* @AfterReturning(pointcut=”execution(“\* add\*(..))”, returning = “result”)

Public void anymethod(JoinPoint joinPoint, List<Account> result){

Sysout(result);

}

* @AfterThrowing(pointcut=”execution(“\* add\*(..))”, throwing = “exc”)

Public void anymethod(JoinPoint joinPoint, Throwable exc){

Sysout(exc.printstacktrace());

}

* @After() //works in both case whether success or exception in code. After runs after AfterThrowing from spring 5.2.7
* @Around(“execution(\* add\*(..))”)

Public void methodName(ProceedingJoinPoint theProceedingJoinPoint) //ProceedingJoinPoint is the handle to the method

{

theProceedingJoinPoint.proceed(); //this will execute the add method

}

//Use Logger api of java for using single output stream. Spring uses this and is inbuilt in java.util

//@Around can be used for exception handling, put proceedingJoinPoint.proceed() inside try block.

//Enable AOP logging by: private Logger logger = Logger.getLogger(getClass().getName());

Logger.info(“print anything”);

WORKING OF MAVEN:

* Download dependencies
* Build/class path setup
* Project structure

All above are handled by Maven.

SPRING REST Exception Handling:

@ExceptionHandler //put above a method in the same controller

Public ResponseEntity<CustomErrorClass> handle(Exception //can specify the exact exception class as well// ex){

CustomerErrorClass customerErrorClass = new CustomerErrorClass();

customerErrorClass.setReason(ex.get..)

customerErrorClass.setTimeStamp(system.getTimeinmilli..)

..so on

Return ResponseEntity< customerErrorClass>;

}

@ControllerAdvice //use for universal exception handler, realtime use of AOP