Revise the following concepts
==========
Ananomyous inner class, lambda expression, method reference

Annotation support in AOP
==========
1. @Aspect
2. @After
3. @Before
4. @Around
5. @AfterReturning
6. @AfterThrowing
7. @Pointcut

JoinPoint => Combination of Advice + PointCut

@Around => It would be used combination of @After and @Before
ProceedingJoinPoint => This interface help us to make a call to buisness method(need to be called using proceed())

Eg#1.

```
@Component
public class EmployeeDao {
	public Integer saveEmployee() { //#3
		System.out.println("Employee Object is saved...");
		return 10;
	}
}
```

```
@Pointcut("execution(public * in.ineuron.dao.EmployeeDao.*())")
public void p1() {}

@Around("p1()")
public void aroundAdvice(ProceedingJoinPoint jp) {

	System.out.println("Before Transaction");  // #1

	try {
		Object object = jp.proceed(); // #2
		System.out.println("DATA IS :: "+object);

	} catch (Throwable e) {
			e.printStackTrace();
	}

	System.out.println("After Transaction"); #4
}

@Component
public class Employeerunner implements CommandLineRunner {

	@Autowired
	private EmployeeDao dao;

	@Override // main---->employeeRunner.run() -----> continue with buisness
```

```
logic
      public void run(String... args) throws Exception {
            dao.saveEmployee();
      }
}

@SpringBootApplication
public class SpringAopAspectJAppApplication {

      public static void main(String[] args) {
            SpringApplication.run(SpringAopAspectJAppApplication.class, args);
      }
}

Output
======
Before Transaction
Employee Object is saved...
DATA IS :: 10
After Transaction
                                    refer:: Spring-AOP-AspectJApp



Application of AOP
=================
public interface IEmployeeService{
      public int saveEmployee(Employee employee);
}

@Service("empService")
public class EmployeeServiceImpl implements IEmployeeService{

      @Autowired
      private IEmployeeDao dao;

      @Transactional // Real Time usage of AOP
      public int saveEmployee(Employee employee){
                  dao.save(Employee);  // internally jpa--->
Hibernate[BeforeTransaction,....,AfterTransaction]
      }
}

public interface IEmployeeDao implements JpaRepository<Employee,Integer>{

}

@Repository
@Transactional(readOnly = true)
public class SimpleJpaRepository<T, ID> implements JpaRepositoryImpl<T,ID>{

      @Transactional
      @Override
      public <S extends T> S save(S entity) {

            Assert.notNull(entity, "Entity must not be null.");

            if (entityInformation.isNew(entity)) {
                  em.persist(entity); //save operation
```

```
                return entity;
            } else {
                return em.merge(entity);//update operation
            }
      }
}
```

Another application is SpringRest-GlobalException Handler[@RestControllerAdvice]

==========

```
@RestController
@RequestMapping("/api/ticket")
public class TicketRestController{

      @GetMapping("/find/{id}")
      public ResponseEntity<Ticket> findById(@Pathvariable Integer id){
                //if exception is recieved from Service<----Dao
      }
}

@RestController
@RequestMapping("/api/student")
public class StudentRestController{

      @GetMapping("/find/{id}")
      public ResponseEntity<Student> findById(@Pathvariable Integer id){
                //if exception is recieved from Service<----Dao
      }
}

@RestController
@RequestMapping("/api/employee")
public class EmployeeRestController{

      @GetMapping("/find/{id}")
      public ResponseEntity<Employee> findById(@Pathvariable Integer id){
                //if exception is recieved from Service<----Dao
      }
}

@RestControllerAdvice
public class GlobalExceptionHandlerAdvice{

      @ExceptionHandler(TicketNotfoundException.class)
      public ResponseEntity<ErrorDetails>
handleTicketNotFoundException(TicketNotFoundException te){
                ;;;;;
      }

      @ExceptionHandler(EmployeeNotFoundException.class)
      public ResponseEntity<ErrorDetails>
EmployeeNotFoundException(EmployeeNotfoundException ef){
                ;;;;;
      }

      @ExceptionHandler(StudentNotFoundException.class)
      public
ResponseEntity<ErrorDetails>StudentNotFoundException(StudentNotFoundException sf){
                      ;;;;;
      }
```

```
	@ExceptionHandler(Exception.class)
	public ResponseEntity<ErrorDetails> handleOtherException(Exception e){
			;;;;;
	}
}
```

Usage of custom annotation to support AOP
==========

```
@Aspect
@Component
public class TransactionManagement {

	@Pointcut("@annotation(in.ineuron.annotation.MyTransaction)")
	public void p1() {
	}

	@Before("p1()")
	public void beforeTransaction() {
		System.out.println("***Before Transaction ***");
	}

	@After("p1()")
	public void afterTransaction() {
		System.out.println("***After Transaction ***");
	}
}

@Retention(RUNTIME) //It indicates where the effect of annotation should be
available.
@Target(METHOD) //It indicates where this annotation can be used in
code[class/Field/Method].
public @interface MyTransaction {

}
```

EmployeeDao.java
==========

```
@Component
public class EmployeeDao {

	@MyTransaction
	public void saveEmployee() {
		System.out.println("Employee Object is saved...");
	}
}
```

Output

```
***Before Transaction ***
Employee Object is saved...
***After Transaction ***
```

refer:: Spring-AOP-AspectJApp2

Runners in SpringBoot
==========

=> Runners java classes cum springbeans of Springboot app implementing

xxxRunner(I) directly or indirectly.
 => EveryRunner class contains run()dealing with one time execution logic and these logics will execute where SpringApplication.run() is about to complete
    all its startup activities[Right after Creating ApplicationContext object and Completing Pre-Instantiation and Injections].
 => The run() of every Runner class will be executed automatically by IOC container only for one time as a part of Application startup process
    that takes place in SpringApplication.run() method.

Two types of Runners in SpringBoot
==================================
 1. CommandLineRunner(I)[1.0V]
 2. ApplicationRunner(I)[1.3V]

What is the difference b/w CommandLineRunner and ApplicationRunner?
  Both the runners are giving run() as the callback method(Becoz it will be called by underlying IOC container automatically)
  Both the runners run() method get command line args(programs args) as the parameter values,but the way they get them is different.

CommandLineRunner(I)
    |=> public void run(String... args)

ApplicationRunner(I)
    |=> public void run(ApplicationArguments args)

What is the usage of Runners in SpringBoot Apps?
 => Executing the dbscripts and keeping table with row ready
 => Perform some testing activities related to AutoConfiguration and its injections.
 => Logging activities related to AutoConfiguration and startup activities
 => To send email or sms alter messages to admin when imp services started during the startup process
    of Spring Boot application,start of server, initialization of buisness methods,smsgateway,payement gateway
 => Though there are multiple modules in Project, if u want to activate only certain modules we can take the support of Runners
 => To create dummy/admin email ids,dummy or admin accounts during the startup process automaticlly we take support of Runners.

Note: In realtime project the main class(Configuration class main class that contains @SpringBootApplication) main() method contains
      only SpringApplication.run() method call.
      The remaining logics like getting SpringBean class object IOC container, invoking b.method and etc ...
      Now this logic can be done from run() of runner class through various injections.


```
public class SpringApplication{
      public ConfigurableApplicationContext run(String... args) {


            long startTime = System.nanoTime();
            DefaultBootstrapContext bootstrapContext = createBootstrapContext();
            ConfigurableApplicationContext context = null;
            configureHeadlessProperty();
            SpringApplicationRunListeners listeners = getRunListeners(args);
```

```
            listeners.starting(bootstrapContext, this.mainApplicationClass);

            try {
                  ApplicationArguments applicationArguments = new
DefaultApplicationArguments(args);
                  ConfigurableEnvironment environment =
prepareEnvironment(listeners, bootstrapContext, applicationArguments);
                  configureIgnoreBeanInfo(environment);

                  Banner printedBanner = printBanner(environment);

                  context = createApplicationContext();
                  context.setApplicationStartup(this.applicationStartup);
                  prepareContext(bootstrapContext, context, environment, listeners,
applicationArguments, printedBanner);
                  refreshContext(context);
                  afterRefresh(context, applicationArguments);
                  Duration timeTakenToStartup = Duration.ofNanos(System.nanoTime()
- startTime);
                  if (this.logStartupInfo) {
                        new
StartupInfoLogger(this.mainApplicationClass).logStarted(getApplicationLog(),
timeTakenToStartup);
                  }
                  listeners.started(context, timeTakenToStartup);
                  callRunners(context, applicationArguments);
            }
            catch (Throwable ex) {
                  handleRunFailure(context, ex, listeners);
                  throw new IllegalStateException(ex);
            }
            try {
                  Duration timeTakenToReady = Duration.ofNanos(System.nanoTime() -
startTime);
                  listeners.ready(context, timeTakenToReady);
            }
            catch (Throwable ex) {
                  handleRunFailure(context, ex, null);
                  throw new IllegalStateException(ex);
            }
            return context;
      }
}
```

Note:: We can have multiple runners in our project, if mulitple runners are available then based on Alphabetical order of
class names the runners will run.
If we want a particular runner to run then we need to use @Order.
Syntax :: @Order(any-no)

In case of Commandline arguments the data will be stored in the form of String[]
Commandline Arguments :: [START, LOAD, END]

refer:: SpringBoot-Runner-01

Note::

```
@FunctionalInterface
public interface ApplicationRunner {
```

```
      void run(ApplicationArguments args) throws Exception;
}

@FunctionalInterface
public interface CommandLineRunner {
      void run(String... args) throws Exception;
}
```