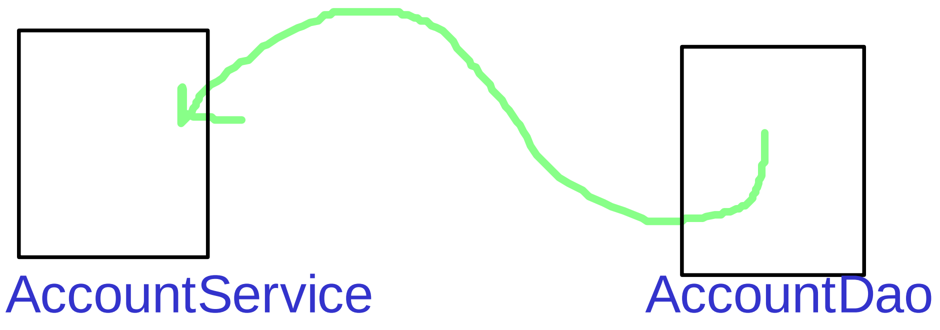
Spring AOP

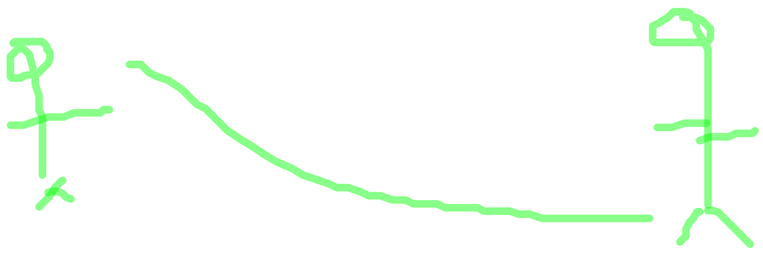
Rajeev Gupta MTech CS Java
Trainer & Consultant

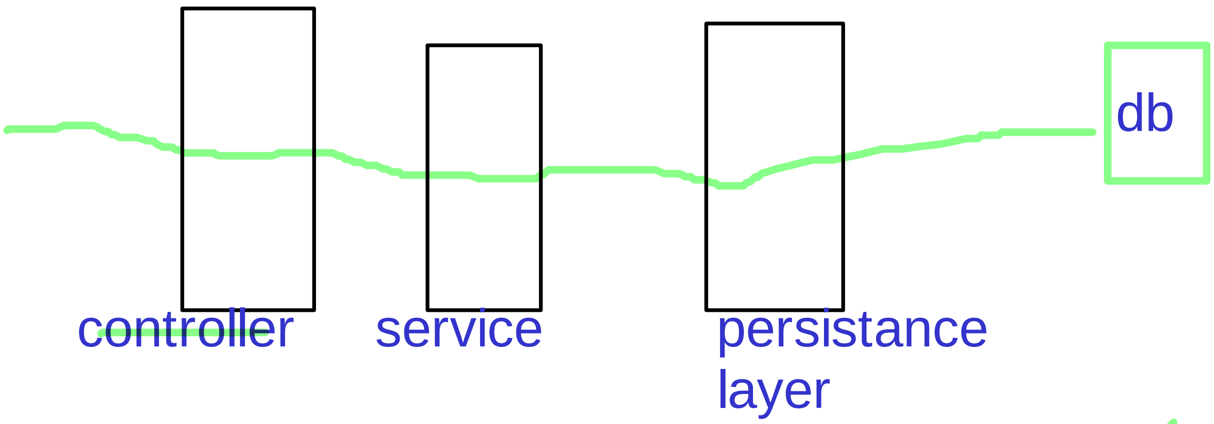
What so far

1. Depdendeny Injection DI



2. AOP(Aspect oriented programming)





service layer = Business logic + Cross cutting conern ✓

ccc

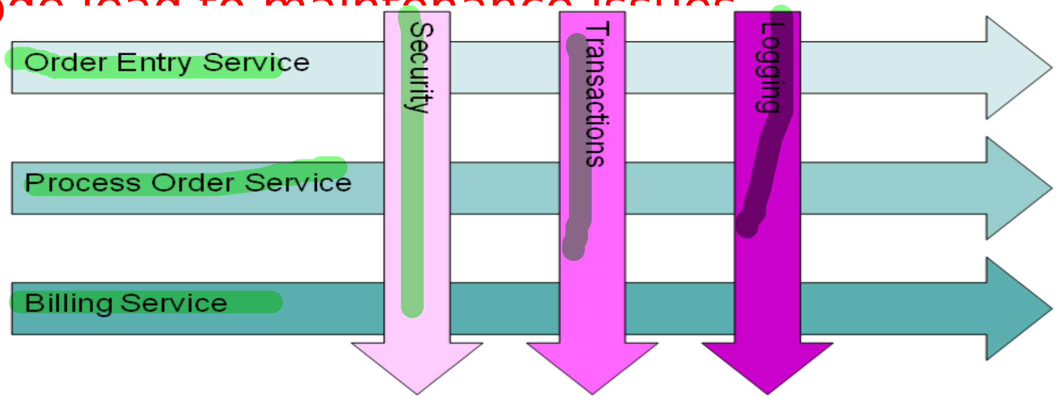
Example of CCC:

Logging
Caching
Tx Mgt
Security
.....

Logging : Log4j framewrok

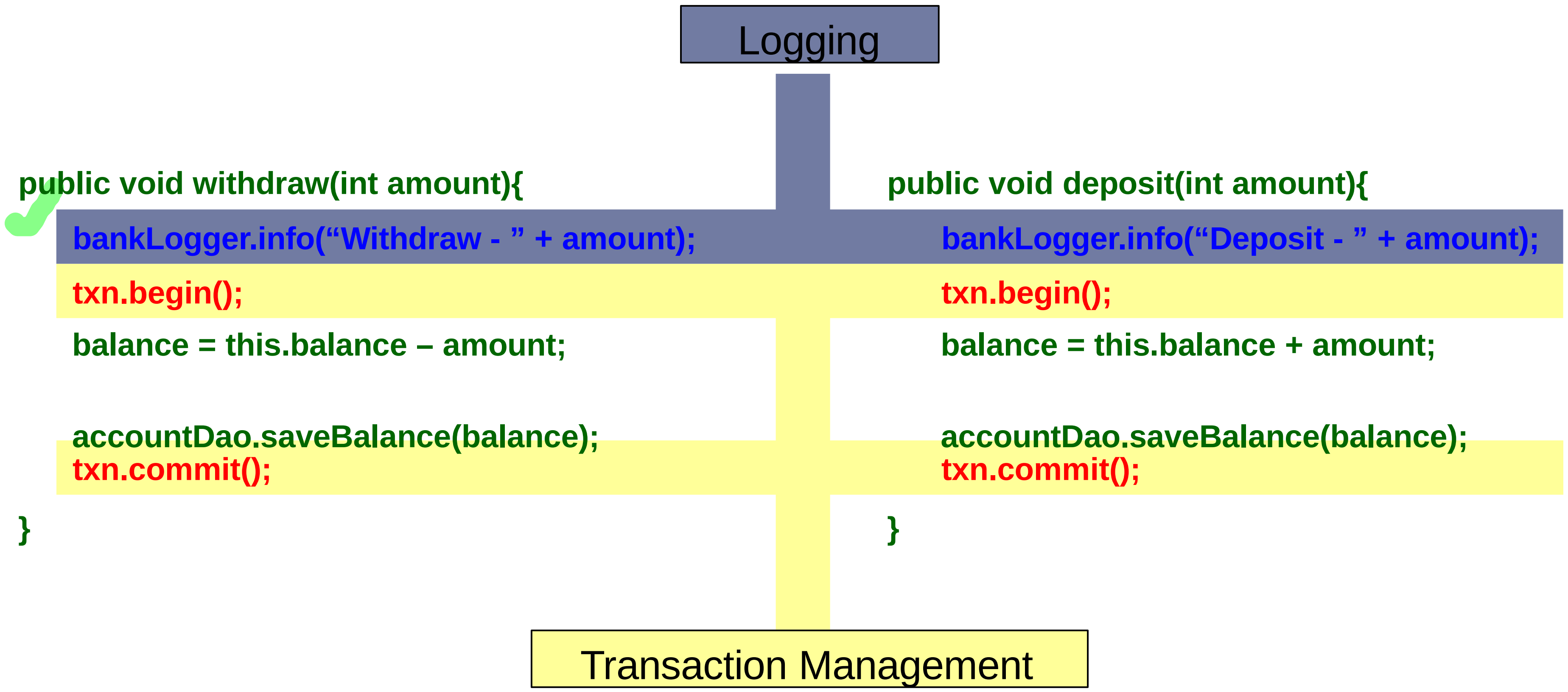
Introduction to AOP

- What is AOP?
  - AOP is a style of programming, mainly good in separating cross cutting concerns
- How AOP works?
  - Achieved usages Proxy design Pattern to separate CCC's form actual code
  - Cross Cutting Concern ?
  - Extra code mixed with the actual code is called CCC's Extra
  - code mixed with code lead to maintenance issues
    - **Logging**
    - **validations**
    - **Auditing**
    - **Security**



Crosscutting Concerns

- Eg: Banking Application



```
public void withdraw(int amount){
    bankLogger.info(“Withdraw - ” + amount);
    txn.begin();
    balance = this.balance – amount;

    accountDao.saveBalance(balance);
    txn.commit();
}
```

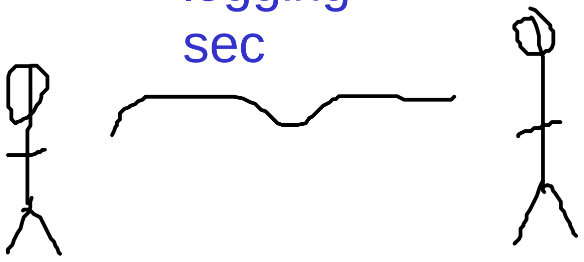
```
public void deposit(int amount){
    bankLogger.info(“Deposit - ” + amount);
    txn.begin();
    balance = this.balance + amount;

    accountDao.saveBalance(balance);
    txn.commit();
}
```

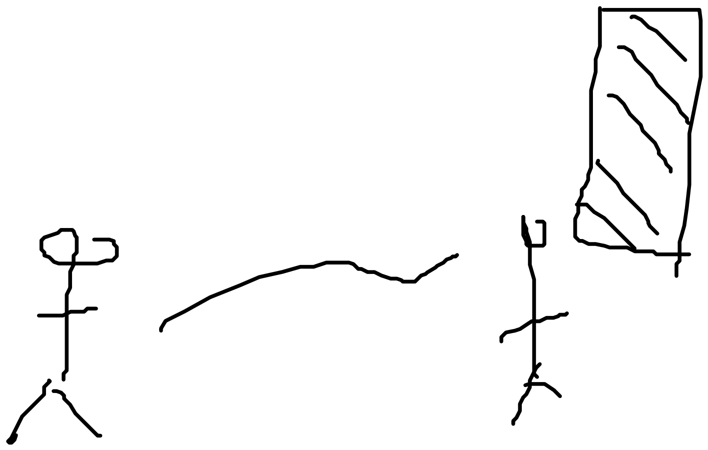
Aspect =advice + pointcut

what when/where

logging

sec





```
@Aspect
@Service
public class LoggingAspect {
	private Logger logger=LoggerFactory.getLogger(AccountServiceImpl.class);

	@Pointcut("execution(public void transfer(..))")
	public void loggingPointCut() {}

	@Around("loggingPointCut()")
	public Object around(ProceedingJoinPoint pjp)throws Throwable {
		long start =System.currentTimeMillis();

		Object value=pjp.proceed();

		long end  =System.currentTimeMillis();

		logger.info("time taken to transfer method: "+(end-start)+" ms");

		return value;
	}
}
```

```
@Service(value = "as")
public class AccountServiceImpl implements AccountService{

	private AccountDao accountDao;

	@Override
	public void transfer(int fromAcc, int toAcc, double amount) {
		Account fromAccount=getById(fromAcc);
		Account toAccount=getById(toAcc);

		fromAccount.setBalance(fromAccount.getBalance()-amount);
		toAccount.setBalance(toAccount.getBalance()+amount);

		accountDao.updateAccount(fromAccount);
		accountDao.updateAccount(toAccount);
	}
}
```

merge

AOP: actual code is merged with ccc code
u dont have to write much code

Advice +pointcut

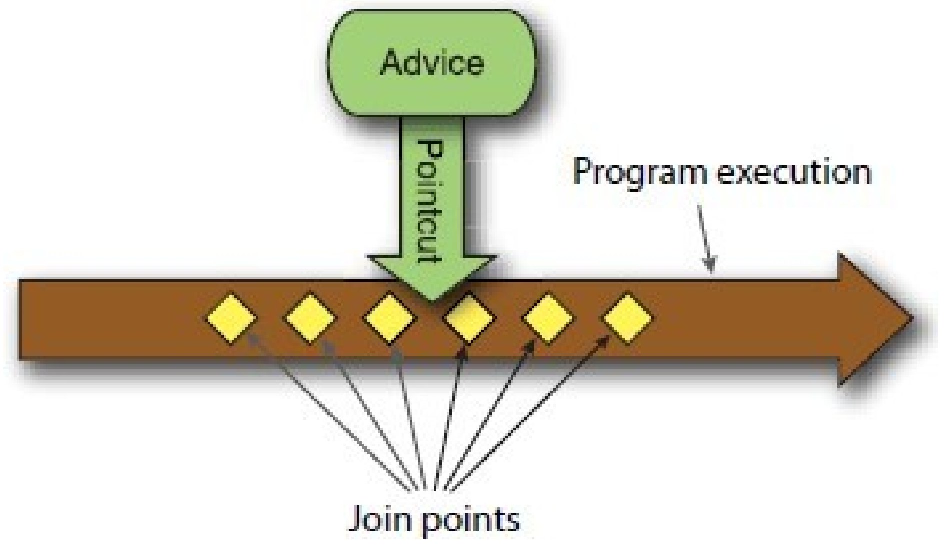
Understanding AOP terminology

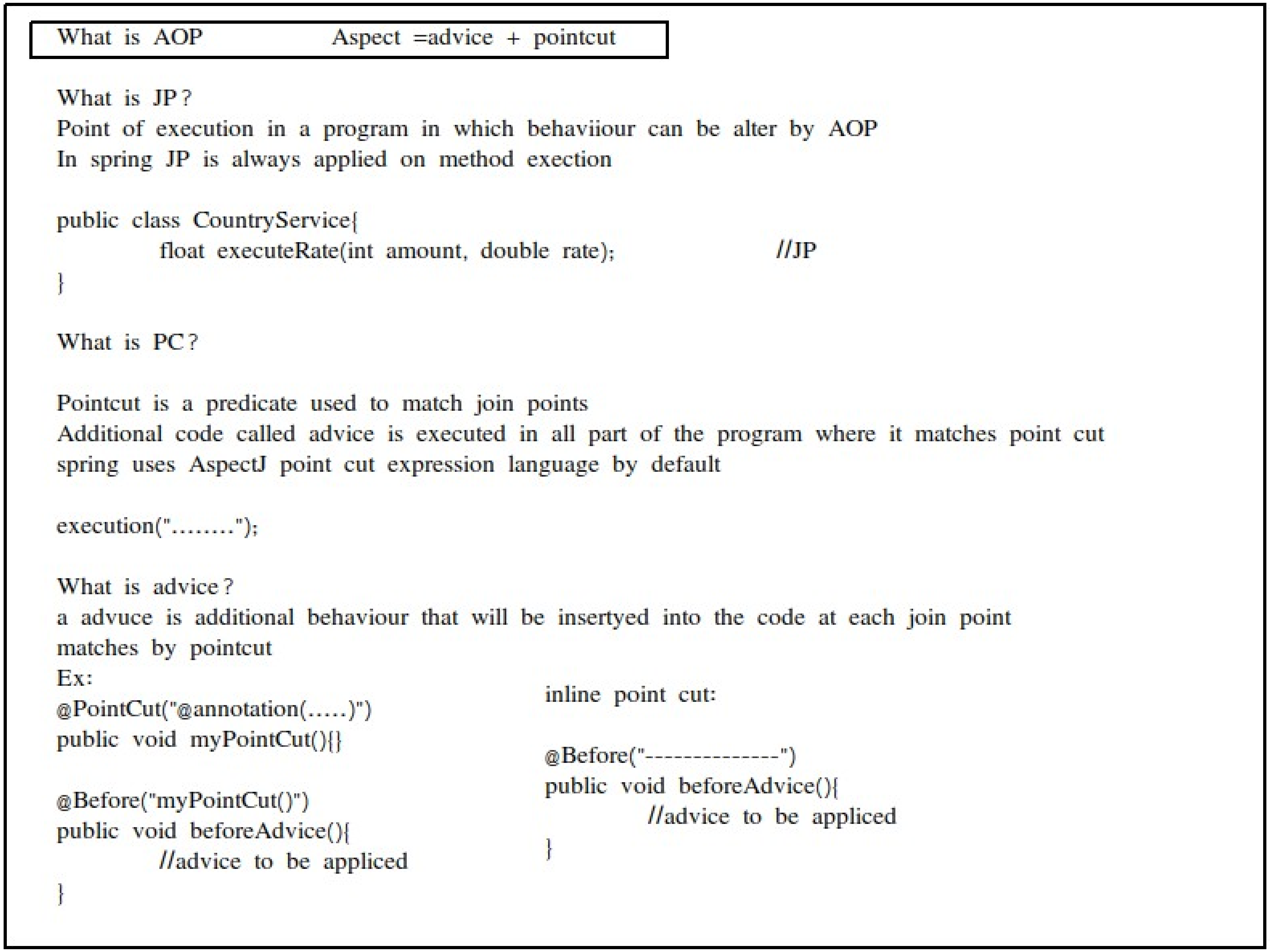
Join points : are the options on the menu and
pointcuts : are the items you select

Aspect =Advices + Point Cut

AOP

Aspect means
what (extra logic) and where it need to be applied (point cut)





What is AOP   Aspect =advice + pointcut

What is JP ?
Point of execution in a program in which behaviiour can be alter by AOP
In spring JP is always applied on method exection

```
public class CountryService{
        float executeRate(int amount, double rate);             //JP
}
```

What is PC ?

Pointcut is a predicate used to match join points
Additional code called advice is executed in all part of the program where it matches point cut
spring uses AspectJ point cut expression language by default

```
execution("........");
```

What is advice ?
a advuce is additional behaviour that will be insertyed into the code at each join point matches by pointcut
Ex:

```
@PointCut("@annotation(.....)")
public void myPointCut(){}

@Before("myPointCut()")
public void beforeAdvice(){
        //advice to be applied
}
```

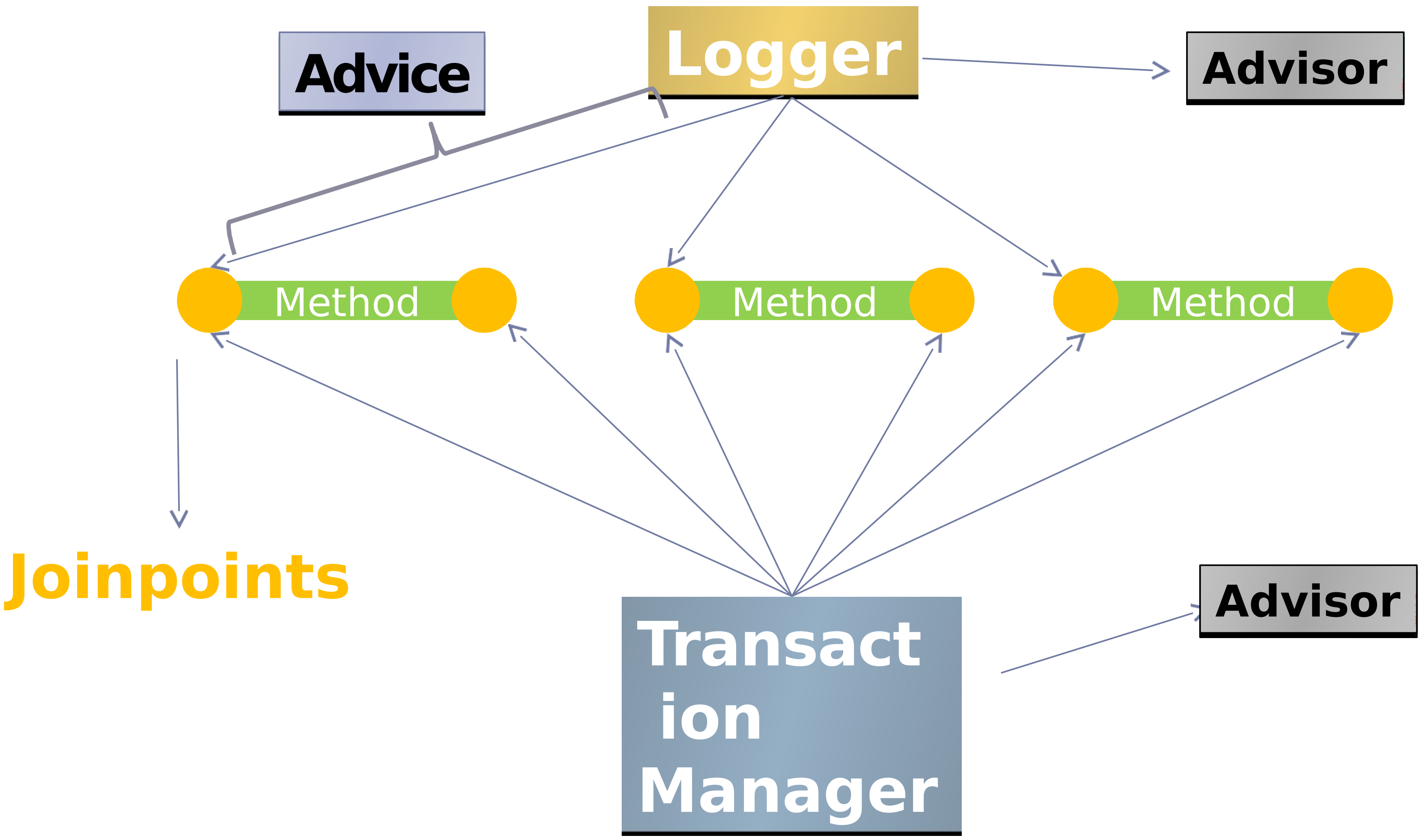
inline point cut:

```
@Before("--------------")
public void beforeAdvice(){
        //advice to be applied
}
```

AOP – Definitions.

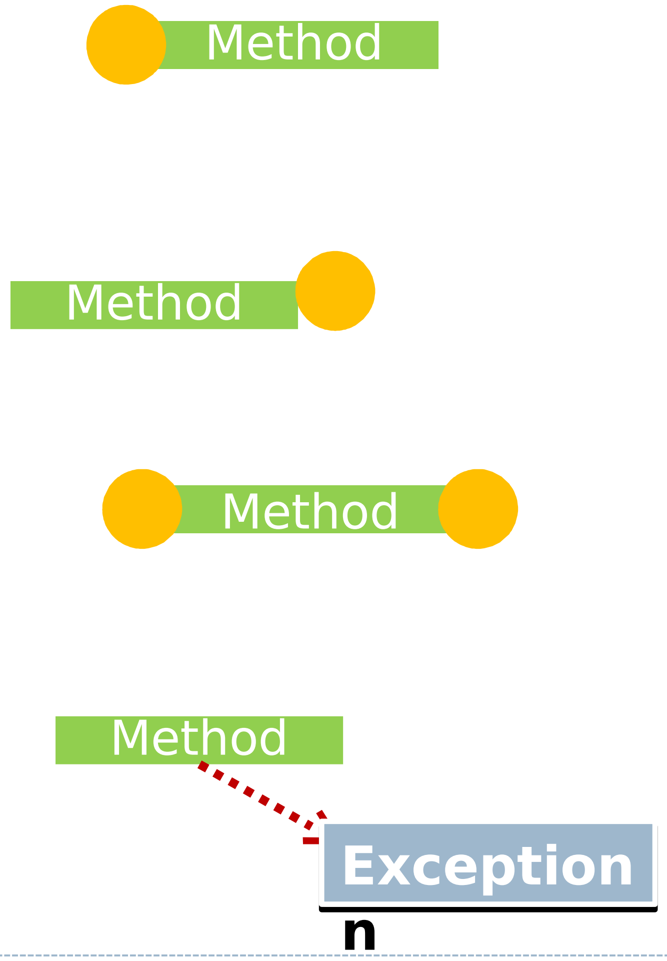
- **Aspect**
- **Joinpoint**
- **Advice**
- **Pointcut**
- **Target Object**
- **AOP Proxy**
- **Weaving**

AOP – Definitions.



Advice Types

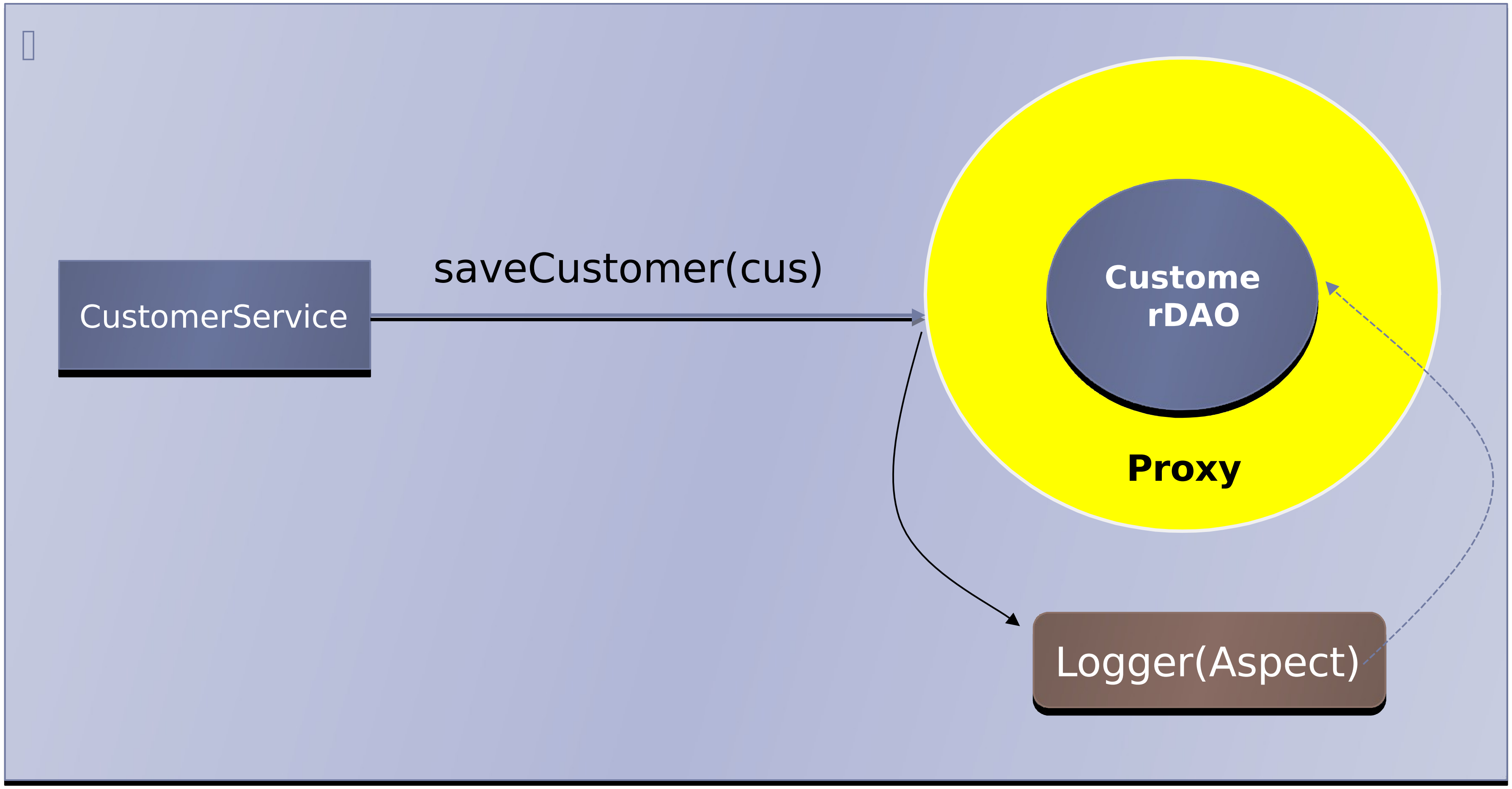
- Before Advice
- After returning Advice
- Around Advice
- Throws Advice



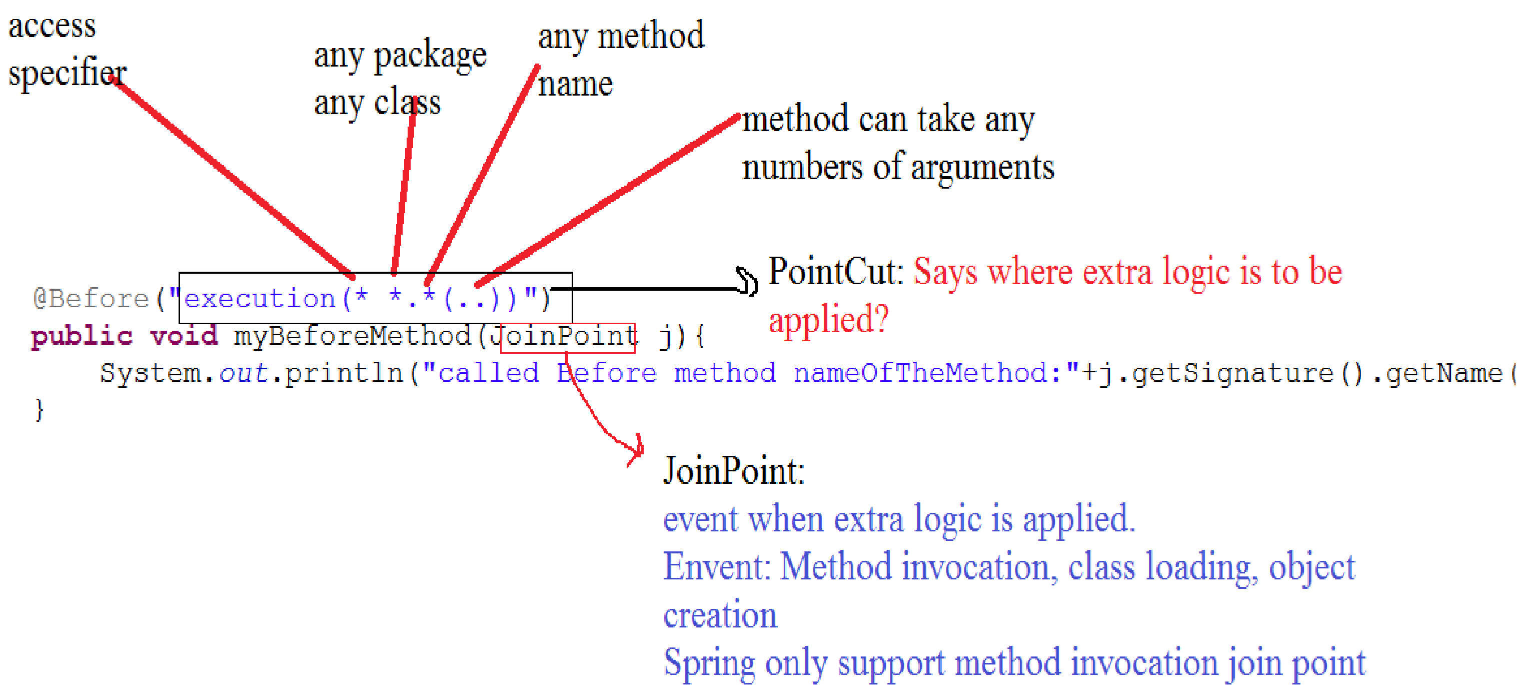
WEAVING

- Weaving is the process of applying aspects to a target object to create a new proxied object. The aspects are woven into the target object at the specified join points. The weaving can take place at several points in the target object’s lifetime:
  - **Compile time** —Aspects are woven in when the target class is compiled.
  - **Classload time** —Aspects are woven in when the target class is loaded into the JVM.
  - **Runtime** —Aspects are woven in sometime during the execution of the applica- tion. Typically, an AOP container will dynamically generate a proxy object that will delegate to the target object while weaving in the aspects.

AOP Weaving


CustomerService
saveCustomer(cus)
Custome rDAO
Proxy
Logger(Aspect)

Understanding Point Cut wildcard



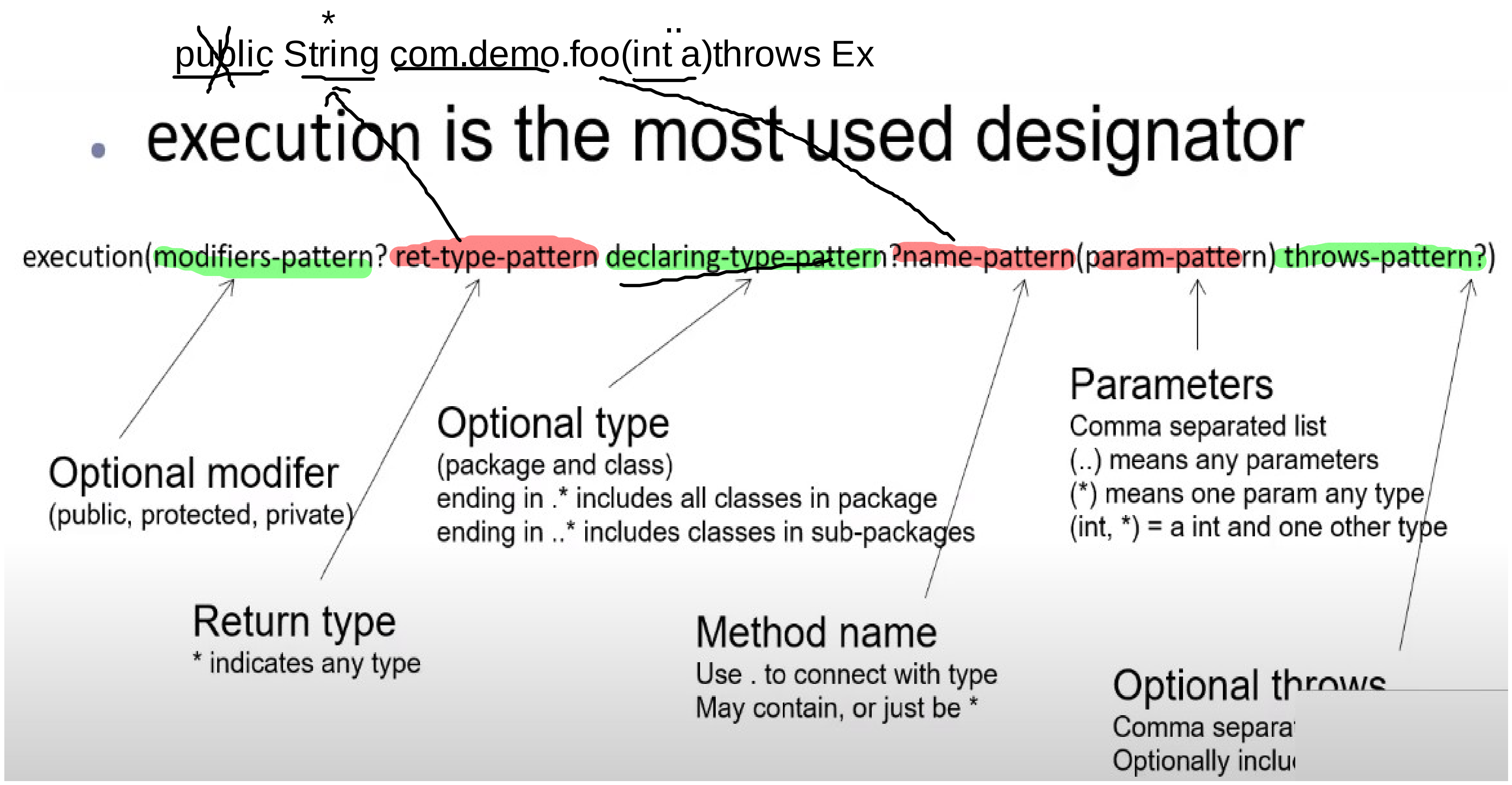
```
@Before("execution(* *.*(..))")
public void myBeforeMethod(JoinPoint j){
    System.out.println("called Before method nameOfTheMethod:"+j.getSignature().getName(
}
```

PointCut: Says where extra logic is to be applied?

JoinPoint:
event when extra logic is applied.
Envent: Method invocation, class loading, object creation
Spring only support method invocation join point

Understanding Point Cut wildcard

public String com.demo.foo(int a)throws Ex



- execution is the most used designator

execution(modifiers-pattern? ret-type-pattern declaring-type-pattern?name-pattern(param-pattern) throws-pattern?)

Optional modifer
(public, protected, private)

Return type
* indicates any type

Optional type
(package and class)
ending in .* includes all classes in package
ending in ..* includes classes in sub-packages

Method name
Use . to connect with type
May contain, or just be *

Parameters
Comma separated list
(..) means any parameters
(*) means one param any type
(int, *) = a int and one other type

Optional throws
Comma separa
Optionally inclu