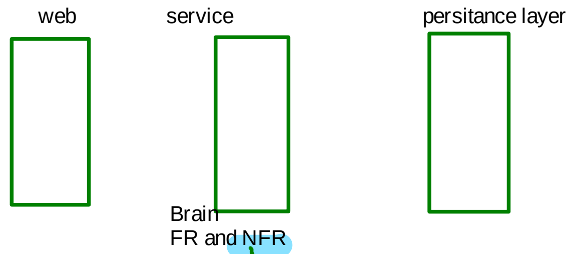
Spring AOP

Rajeev Gupta MTech CS Java
Trainer & Consultant

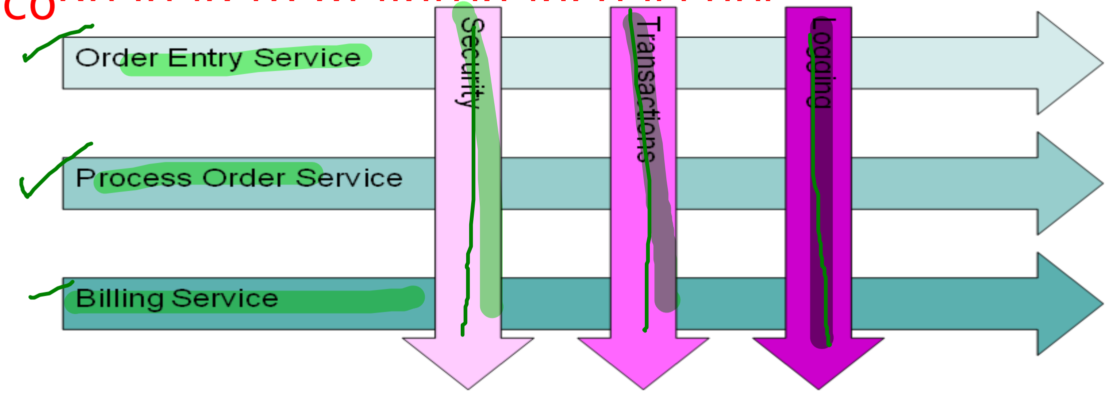
```xml
<bean id="bank" class="com.demo.Bank">
        <property name="bankName" value="SBI Delhi"/>

        <property name="accounts">
                <list>
                        <ref bean="acc1"/>
                        <ref bean="acc2"/>
                        <ref bean="acc3"/>
                </list>

        </property>

        <property name="customerCount">
                <map>
                        <entry key="SBI Preet vihar" value="2000"></entry>
                        <entry key="SBI Kr vihar" value="2090"></entry>
                </map>
        </property>

        <property name="branches">
                <map>
                        <entry key="SBI Preet vihar" value-ref="bl1"></entry>
                        <entry key="SBI Kr vihar" value-ref="bl2"></entry>
                </map>

        </property>

        <property name="branchManagers">
                <props>
                        <prop key="SBI Preet vihar">Mr Ramesh Kr </prop>
                        <prop key="SBI Kr vihar">Mr Kapil Gupta </prop>
                </props>
        </property>
    </bean>

    <bean id="bl1" class="com.demo.BranchLocation">
        <property name="address" value="SBI Preet vihar Delhi 91"/>
        <property name="city" value="Delhi"/>
    </bean>


    <bean id="bl2" class="com.demo.BranchLocation">
        <property name="address" value="SBI Kr vihar Delhi 51"/>
        <property name="city" value="Delhi"/>
    </bean>

    <bean id="acc1" class="com.demo.Account">
        <property name="id" value="33"/>
        <property name="name" value="sumit"/>
        <property name="balance" value="5000"/>
    </bean>

    <bean id="acc2" class="com.demo.Account">
        <property name="id" value="3"/>
        <property name="name" value="rajat"/>
        <property name="balance" value="9000"/>
    </bean>


    <bean id="acc3" class="com.demo.Account">
        <property name="id" value="330"/>
        <property name="name" value="ekta"/>
        <property name="balance" value="50000"/>
    </bean>
```

Aspect OP



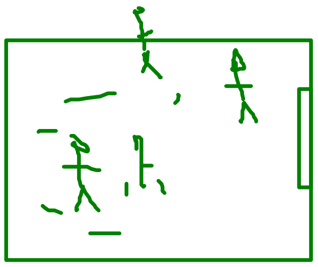
Introduction to AOP

- 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**



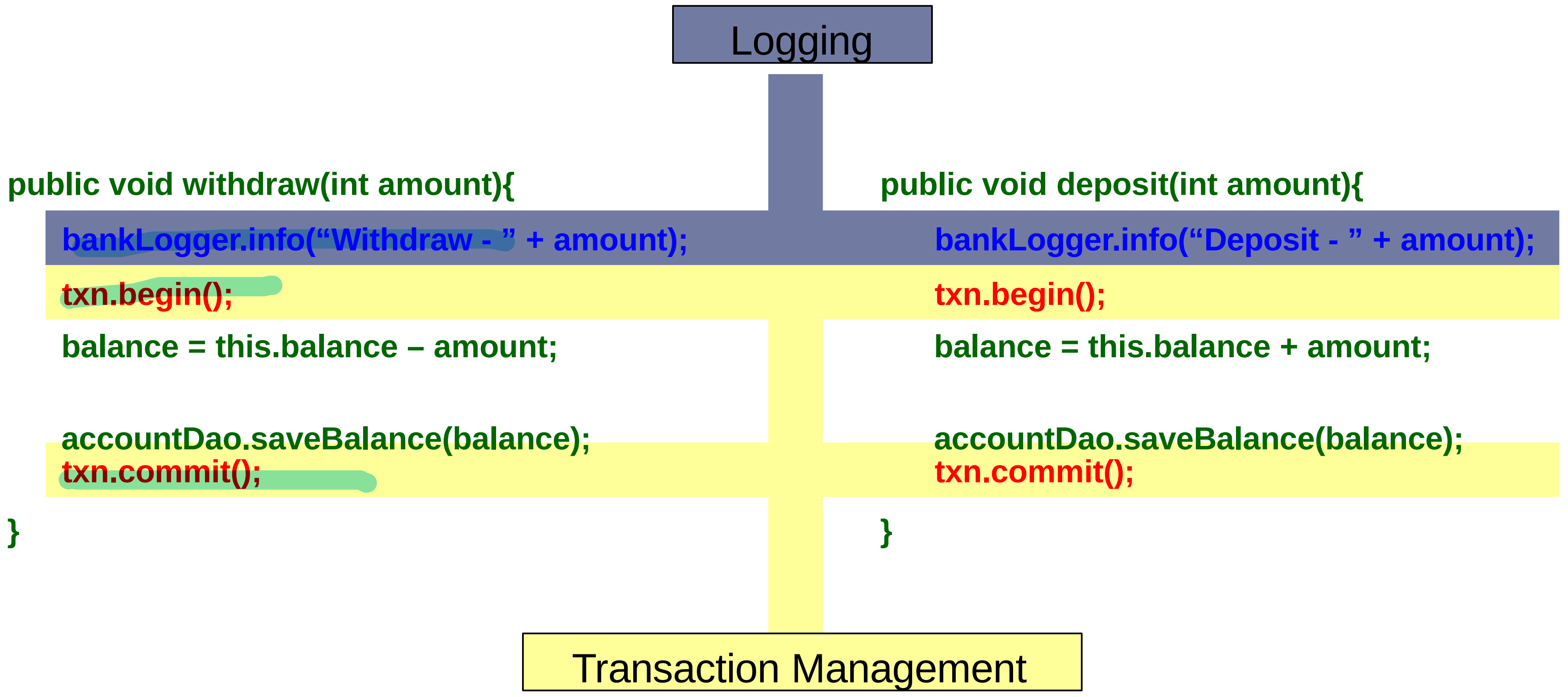
Weaving

Cross cutting concern



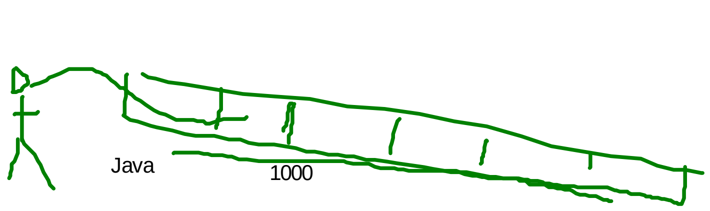
Crosscutting Concerns

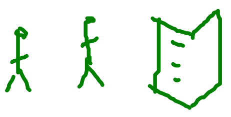
- Eg: Banking Application

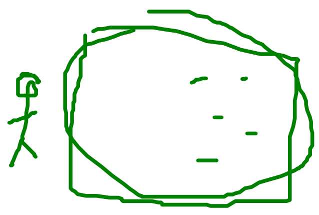


```
public void withdraw(int amount){
    bankLogger.info("Withdraw - " + amount);
    txn.begin();
    balance = this.balance – amount;

    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();
}
```







i could teach 1000-------------------> 10 people

JP PC(is subset of JP)

//JP (Joint point) and PC(point cut)

```
@Component(value = "magician")
 public class Magician {

    public void doMagic() {
        System.out.println("abra ka dabra...");
    }

    public void eat() {
        System.out.println("abra ka dabra...");
    }

    public void bath() {
        System.out.println("abra ka dabra...");
    }

    public void playingWithFamily() {
        System.out.println("abra ka dabra...");
    }
}
```

```
@Component
@Aspect
public class AudianceAspect {

    @Pointcut("execution(public void doMagic())")
    public void myPointCut() {}

    @After("myPointCut()")
    public void clapping() {
        System.out.println(" wow maza aa gaya");
    }
}
```

JP: point of ex in a program where behaviour can be alter using aop

PC: predicate used to match JP

JP > PC

```
@Component(value = "magician")
 public class Magician {

        public void doMagic() {
                System.out.println("abra ka dabra...");
        }

}
```

```
@Component
@Aspect
public class AudianceAspect {


        @After("execution(public void doMagic())")
        public void clapping() {
                System.out.println(" wow maza aa gaya");
        }
}
```

```
<context:annotation-config/>
<context:component-scan base-package="com.demo"/>

<!-- how to integrate spring with AspectJ -->
<aop:aspectj-autoproxy/>
```

```
ApplicationContext ctx=new ClassPathXmlApplicationContext("beans.xml");
        Magician magician=ctx.getBean("magician",Magician.class);
        magician.doMagic();
```

```
public class Magician$$EnhancerBySpringCGLIB$$cbe950e9 extends Magicain {

        public void doMagic() {
                super.doMagic();
                        System.out.println(" wow maza aa gaya");
        }

}
```

Type of advices:

before
after

after returing

after throwing

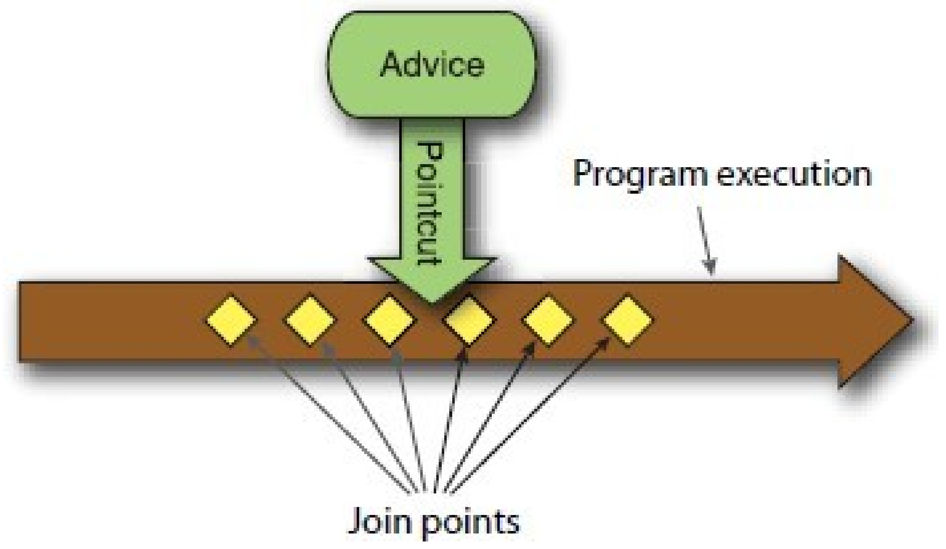
around

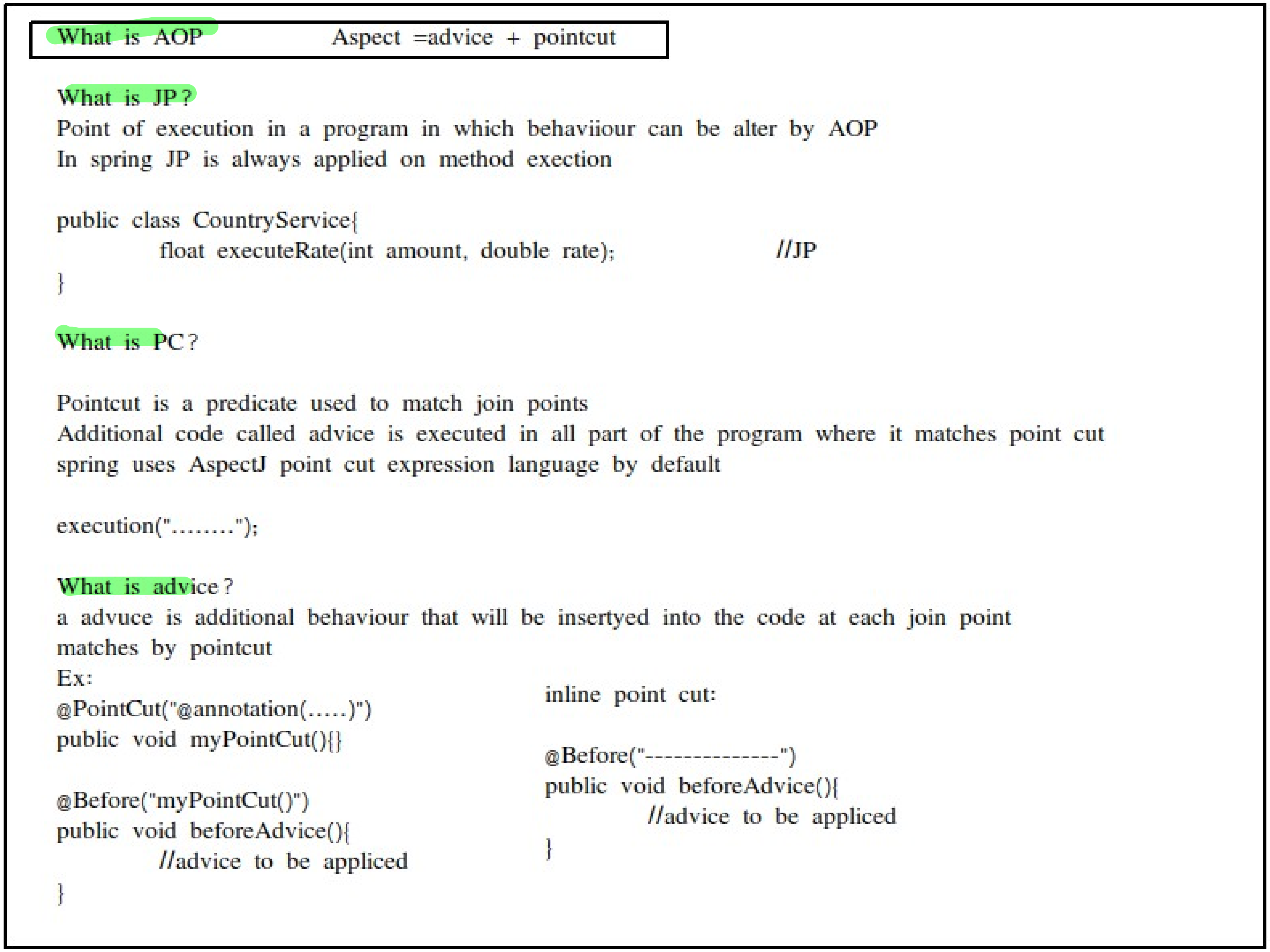
Understanding AOP terminology

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

Aspect =Advices + Point Cut

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 applyed
}
```

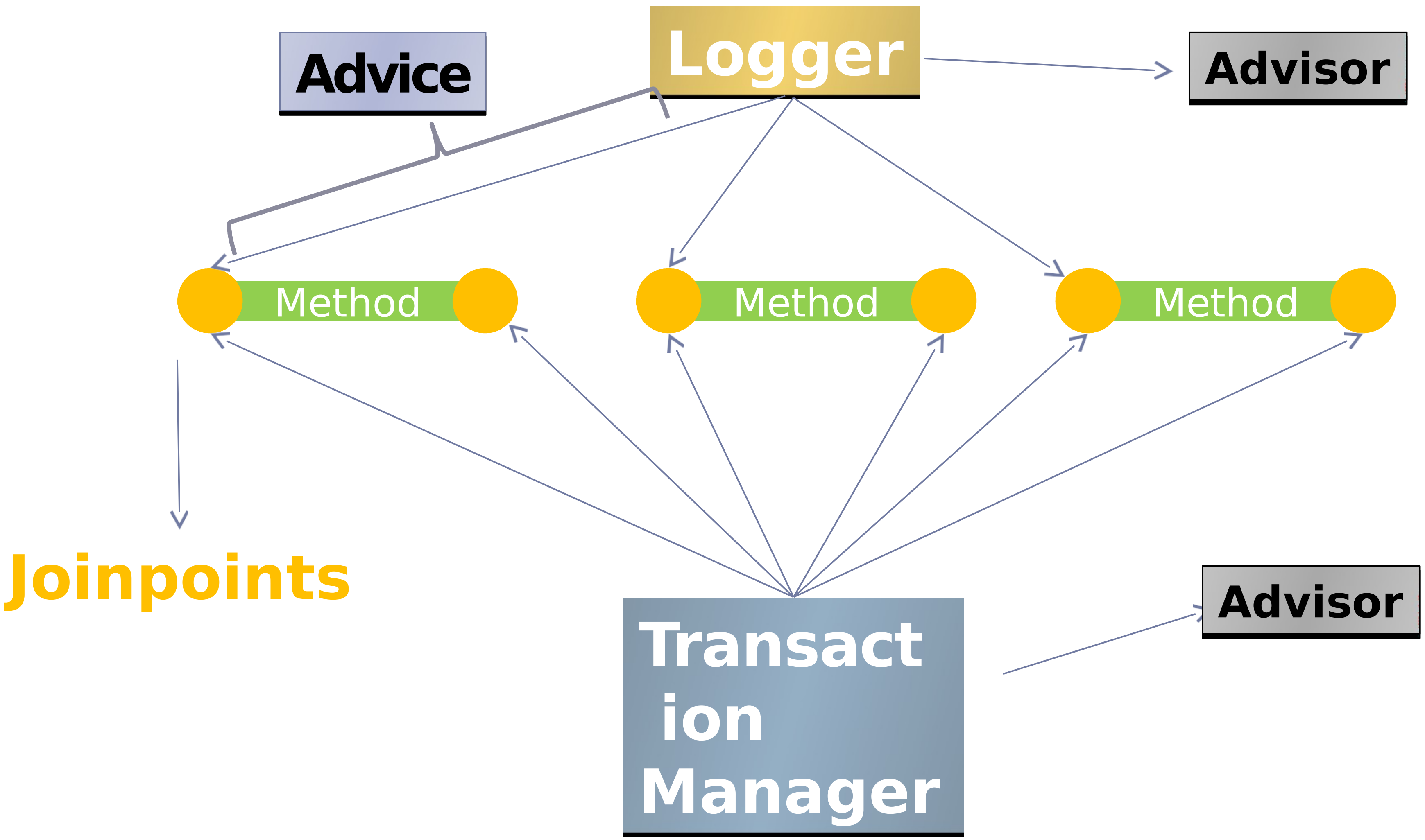
inline point cut:

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

AOP – Definitions.

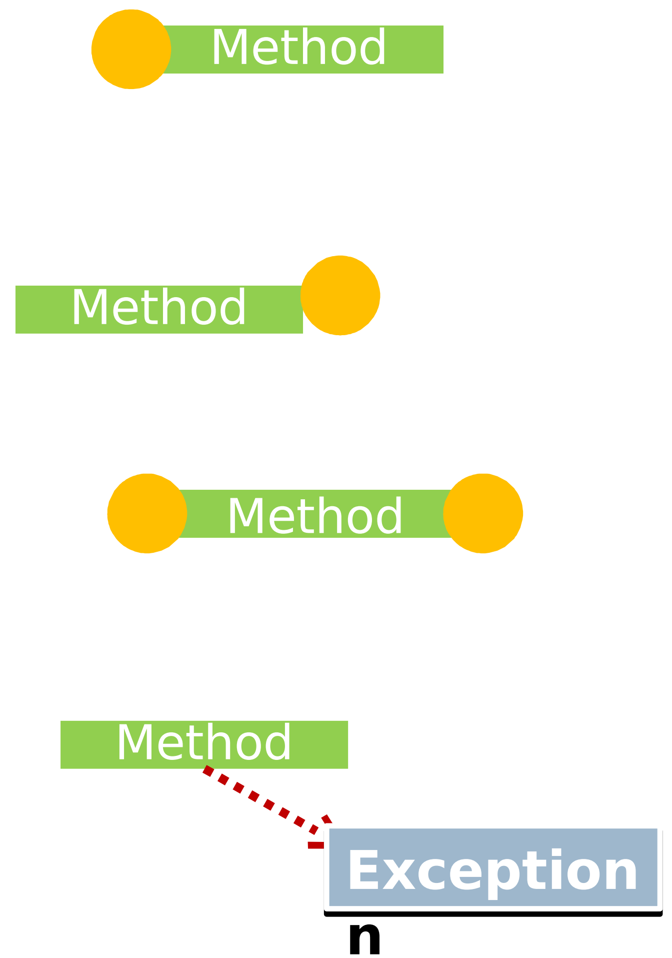
- **Aspect**
- **Joinpoint**
- **Advice**
- **Pointcut**
- **Target Object**
- **AOP Proxy**
- **Weaving**

AOP – Definitions.



Advice Types

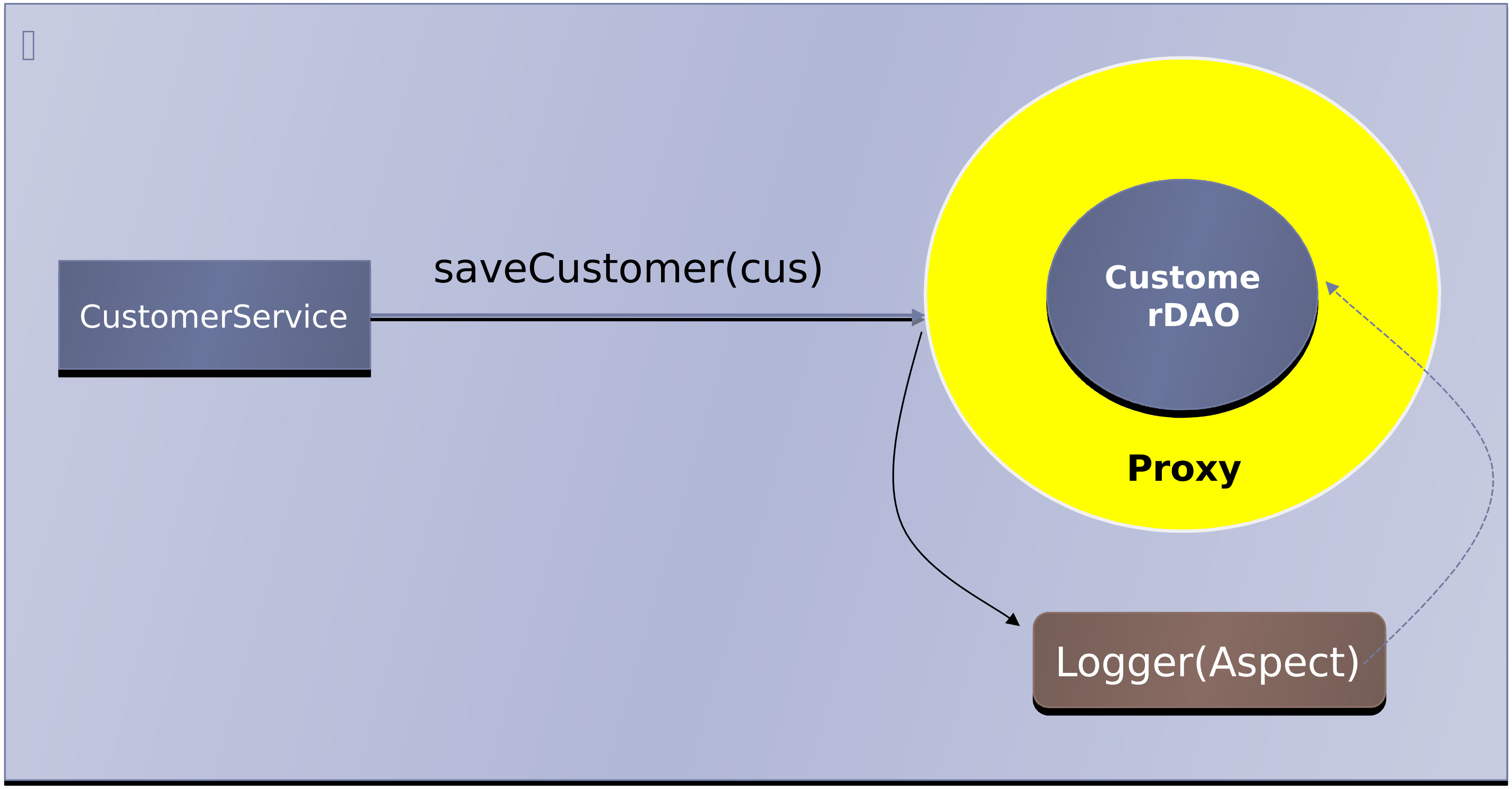
- Before Advice
- After returning Advice
- Around Advice
- Throws Advice



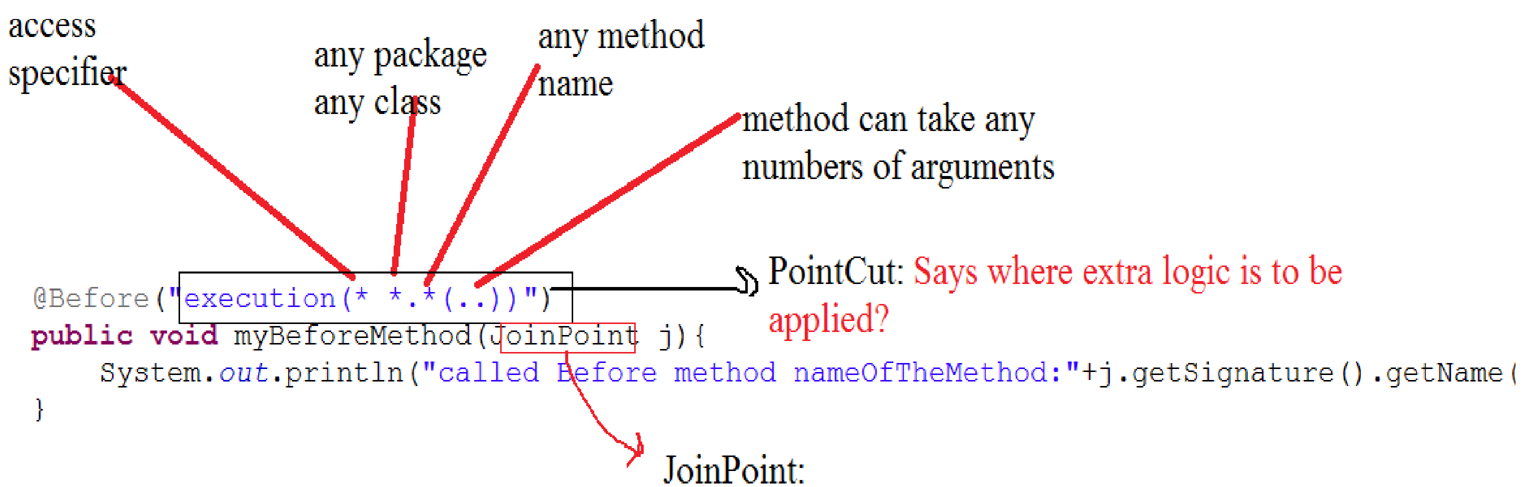
WEAVING

- 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)
CustomerDAO
Proxy
Logger(Aspect)

Understanding Point Cut wildcard

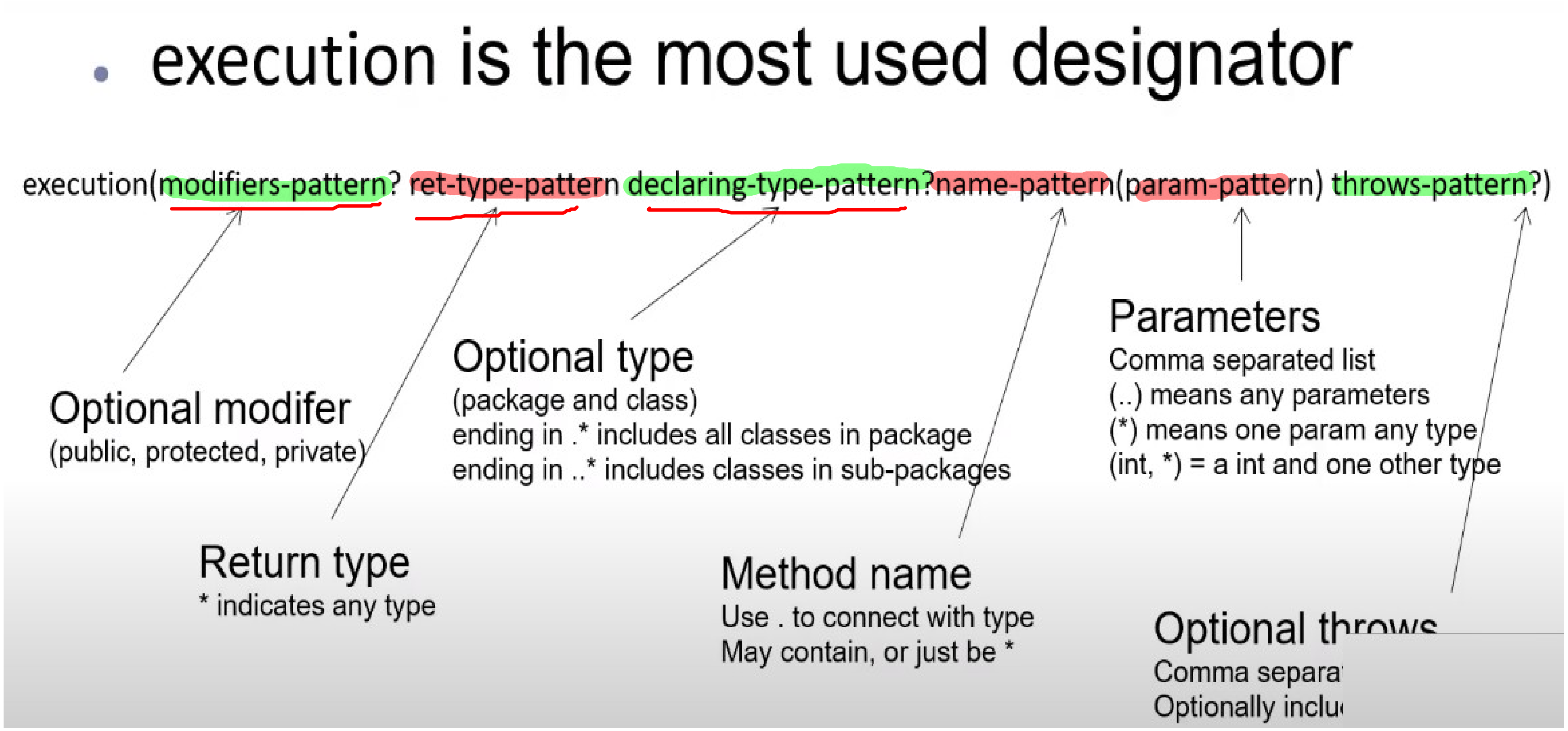


```
@Before("execution(* *.*(..))")
public void myBeforeMethod(JoinPoint j){
    System.out.println("called Before method nameOfTheMethod:"+j.getSignature().getName(
}
```

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



- 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