**What is Annotation**

1. Annotations are meta informations for a field, method, constructor or method arguments.
2. Annotations can be applied during compile time, runtime.
3. This provides the annotation processor (EX: AOP or custom annotation processor) what needs to be done on seeing the annotation.

**How Annotation Works (Example of method annotation)**

1. Declare an annotation. Ex: EmployeeEvent
2. Place the annotation in any class file. Ex: Employee
3. Define an Proxy that iterates all or specific classes for annotation EmployeeEvent. Ex: EmployeeProxy
   1. The Employee proxy will load Employee class using reflection.
   2. Gets the list of methods present.
   3. Iterates each method, verifies the EmployeeEvent Annotation. Does some custom logic and then invokes the method itself.
   4. Client needs to call the EmployeeProxy with method arguments.

Note: The main disadvantage here is all the method in the classes are iterated to verify that annotation is present or not.

Input (Green Colour)and Output (Black Colour)

1

Prabhu

SENIOR

30

Employee Proxy is called before Employee invocation

Employee Name is Prabhu

He/She is a Senior Member

Has Spend: 30

Remaining Budget Left are : 70

Employee Proxy is called after Employee invocation

**package** com.sample.basics;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

**import** java.lang.reflect.Method;

**import** java.util.Scanner;

@Target(ElementType.***METHOD***)

@Retention(RetentionPolicy.***RUNTIME***)

**@interface** EmployeeEvent {

String employeeRole() **default** "GUEST";

**int** budgetLimit() **default** 0;

}

**class** Employee {

String name;

Employee(String name) {

**this**.name = name;

}

@EmployeeEvent(employeeRole = "SENIOR", budgetLimit = 100)

**public** **void** seniorEmployeeBenefits(**int** budget, **int** moneySpend) {

System.***out***.println("Employee Name is " + name);

System.***out***.println("He/She is a Senior Member");

System.***out***.println("Has Spend: " + moneySpend);

System.***out***.println("Remaining Budget Left are : " + (budget - moneySpend));

}

@EmployeeEvent(employeeRole = "JUNIOR", budgetLimit = 50)

**public** **void** juniorEmployeeBenefits(**int** budget, **int** moneySpend) {

System.***out***.println("Employee Name is " + name);

System.***out***.println("He/She is a Junior Member");

System.***out***.println("Has Spend: " + moneySpend);

System.***out***.println("Remaining Budget Left are : " + (budget - moneySpend));

}

}

**class** EmployeeProxy {

**public** **static** **void** execute(Employee emp, String role, **int** spend) {

**try** {

Method[] methods = emp.getClass().getMethods();

**for** (Method method : methods) {

**if** (method.isAnnotationPresent(EmployeeEvent.**class**)) {

EmployeeEvent family = method

.getAnnotation(EmployeeEvent.**class**);

String userRole = family.employeeRole();

**int** userBudget = family.budgetLimit();

**if** (userRole.equals(role)) {

**if** (userBudget > spend) {

System.***out***

.println("Employee Proxy is called before Employee invocation");

method.invoke(emp,

userBudget, spend);

} **else** {

System.***out***.println("Budget Limit Over");

}

}

}

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

System.***out***

.println("Employee Proxy is called after Employee invocation");

}

}

}

**public** **class** Dummy {

**public** **static** **void** main(String[] args) {

Scanner in = **new** Scanner(System.***in***);

**int** testCases = Integer.*parseInt*(in.nextLine());

**while** (testCases > 0) {

String name = in.next();

String role = in.next();

**int** spend = in.nextInt();

Employee e = **new** Employee(name);

EmployeeProxy.*execute*(e, role, spend);

testCases--;

}

/\*

\* Employee e = new Employee(); e.juniorEmployee(100, 40);

\*/}

}