# Java Annotations

by akashs_csedu

Java annotation can be used to define the metadata of a Java class or class element. We can use Java annotation at the compile time to instruct the compiler about the build process. Annotation is also used at runtime to get insight into the properties of class elements.

Java annotation can be added to an element in the following way:

```
@Entity
Class DemoClass{

}
```

We can also set a value to the annotation member. For example:

```
@Entity(EntityName="DemoClass")
Class DemoClass{

}
```

In Java, there are several built-in annotations. You can also define your own annotations in the following way:

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@interface FamilyBudget {
   String userRole() default "GUEST";
}
```

Here, we define an annotation $\text{FamilyBudget}$, where $\text{userRole}$ is the only member in that custom annotation. The $\text{userRole}$ takes only $\text{String}$ type values, and the default is **"GUEST"**. If we do not define the value for this annotation member, then it takes the default. By using **@Target**, we can specify where our annotation can be used. For example, the $\text{FamilyBudget}$ annotation can only be used with the method in a class. **@Retention** defines whether the annotation is available at runtime. To learn more about Java annotation, you can read the tutorial and oracle docs.

Take a look at the following code segment:

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@interface FamilyBudget {
    String userRole() default "GUEST";
}

class FamilyMember {

    public void seniorMember(int budget, int moneySpend) {
        System.out.println("Senior Member");
        System.out.println("Spend: " + moneySpend);
        System.out.println("Budget Left: " + (budget - moneySpend));
    }

    public void juniorUser(int budget, int moneySpend) {
        System.out.println("Junior Member");
        System.out.println("Spend: " + moneySpend);
        System.out.println("Budget Left: " + (budget - moneySpend));
    }
}

public class Solution {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int testCases = Integer.parseInt(in.nextLine());
        while (testCases > 0) {
            String role = in.next();
            int spend = in.nextInt();
            try {
                Class annotatedClass = FamilyMember.class;
                Method[] methods = annotatedClass.getMethods();
                for (Method method : methods) {
                    if (method.isAnnotationPresent(FamilyBudget.class)) {
                        FamilyBudget family = method
                                .getAnnotation(FamilyBudget.class);
                        String userRole = family.userRole();
                        int budgetLimit = family.budgetLimit();
                        if (userRole.equals(role)) {
                            if(spend<=budgetLimit){
                                method.invoke(FamilyMember.class.newInstance(),
                                        budgetLimit, spend);
                            }else{
                                System.out.println("Budget Limit Over");
                            }
                        }
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            testCases--;
        }
    }
}
```

Here, we partially define an annotation, $\text{FamilyBudget}$ and a class, $\text{FamilyMember}$. In this problem, we give the user role and the amount of money that a user spends as inputs. Based on the user role, you have to call the appropriate method in the $\text{FamilyMember}$ class. If the amount of money spent is over the budget limit for that user role, it prints `Budget Limit Over`.

Your task is to complete the $\text{FamilyBudget}$ annotation and the $\text{FamilyMember}$ class so that the $\text{Solution}$ class works perfectly with the defined constraints.

**Note**: You must complete the $5$ incomplete lines in the editor. You are not allowed to change, delete or modify any other lines. To restore the original code, click on the top-left button on the editor and create a new buffer.

**Input Format**

The first line of input contains an integer $N$ representing the total number of test cases. Each test case contains a string and an integer separated by a space on a single line in the following format:

```
UserRole MoneySpend
```

**Constraints**

$2 \le N \le 10$
$0 \le MoneySpend \le 200$
$|UserRole| = 6$

Name contains only lowercase English letters.

**Output Format**

Based on the user role and budget outputs, output the contents of the certain method. If the amount of money spent is over the budget limit, then output `Budget Limit Over`.

**Sample Input**

```
3
SENIOR 75
```

```
JUNIOR 45
SENIOR 40
```

**Sample Output**

```
Senior Member
Spend: 75
Budget Left: 25
Junior Member
Spend: 45
Budget Left: 5
Senior Member
Spend: 40
Budget Left: 60
```

**Solved score:** 12.50pts

**Submissions:**6059

**Max Score:**25
**Difficulty:** Medium

**Rate This Challenge:**
☆☆☆☆☆

More

**Current Buffer** (saved locally, editable)

Java 7

```java
import java.lang.annotation.*;
import java.lang.reflect.*;
import java.util.*;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@interface FamilyBudget {
   String userRole() default "GUEST";
   ~~Complete the interface~~
}

class FamilyMember {
   ~~Complete this line~~
   public void seniorMember(int budget, int moneySpend) {
      System.out.println("Senior Member");
      System.out.println("Spend: " + moneySpend);
      System.out.println("Budget Left: " + (budget - moneySpend));
   }

   ~~Complete this line~~
   public void juniorUser(int budget, int moneySpend) {
      System.out.println("Junior Member");
      System.out.println("Spend: " + moneySpend);
      System.out.println("Budget Left: " + (budget - moneySpend));
   }
}

public class Solution {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      int testCases = Integer.parseInt(in.nextLine());
      while (testCases > 0) {
         String role = in.next();
         int spend = in.nextInt();
         try {
            Class annotatedClass = FamilyMember.class;
            Method[] methods = annotatedClass.getMethods();
            for (Method method : methods) {
               if (method.isAnnotationPresent(FamilyBudget.class)) {
                  FamilyBudget family = method
```

```java
                        .getAnnotation(FamilyBudget.class);
                  String userRole = family.userRole();
                  int budgetLimit = ~~Complete this line~~;
                  if (userRole.equals(role)) {
                     if(~~Complete this line~~){
                        method.invoke(FamilyMember.class.newInstance(),
                              budgetLimit, spend);
                     }else{
                        System.out.println("Budget Limit Over");
                     }
                  }
               }
            }
         } catch (Exception e) {
            e.printStackTrace();
         }
         testCases--;
      }
   }
}
```

Line: 1 Col: 1

Upload Code as File

Test against custom input

Run Code

Submit Code