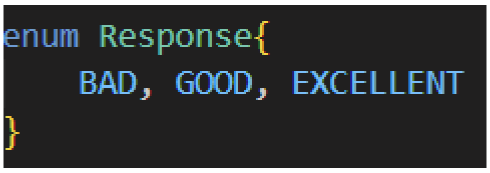
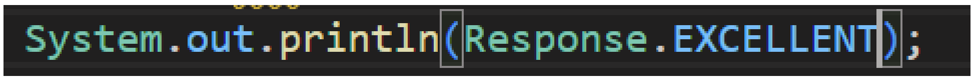
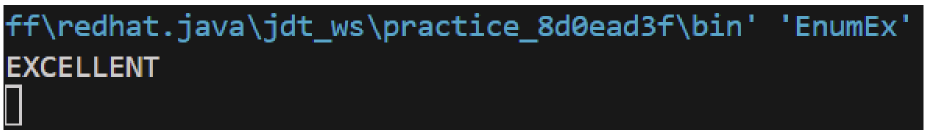
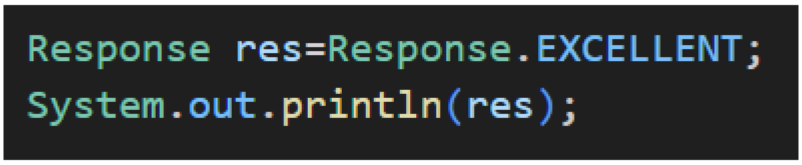
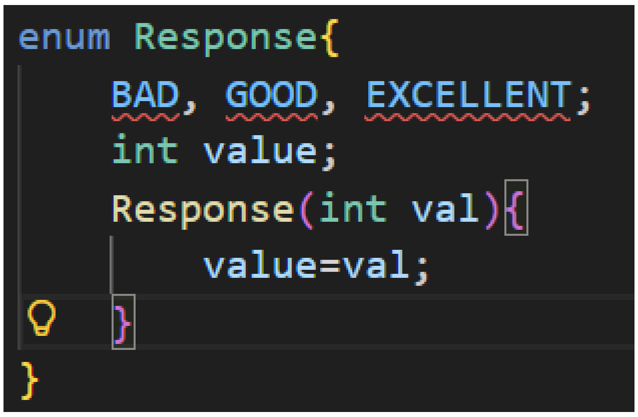
13/02/26 Enumerators

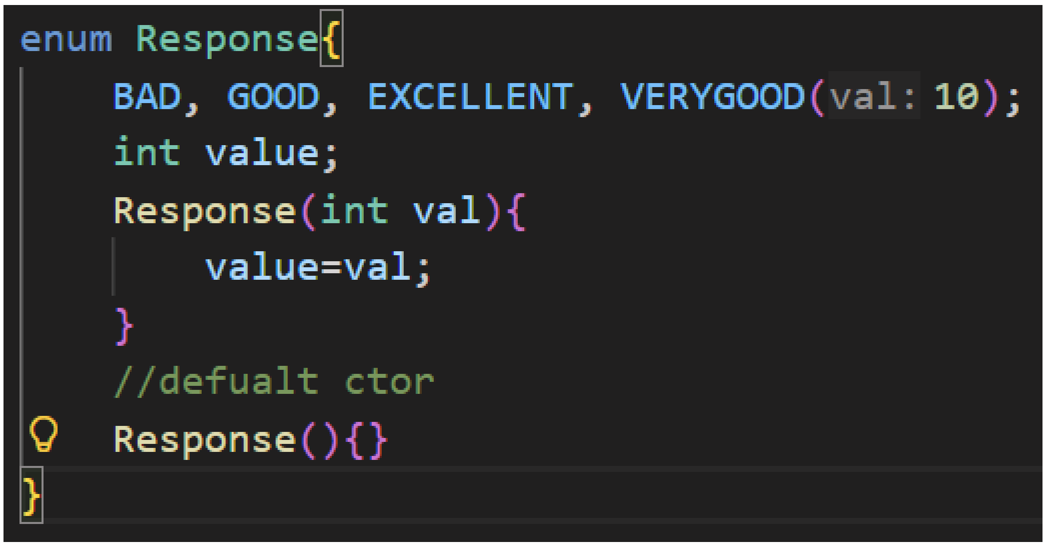
1. Used to create the collection of constants
2. By default all the members of enums are public static final and self typed
    a. Means you need not create it using new key word
        i. Response LOW=new Response() //This one is wrong.
        ii. LOW //this one is valid
3. Ex:
    a. 
    ```
    enum Response{
        BAD, GOOD, EXCELLENT
    }
    ```
    b. Here need not terminate with ;, because JVM automatically assumes enum variables are terminated.
4. Accessing the Response object outside the enum
    a. 
    ```
     System.out.println(Response.EXCELLENT);
    ```
    b. output
    c. 
    ```
    ff\redhat.java\jdt_ws\practice_8d0ead3f\bin' 'EnumEx'
    EXCELLENT
    ```
5. We can also store it in another object.
    
    ```
    Response res=Response.EXCELLENT;
    System.out.println(res);
    ```
    Since EXCELLENT type is RESPONSE, we are creating response reference called res, and storing EXCELLENT in res. Now if I print res, I will get EXCELLENT as output. Now both res and EXCELLENT pointing to same memory.

6. Enums can also have instance variable and methods just like class, but they have to be placed after Object creation of enum



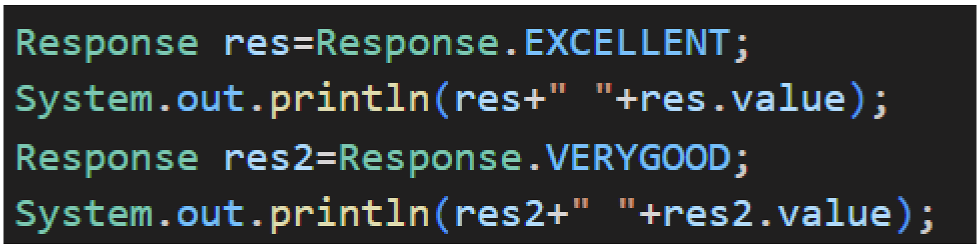
```java
enum Response{
    BAD, GOOD, EXCELLENT;
    int value;
    Response(int val){
        value=val;
    }
}
```

Since I have created ctor, there is red underline, because BAD,GOOD,EXCELLENT invoking default ctor, but there is no default ctor

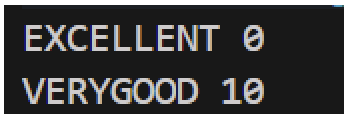


```java
enum Response{
    BAD, GOOD, EXCELLENT, VERYGOOD(val: 10);
    int value;
    Response(int val){
        value=val;
    }
    //defualt ctor
    Response(){}
}
```

Now VERYGOOD will invoke Response(int val) with 10, so value field of VERYGOOD will become 10. The value of BAD,GOOD,EXCELLENT will become 0, default ctor.

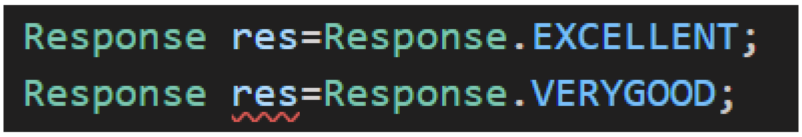


```java
Response res=Response.EXCELLENT;
System.out.println(res+" "+res.value);
Response res2=Response.VERYGOOD;
System.out.println(res2+" "+res2.value);
```



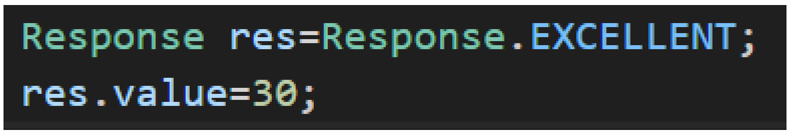
```
EXCELLENT 0
VERYGOOD 10
```

See all object of Response is final, so we cannot change its value.



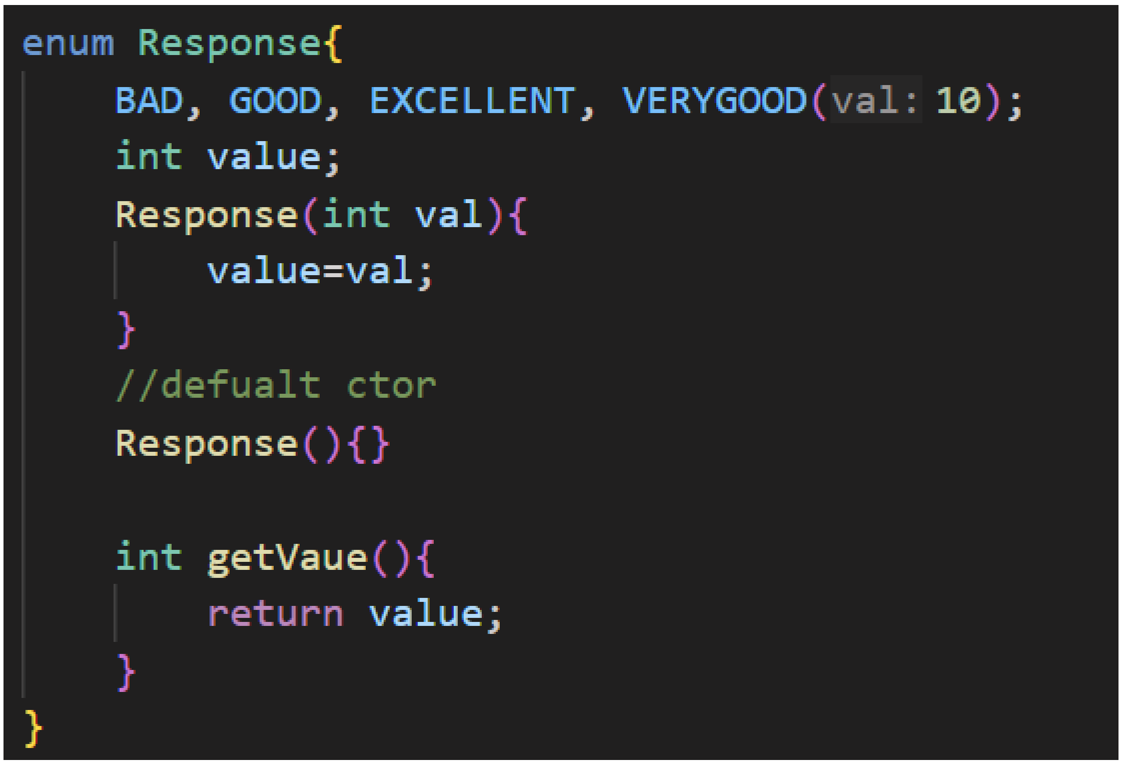
```java
Response res=Response.EXCELLENT;
Response res=Response.VERYGOOD;
```

See I have created reference of Response res, and assigned EXCELLENT in that. Now res also became final. So next when I again assign res with VERYGOOD, it throws me error.



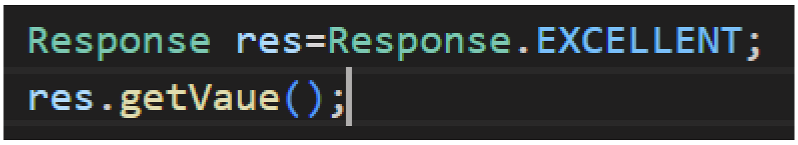
```
Response res=Response.EXCELLENT;
res.value=30;
```

See I can change the value inside EXCELLENT object, because it is not final, but object is final.



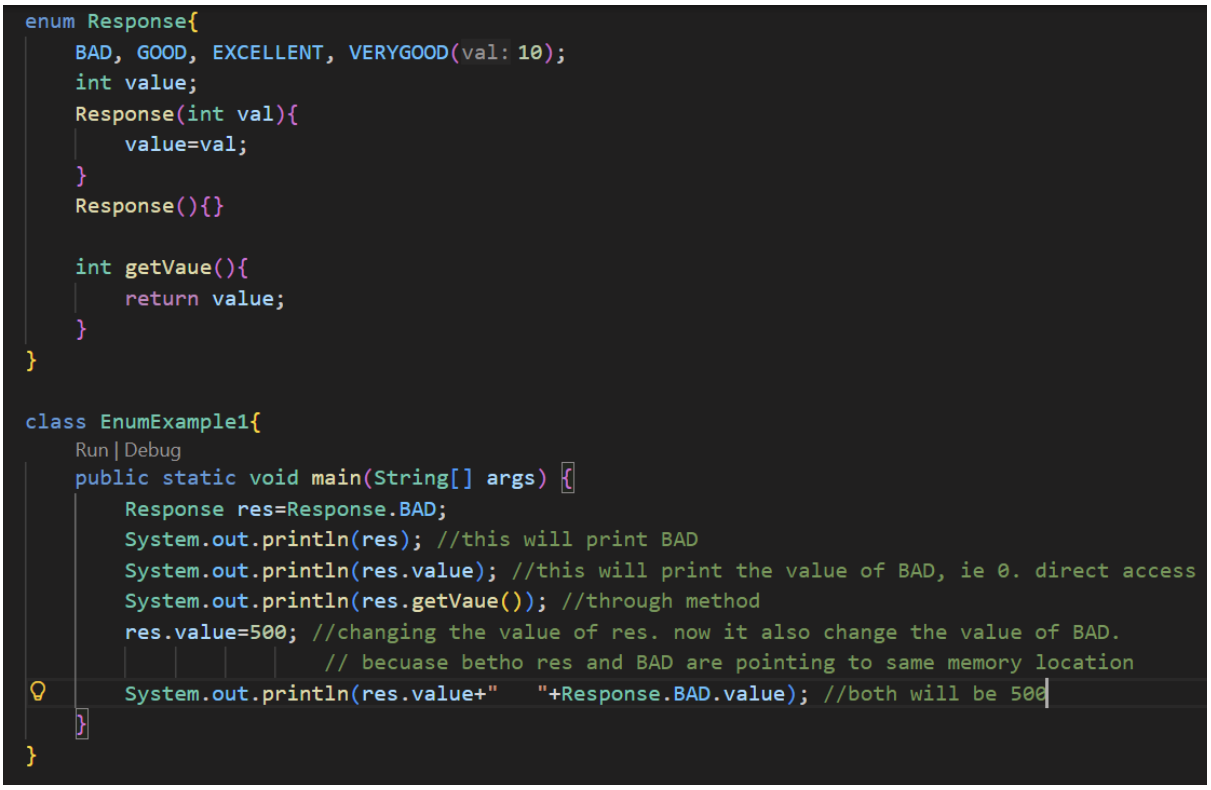
```
enum Response{
    BAD, GOOD, EXCELLENT, VERYGOOD(val: 10);
    int value;
    Response(int val){
        value=val;
    }
    //defualt ctor
    Response(){}

    int getVaue(){
        return value;
    }
}
```



```
Response res=Response.EXCELLENT;
res.getVaue();
```

So instead of accessing value field of EXCELLENT directly outside the classs, we can create one function and access it outside.



```
enum Response{
    BAD, GOOD, EXCELLENT, VERYGOOD(val: 10);
    int value;
    Response(int val){
        value=val;
    }
    Response(){}

    int getVaue(){
        return value;
    }
}

class EnumExample1{
    Run | Debug
    public static void main(String[] args) {
        Response res=Response.BAD;
        System.out.println(res); //this will print BAD
        System.out.println(res.value); //this will print the value of BAD, ie 0. direct access
        System.out.println(res.getVaue()); //through method
        res.value=500; //changing the value of res. now it also change the value of BAD.
                        // becuase betho res and BAD are pointing to same memory location
        System.out.println(res.value+"    "+Response.BAD.value); //both will be 500
    }
}
```

```
import java.util.Scanner;

enum Response{
  BAD, GOOD, EXCELLENT, VERYGOOD(10);
  int value;
  Response(int val){
    value=val;
  }
  Response(){}

  int getVaue(){
    return value;
  }
}

class EnumExample1{
  public static void main(String[] args) {
    Response res=Response.BAD;
    System.out.println(res); //this will print BAD
    System.out.println(res.value); //this will print the value of BAD, ie 0. direct
access
    System.out.println(res.getVaue()); //through method
```

```
        res.value=500; //changing the value of res. now it also change the value of
BAD.
                    // becuase betho res and BAD are pointing to same memory location
        System.out.println(res.value+"   "+Response.BAD.value); //both will be 500
    }
}
```