

★ Lecture 06 :- Switch Statements + Nested Case in Java

#1) Need of Switch statements :-

Switch statements are needed to optimise multiple if conditions.

ex:- Code :- P21MulIfConditions.java

```
package com.tejas;  
import java.util.Scanner;
```

```
public class P21MulIfConditions {
```

```
    public static void main(String[] args)
```

```
    {  
        Scanner sc = new Scanner(System.in);  
        String fruit = sc.next();
```

```
        if (fruit == "mango" || fruit.equals("mango"))  
        {  
            System.out.println("King of fruit");  
        }
```

```
        if (fruit.equals("apple")) {  
            System.out.println("a sweet red fruit");  
        }
```

```
    }
```

```
}
```

Note:- 1.) Why didn't we use else if here?

Ans:- See if fruit string is equal to mango, then it can't be equal to anything else. So it doesn't need to go to other statements. That's why using else if or not is same thing. Only one of the condⁿ should true

2.) Why fruit.equals("String") instead fruit=="String"?

Ans:- a) .equals only check the value and if value is exact same provided that ~~and~~ both variable points to different objects then only .equals return true.

ex:- String a = "Kunal";
String b = "Kunal";

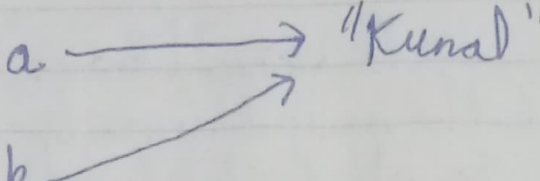
So .equals gives true only if ^{a)}Kunal and ^{b)}Kunal is same in value even if they are diffⁿ in terms of objects.

b.) == checks both value and reference ~~Objects~~ of the variables

ex:- String a = "Kunal";
String b = "Kunal";

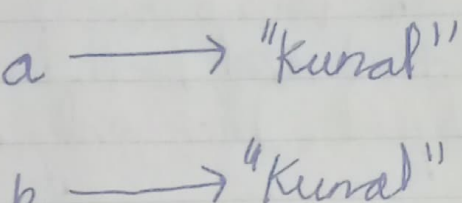
and if a == b
means, a and b pointing to same object hence their value is same.

Let's understand this with diagrams

1.)  then, `==` gives True
and, `.equals` also gives `True`

but `==` gives True cos `a` and `b` pointing to same object.

and, `.equals` gives True cos value of `a` and `b` is same.

2.)  then, `==` gives False cos
`a` and `b` are not pointing
same object.

but, `.equals` still gives
True cos value is same
and same object pointing
doesn't matter for `.equals`.

Now code in `if` conditions, java is little
messy or repetitive and don't look nice as
syntax.

For that reason we use Switch Statements.

2) Switch Statements :-

a) Switch Statements (Traditional) :-

In switch statements, you can jump to various cases based on your expression.

Syntax :-

```
switch (expression) {
```

```
    case one :  
        // do something  
        break ;
```

```
    case two :  
        // do something  
        break ;
```

```
    default :  
        // do something  
        break ; [Needed only if you writing any case  
                  below default]
```

```
}
```

→ Properties :-

- Cases have to be the same type as expressions, must be a constant or literal (means can't be variables etc)
- Duplicate case values are not allowed.
- Break is used to terminate the sequence.
- If break is not used, it will continue to next case.
- default will execute when none of the above does.
- if default is not at the end, put break after it.

Code :- P22SwitchStatements.java

```
package com.tejas;  
import java.util.Scanner;  
  
public class P22SwitchStatements {  
    public static void main (String[] args)  
    {  
        Scanner sc = new Scanner(System.in);  
        String fruit = sc.next();  
    }  
}
```



```
switch (fruit) {
```

```
    case "Mango":
```

```
        System.out.println("King of fruits.");
```

```
        break;
```

```
    case "Apple":
```

```
        System.out.println("A sweet red fruit.");
```

```
        break;
```

```
    case "Banana":
```

```
        System.out.println("A yellow with lot's of
```

```
        carbohydrates.");
```

```
        break;
```

```
    case "Orange":
```

```
        System.out.println("A fruit with same name
```

```
        as orange color.");
```

```
        break;
```

```
    default:
```

```
        System.out.println("Please enter valid fruit
```

```
        name.");
```

```
}
```

```
}
```

```
}
```

b.) Enhanced Switch:-

Syntax:-

```
switch (expression) {
```

```
    case one -> // do something ;
```

```
    case two -> // do something ;
```

```
    case three -> // do something ;
```

```
    default -> System.out.println("Please enter  
               a valid fruit. "); // do something ;
```

}

B

Code: P23 EnhancedSwitch.java

```
package com.tejas;  
import java.util.Scanner;
```

```
public class P23 EnhancedSwitch java {  
    public static void main (String [] args)  
    {  
        Scanner sc = new Scanner (System.in);  
        String fruit = sc.next();
```

switch(fruit) {

case "Mango" → System.out.println("King of Fruits.");
case "Apple" → System.out.println("A sweet red fruit.");
case "Orange" → System.out.println("Round fruit.");
case "Banana" → System.out.println("A fruit with lots
of carbohydrates");
default → System.out.println("please enter a
valid fruit.");

}

}

}

Code :- a) P24 weekdays ES, java → Showing Mul. Statements
↑
Enhanced in switch case.

- ES means enhanced switch
- SS means Switch statements (Traditional)
- ~~MI~~ means Multiple If
MI

b.) P25 weekendCheck SS, java → Showing how ~~not~~ to
use break cond. with
many cases.

c.) P26 weekday MI, java → Showing Mul. if program of
P24 weekdays ES, java.

d.) P27 weekendCheck ES, java → Showing same as b.)

C.) Nested Switch case :-

It basically means switch case inside switch case like we have nested if-else.

code :- P28 NestedSwitchSS.java
P29 NestedSwitchES.java