Conditional Statements:

1> Why we need conditional statements?

2> Where it is used in the Industry or coding
- *Logical bugs - shopping cart
- * Error Handling - Crowdstrike

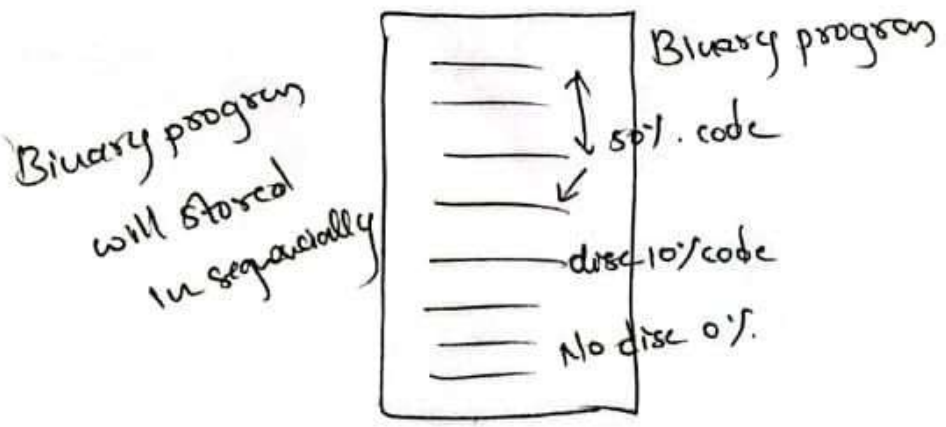
3> Types or patterns of conditional statements
- a> If
- b> If else
- c> If else If else
- d> Nested If else
- e> Conditional
- f> Switch

4> How to write clear & clean conditional Statements.

5> Sample Examples:
- a> Is number positive
- b> Is number positive / Negative
- c> classify student percentage into distraction, first class, 2nd class etc
- d> Convert day month in digit to word ex: if input is 5 then return May
- e> Greater or smaller number using conditional operator.

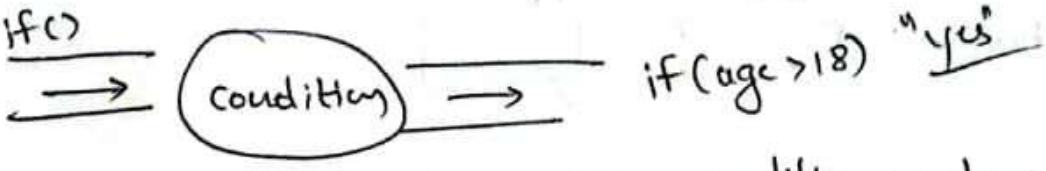
Lets explain all ----

1> Why we need conditional statements?



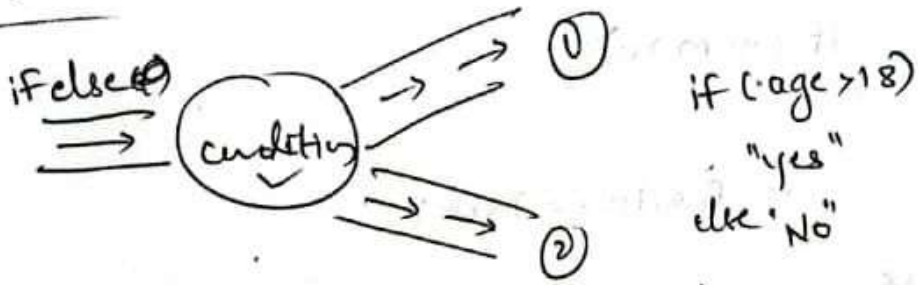
In apps like Zomato or Swiggy there are discounts are there So two codes have two discounts offers codes two you can apply according to your option.

#pattern 1



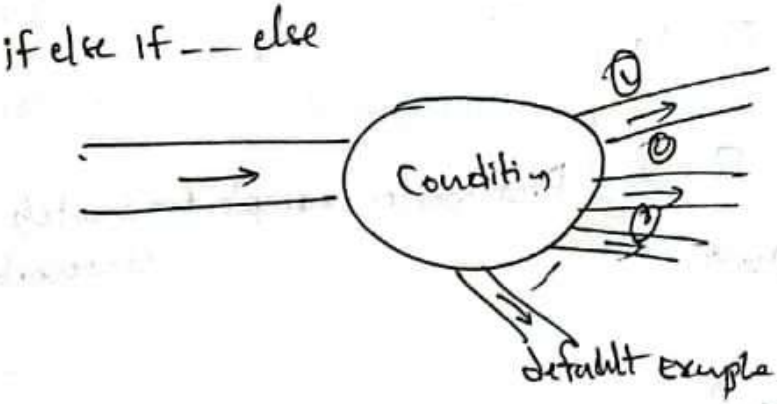
In this 1st pattern if the condition is true we can go for forward otherwise no.

#pattern 2



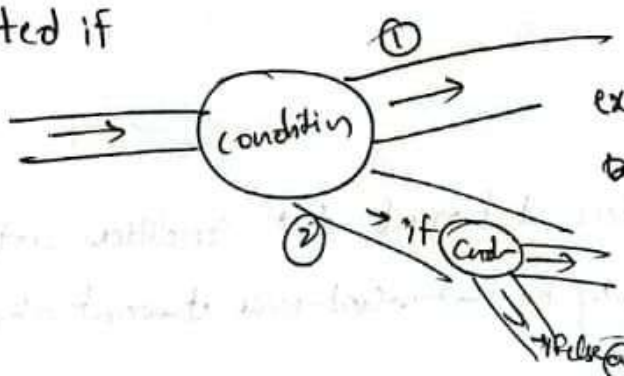
In pattern 2 we have two paths, if else statement executes both true and false. means if true go to path ① else false go to path ②

#pattern 3



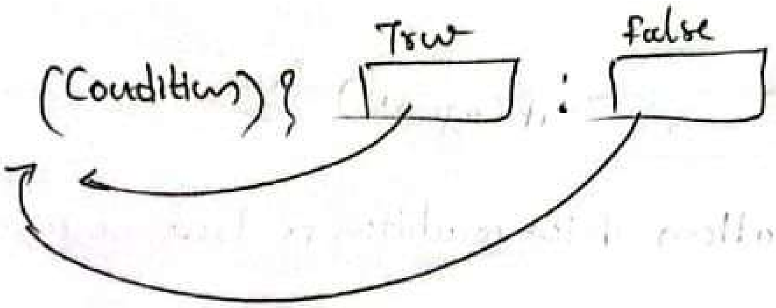
— In pattern 3 there are many paths we can check one after another using if else if, if no one matches lastly the default is there.

#pattern 4



In pattern 4 we can have example 2 paths ① and ② we can create another paths in path ① or ② using Nested if statement.

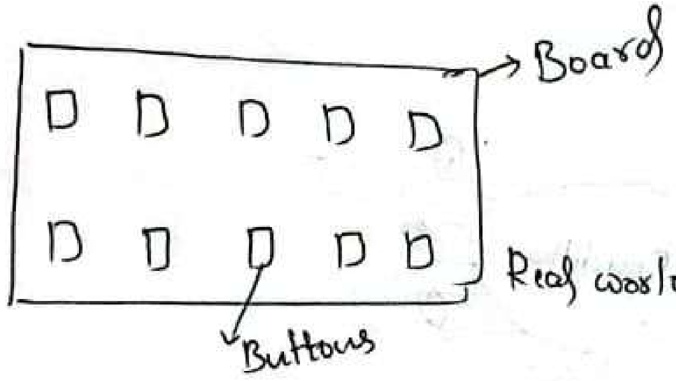
Conditional Statements:



is_Positive = (num>0) ? True : False ;

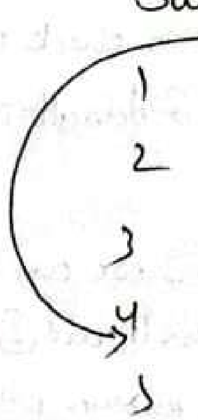
```
if (num>0)
{
    is Positive = True;
else
{   is Positive = False;
}
```

Switch



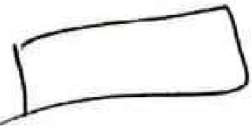
Real world example for switch statements

Switch (condition)



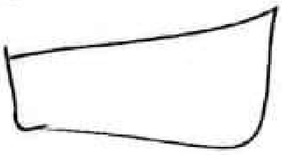
In switch statement the condition not check ~~the sequ~~ it directly apply to ~~the what ever it want~~ above condition

if ( )



①

else



```
fun (-, -, - - -)

Checking the inputs are correct or not
  if(input1)
  {
     if(input2)
     {
        if(input3)
     }
  }
}
```

Early exit

```
if (input1 is not X)
    return;
if (inp 2 is X)
    return;
```

Understanding Conditional Statements In Java

1. Why We Need Conditional Statements

Conditional statements are a fundamental concept in programming that allow a program to make decisions. They enable the program to execute different blocks of code based on whether a specific condition is true or false. Without them, a program would simply execute the same sequence of instructions every time, regardless of the input or situation. They are essential for creating dynamic and responsive applications.

2. Where They're Used in the Industry or Coding

Conditional statements are used everywhere in software development. They are the backbone of program logic. Here are a few examples:

- **Logical Bugs:** A common example is in an e-commerce **shopping cart**. A condition might check if a product is in stock before allowing a user to add it to their cart. If this condition is missing or incorrect, it can lead to a "logical bug" where a user can purchase an item that's unavailable.
- **Error Handling:** Programs need to gracefully handle unexpected situations. A conditional statement can check if a file exists before trying to open it, or if a user entered a valid email address before submitting a form. This prevents the program from crashing.
- **User Interface (UI) and Game Logic:** In a game like **Counter-Strike**, a conditional statement checks if a player has enough currency to buy a weapon. In a website, a condition might check if a user is logged in to determine whether to display a "log out" button or a "log in" button.

3. Types or Patterns of Conditional Statements

There are several ways to implement conditional logic. The most common types are:

- **if:** The simplest form. The code inside the if block only runs if the condition is true.
- **if else:** This provides an alternative path. If the initial condition is true, the if block runs. If it's false, the else block runs.
- **if else if else:** This pattern is used to check multiple, mutually exclusive conditions in a specific order. The program checks the first if condition, and if it's false, it moves to the next else if, and so on, until a true condition is found or it reaches the final else block.
- **Nested if else:** This involves placing one conditional statement inside another. This is used when a condition depends on a previous condition being true. For example, if (user Is Logged In) then if (user Has Admin Rights).
- **Ternary (Conditional) Operator:** A shorthand way of writing a simple if else statement. It's often used for assigning a value to a variable based on a condition. The syntax is condition ? value_if_true : value_if_false.

- **switch:** This is an alternative to long if else if chains, especially when you are comparing a single variable against a series of possible constant values. It's often more readable and sometimes more efficient.

4. How to Write Clear, Clean Conditional Statements

- **Clarity over Complexity:** Avoid overly complex nested if statements. If a function has too many nested conditions, it can become difficult to read and debug.
- **Meaningful Variable Names:** Use descriptive variable names that make the condition easy to understand, such as isUserLoggedIn instead of x.
- **Boolean Flags:** Use a boolean variable (a variable that is either true or false) to simplify complex conditions. For example, let canAfford = (userMoney >= itemPrice); then use if (canAfford).

5. Sample Examples

Let's look at how to implement the examples you've listed.

a) Is number positive?

```
let number = 10;
if (number > 0) {
  console.log("The number is positive.");
}
```

b) Is number positive or negative?

```
let number = -5;
if (number > 0) {
  console.log("The number is positive.");
} else {
  console.log("The number is negative or zero.");
}
```

c) Classify student percentage into distraction, first class, second class, etc.

```
let percentage = 75;
if (percentage >= 80) {
  console.log("First Class");
} else if (percentage >= 60) {
  console.log("Second Class");
```

```
} else if (percentage >= 40) {
  console.log("Third Class");
} else {
  console.log("Distraction (Fail)");
}
```

d) Convert day/month/digit to a word

This would likely use a switch statement for a clean implementation.

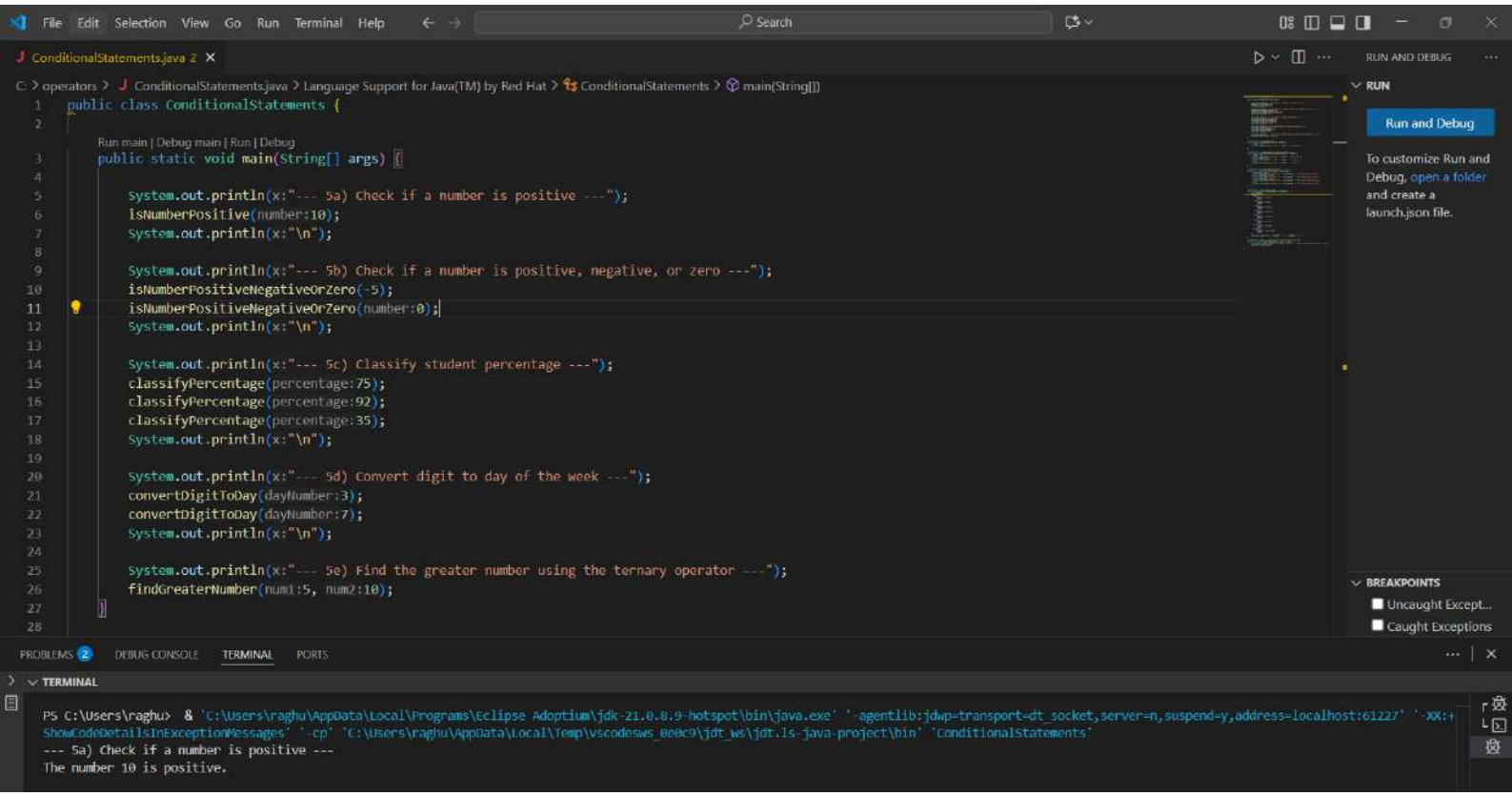
```
let dayNumber = 3;
let dayName;

switch (dayNumber) {
  case 1:
    dayName = "Sunday";
    break;
  case 2:
    dayName = "Monday";
    break;
  case 3:
    dayName = "Tuesday";
    break;
  default:
    dayName = "Invalid Day";
}

console.log(dayName); // Output: Tuesday
```

e) Greater or smaller number using conditional operator

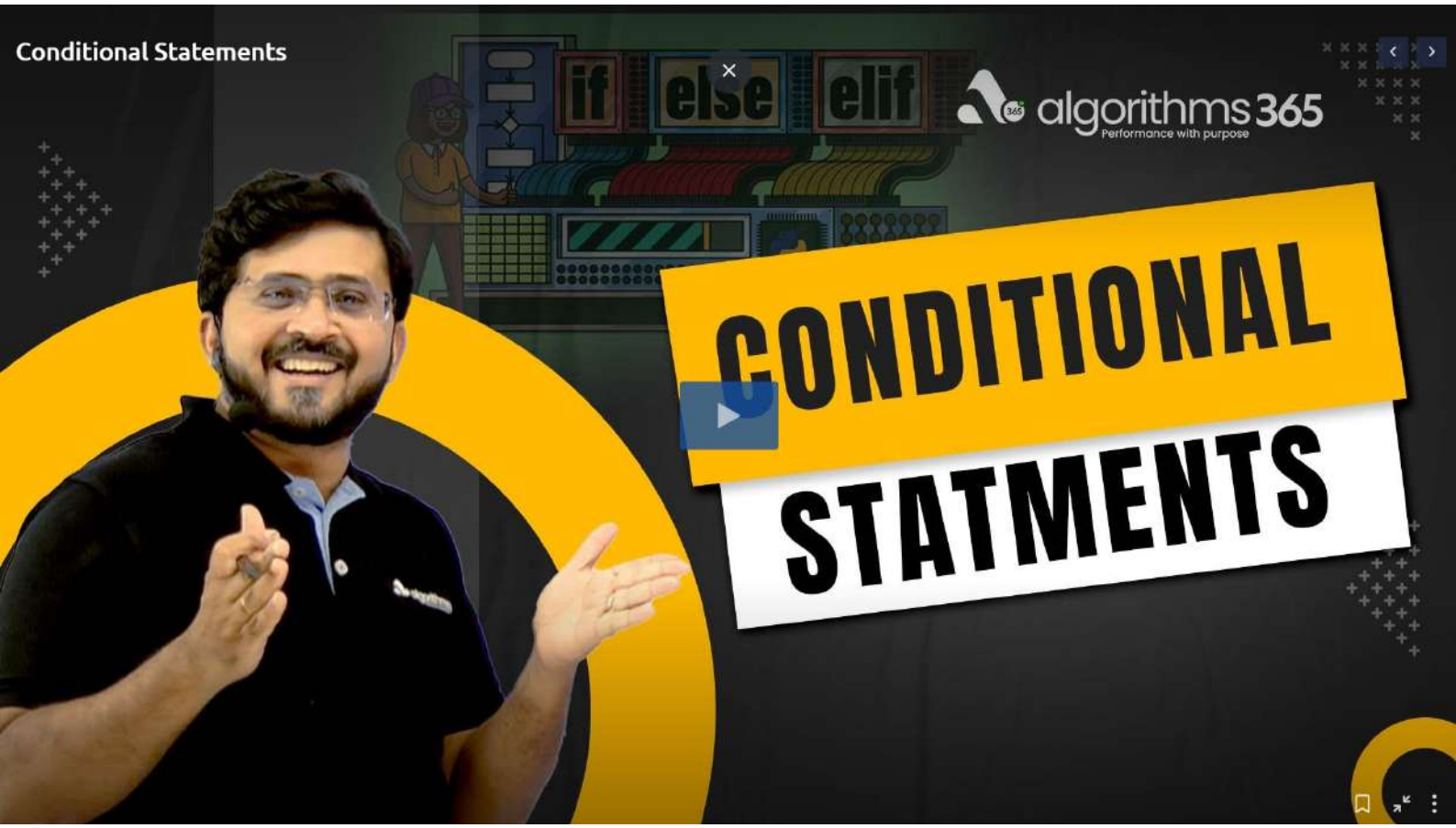
This example uses the **ternary operator** for a compact solution.

```
let num1 = 5;
let num2 = 10;

// The syntax is: (condition) ? value_if_true : value_if_false;
```

```
let result = (num1 > num2) ? "num1 is greater" : "num2 is greater or equal";

console.log(result); // Output: num2 is greater or equal
```




```java
public class ConditionalStatements {

    Run main | Debug main | Run | Debug
    public static void main(String[] args) {

        System.out.println("--- 5a) Check if a number is positive ---");
        isNumberPositive(number:10);
        System.out.println("\n");

        System.out.println("--- 5b) Check if a number is positive, negative, or zero ---");
        isNumberPositiveNegativeOrZero(-5);
        isNumberPositiveNegativeOrZero(number:0);
        System.out.println("\n");

        System.out.println("--- 5c) Classify student percentage ---");
        classifyPercentage(percentage:75);
        classifyPercentage(percentage:92);
        classifyPercentage(percentage:35);
        System.out.println("\n");

        System.out.println("--- 5d) Convert digit to day of the week ---");
        convertDigitToDay(dayNumber:3);
        convertDigitToDay(dayNumber:7);
        System.out.println("\n");

        System.out.println("--- 5e) Find the greater number using the ternary operator ---");
        findGreaterNumber(num1:5, num2:10);
    }
```


Conditional Statements
if
else
elif
algorithms365
Performance with purpose
CONDITIONAL
STATMENTS