**Polling Question 1**  
How do you identify good code versus bad code?  
  
**1: 1 | 4%**

**2: 0 | 0%**

**3: 0 | 0%**

**4: 24 |||||||||||||||||||| 96%   
  
Polling options**1: Good code is well-structured, readable, and maintainable \*

2: Bad code has complex and convoluted logic

3: Good code follows established coding convention

4: All of the above

**Polling Question 2**  
Which design principle states that a class should have only one reason to change?  
  
**1: 23 ||||||||||||||||||| 92%**

**2: 2 || 8%**

**3: 0 | 0%**

**4: 0 | 0%   
  
Polling options**1: Single Responsibility Principle \*

2: Open/Closed Principle

3: Liskov Subtitution Principle

4: Interface segregation principle

**Polling Question 3**  
What is the purpose of the Decorator pattern?  
  
**1: 3 ||| 10%**

**2: 24 ||||||||||||||||| 80%**

**3: 1 | 3%**

**4: 2 || 7%   
  
Polling options**1: To encapsulate a request as an object

2: add new functionality to an existing object \*

3: provide an interface for creating objects

4: handle failures and errors in distributed sys

<https://www.tutorialspoint.com/uml/uml_activity_diagram.htm>

<https://www.educba.com/uml-activity-diagram/>

**Problem Statement 1:**

Get a number from the user if it is odd add with 5 and multiply with 5 now display and now add 20  
  
if it is even add 7 multiply with 7 display and add 20

public class Main {

    public static void main(String[] args) {

        int num = 5;

        if (num % 2 != 0) {

            num += 5;

            num \*= 5;

            System.out.println(num);

            num += 20;

        } else {

            num += 7;

            num \*= 7;

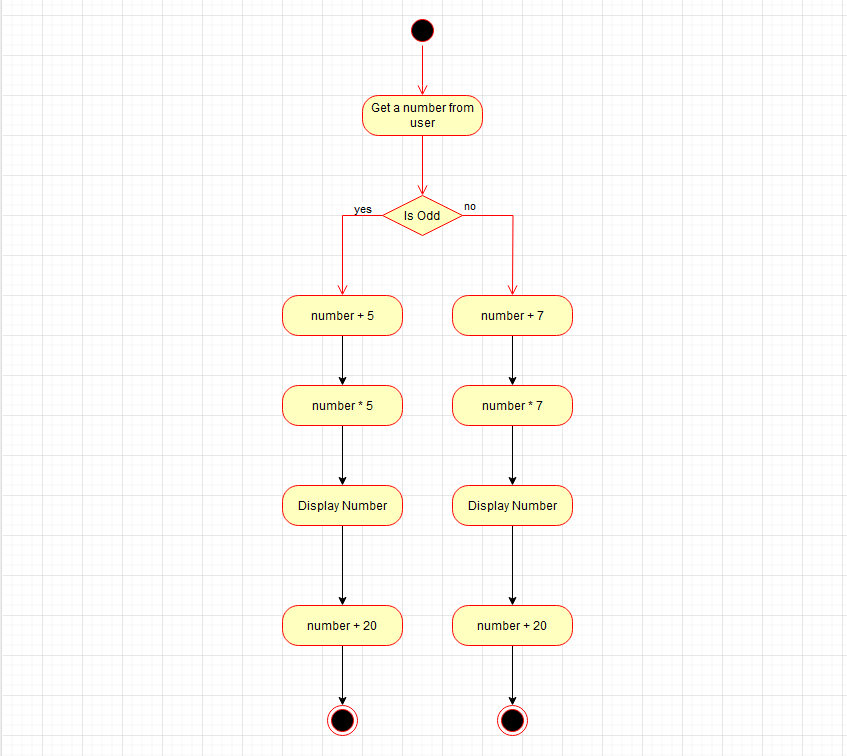
            System.out.println(num);

            num += 20;

        }

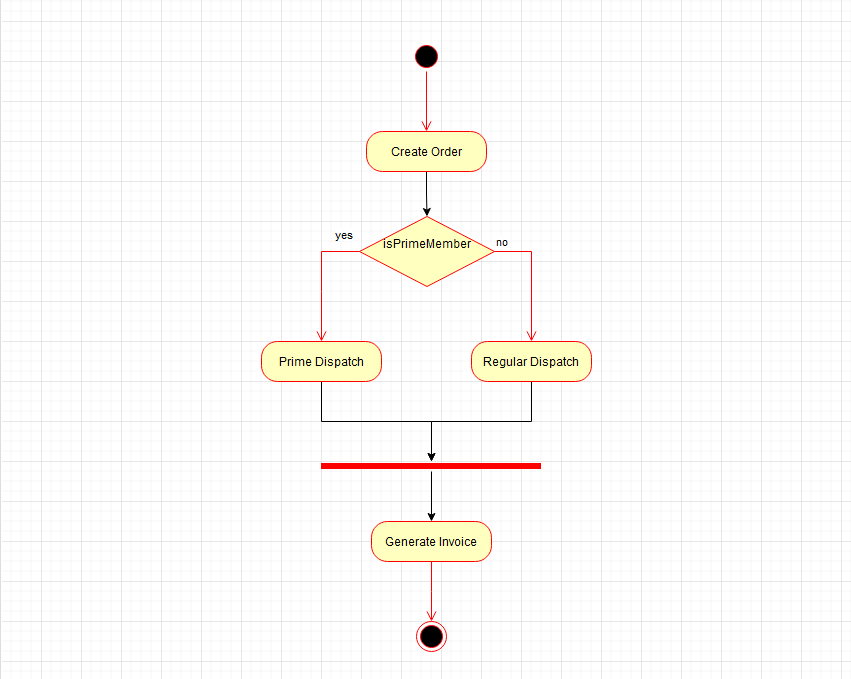
    }

}



**Problem Statement 2:**

Create order if customer is prime, prime dispatch option(genarating an invoice) if not normal dispatch option(generating an invoice)



public class Main {

    public static void main(String[] args) {

        placeOrder("Book", true);

    }

    public static void placeOrder(String item, boolean isPrimeMember) {

        if (isPrimeMember) {

            System.out.println("Prime Dispatch");

        } else {

            System.out.println("Regular Dispatch");

        }

        System.out.println("Invoise Generated");

    }

}

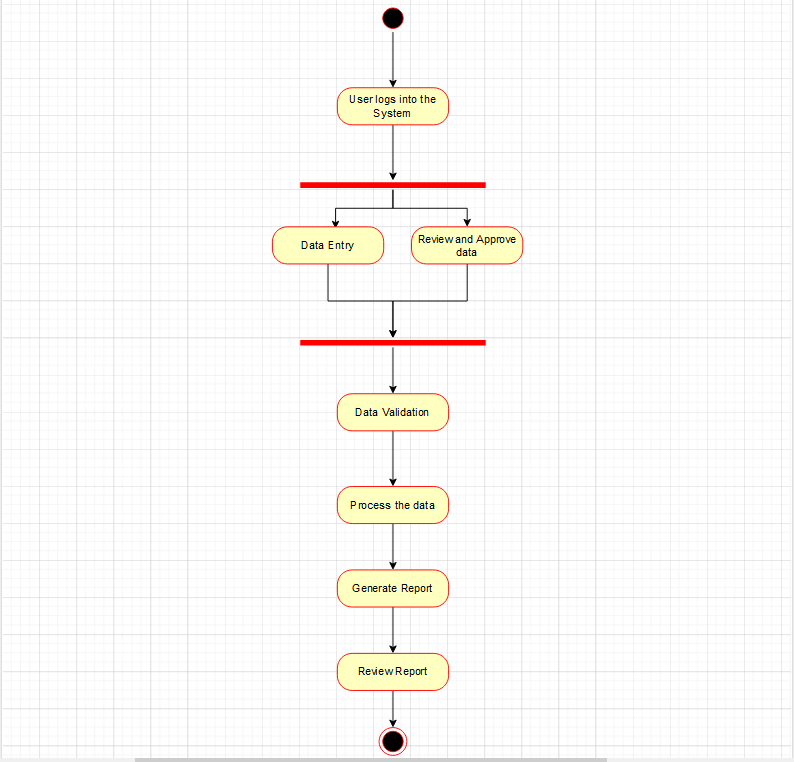
**Problem Statement 3:**

**Parallel Process Execution**

1. Start
2. User logs into the system
3. Fork (@):

* Path 1: User performs data entry
* Path 2: User reviews and approves data

1. Join (@): After both paths 3.1 and 3.2 are complete
2. System performs data validation
3. System processes the data
4. System generates a report
5. User receives the report
6. End



public class Main {

    public static void main(String[] args) {

        System.out.println("User logs into the system");

        fork();

    }

    public static void path1() {

        System.out.println("Performing Data Entry...");

    }

    public static void path2() {

        System.out.println("Reviewing and Approving data...");

    }

    public static void fork() {

        path1();

        path2();

        join();

    }

    public static void join() {

        System.out.println("Perform Data Validation");

        System.out.println("Process the data");

        System.out.println("Generate a report");

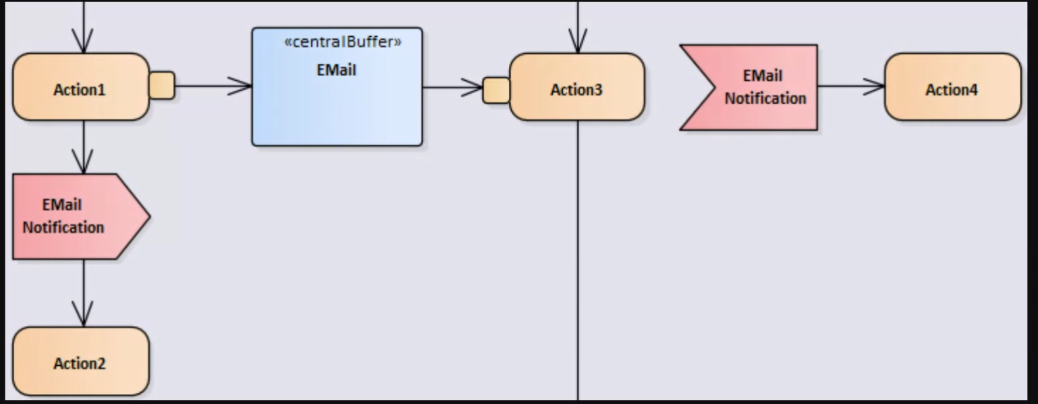
        System.out.println("User receives the report");

        System.out.println("End");

    }

}

**Activity:**



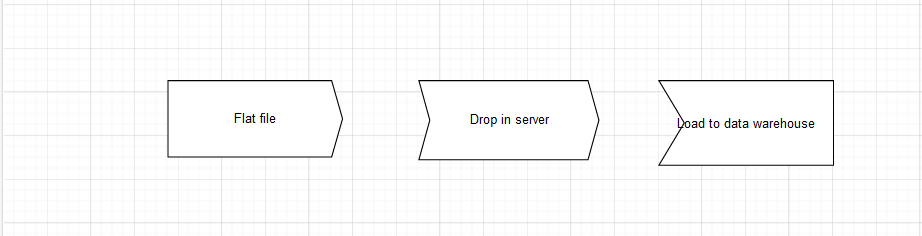
**Problem Statement 4:**

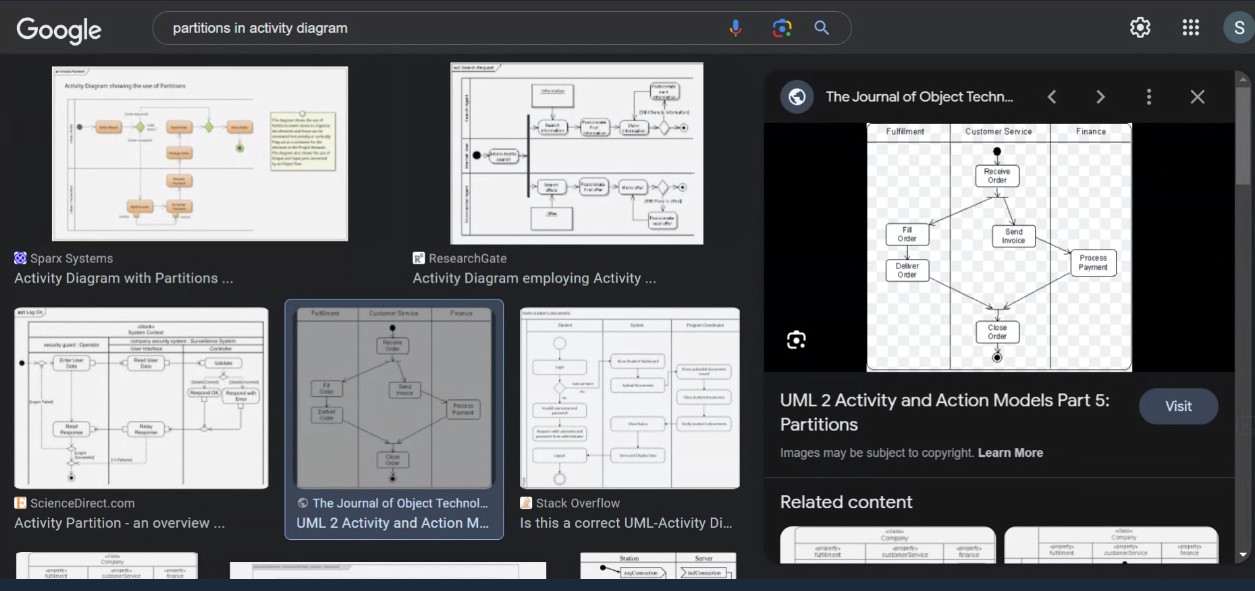
There is a Database, Query run and captures 1000

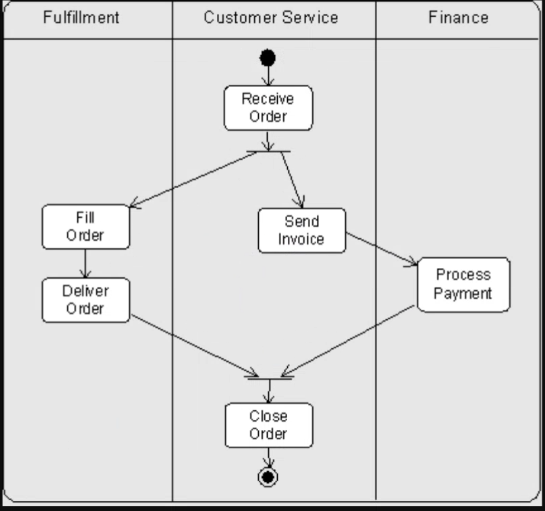
Activity 1: Generate flat file

Activity2.1: Drop file in server

Activity 2.2: Load it into data warehouse.







**Activity Diagram for Delivery System:**

