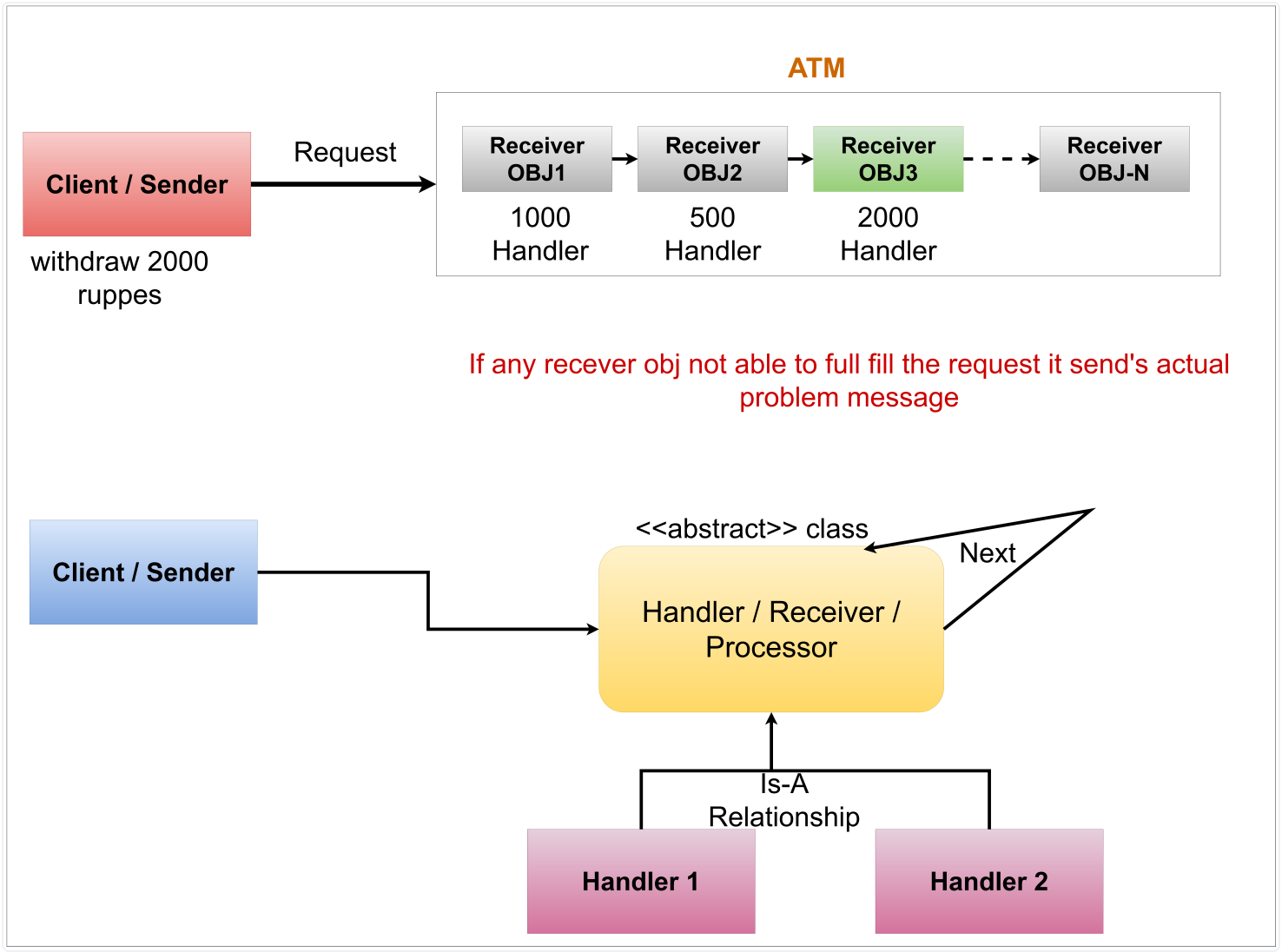
Design Logging System | Chain of Responsibility Design Pattern | Amazon System Design interview

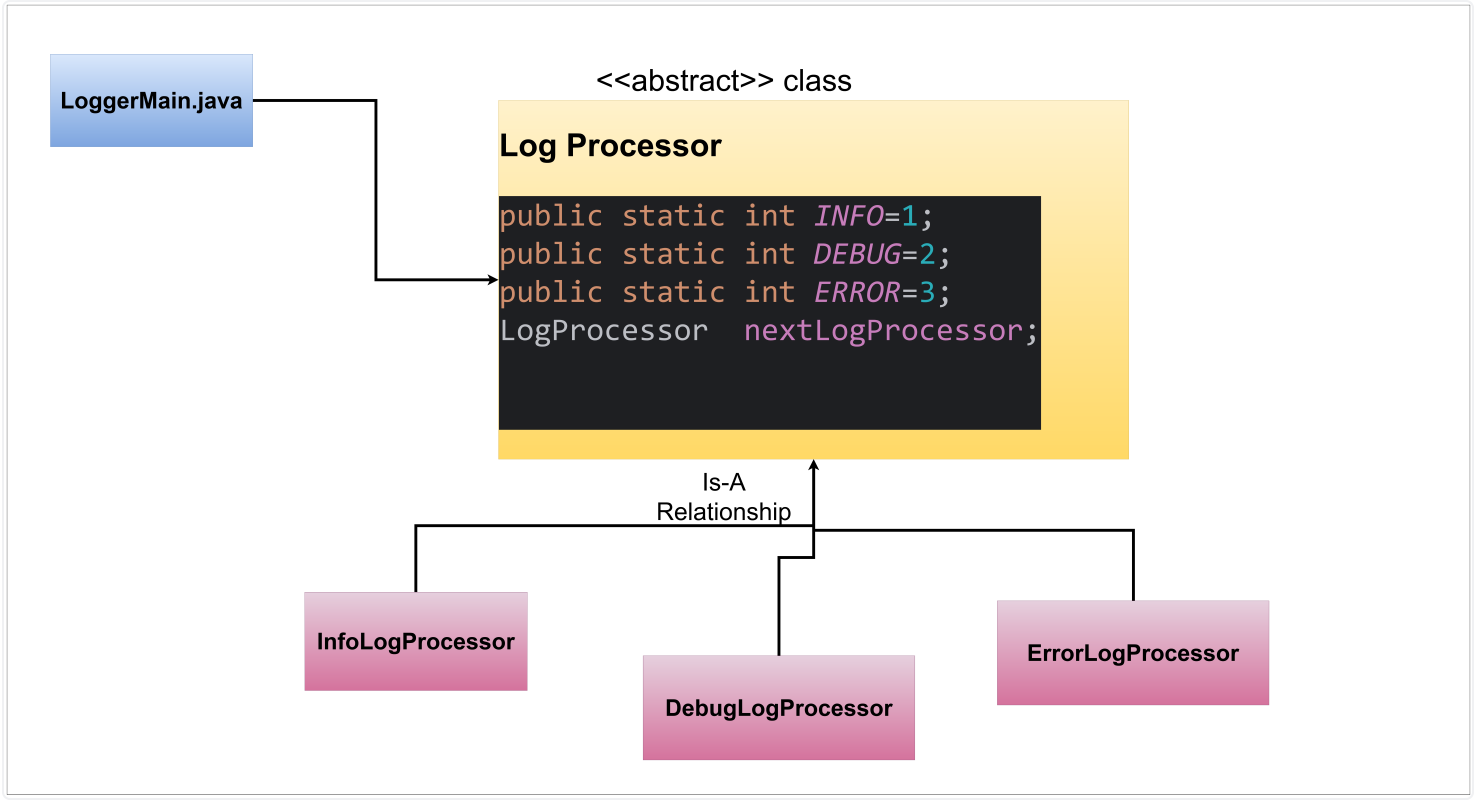
Chain Of Responsibility Design Pattern.
Application Usage:

- ATM Machine
- Design Logger [**Amazon**]
- Vending Machine [user can ask for water,pepsi,cola and daru].

Chain Of Responsibility Design Pattern.

- When a sender send a request and they doesn't care about who gonna full fill the request.
- There is a chain of receiver Object form obj-1 to obj-n.
- request goes to receiver ob1 then receiver-obj1 check , whether reciver-obj1 is able to full fill that request or not.


ATM
Client / Sender
withdraw 2000 ruppes
Request
Receiver OBJ1
Receiver OBJ2
Receiver OBJ3
Receiver OBJ-N
1000 Handler
500 Handler
2000 Handler
If any recever obj not able to full fill the request it send's actual problem message
Client / Sender
<<abstract>> class
Handler / Receiver / Processor
Next
Is-A Relationship
Handler 1
Handler 2



code:

```
package System_Design.DesignPatterns.chainResponsibilityDesignPattern;

public abstract class LogProcessor {
   public static int INFO=1;
   public static int DEBUG=2;
   public static int ERROR=3;

   LogProcessor  nextLogProcessor;
   LogProcessor(LogProcessor logProcessor){
       this.nextLogProcessor=logProcessor;
   }

   public void log(int logLevel, String message){
       if (nextLogProcessor!=null){
           nextLogProcessor.log(logLevel,message);
       }
   }
}
```

```
package System_Design.DesignPatterns.chainResponsibilityDesignPattern;

public class InfoLogProcessor extends LogProcessor {
    InfoLogProcessor(LogProcessor logProcessor) {
        super(logProcessor);
    }
```

```
    @Override
    public void log(int logLevel, String message) {
        if(logLevel==INFO){
            System.out.println("INFO: "+message);
        } else {
            super.log(logLevel, message);
        }

    }
}
```

```
package System_Design.DesignPatterns.chainResponsibilityDesignPattern;

public class DebugLogProcessor extends LogProcessor{
    DebugLogProcessor(LogProcessor logProcessor) {
        super(logProcessor);
    }

    @Override
    public void log(int logLevel, String message) {
        if(logLevel==DEBUG){
            System.out.println("ERROR: "+message);
        }else{
            super.log(logLevel, message);
        }
    }
}
```

```
package System_Design.DesignPatterns.chainResponsibilityDesignPattern;

public class ErrorLogProcessor extends LogProcessor{
    ErrorLogProcessor(LogProcessor logProcessor) {
        super(logProcessor);
    }

    @Override
    public void log(int logLevel, String message) {
        if(logLevel==ERROR){
            System.out.println("ERROR: "+message);
        } else {
            super.log(logLevel, message);
        }

    }
}
```

```
package System_Design.DesignPatterns.chainResponsibilityDesignPattern;

public class LoggerMain {

   public static void main(String[] args){
       LogProcessor logProcessor = new InfoLogProcessor( new DebugLogProcessor( new
ErrorLogProcessor(null)) );

       logProcessor.log(LogProcessor.ERROR,"Exceptions Happened");
       logProcessor.log(LogProcessor.DEBUG,"DEBUGGING");
       logProcessor.log(LogProcessor.INFO,"INFO ABOUT COOL");
   }


}
output
ERROR: Exceptions Happened
ERROR: DEBUGGING
INFO: INFO ABOUT COOL

Process finished with exit code 0
```