Chain of Responsibility Design Pattern

- This pattern sends data to an object and if that object can't use it, it sends it to any number of other objects that may be able to use it.
  - Create 4 objects that can either add, subtract, multiply, or divide.
  - Send 2 numbers and a command and allow these 4 objects to decide which can handle the requested calculation.

```
//===================================
// AddNumbers.java
public class AddNumbers implements Chain{

        private  Chain nextInChain;

        // Defines the next Object to receive the
        // data if this one can't use it

        public void setNextChain(Chain nextChain) {

                nextInChain = nextChain;

        }

        // Tries to calculate the data, or passes it
        // to the Object defined in method setNextChain()

        public void calculate(Numbers request) {

                if(request.getCalcWanted() == "add"){

                        System.out.print(request.getNumber1() + " + " +
                            request.getNumber2() + " = "+
                                        (request.getNumber1()+request.getNumber2()));

                } else {

                        nextInChain.calculate(request);

                }
        }
}

//===================================
// Chain.java
// The chain of responsibility pattern has a
// group of objects that are expected to between
// them be able to solve a problem.
// If the first Object can't solve it, it passes
// the data to the next Object in the chain

public interface Chain {

        // Defines the next Object to receive the data
        // if this Object can't process it

        public void setNextChain(Chain nextChain);

        // Either solves the problem or passes the data
        // to the next Object in the chain

        public void calculate(Numbers request);

}

//===================================
// DivideNumbers.java
public class DivideNumbers implements Chain{

        private  Chain nextInChain;
```

```
        @Override
        public void setNextChain(Chain nextChain) {

                nextInChain = nextChain;

        }

        @Override
        public void calculate(Numbers request) {

                if(request.getCalcWanted() == "div"){

                        System.out.print(request.getNumber1() + " / " +
                            request.getNumber2() + " = "+
                                        (request.getNumber1()/request.getNumber2()));

                } else {

                        System.out.print("Only works for add, sub, mult, and div");

                }
        }
}

//===================================
// MultNumbers.java
public class MultNumbers implements Chain{

        private  Chain nextInChain;

        @Override
        public void setNextChain(Chain nextChain) {

                nextInChain = nextChain;

        }

        @Override
        public void calculate(Numbers request) {

                if(request.getCalcWanted() == "mult"){

                        System.out.print(request.getNumber1() + " * " +
                            request.getNumber2() + " = "+
                                        (request.getNumber1()*request.getNumber2()));

                } else {

                        nextInChain.calculate(request);

                }
        }
}

//===================================
// Numbers.java
// This object will contain 2 numbers and a
// calculation to perform in the form of a String

public class Numbers {

        private int number1;
        private int number2;
```

```
        private String calculationWanted;

        public Numbers(int newNumber1, int newNumber2, String calcWanted){

                number1 = newNumber1;
                number2 = newNumber2;
                calculationWanted = calcWanted;

        }

        public int getNumber1(){ return number1; }
        public int getNumber2(){ return number2; }
        public String getCalcWanted(){ return calculationWanted; }

}

//====================================
// SubtractNumbers.java
public class SubtractNumbers implements Chain{

        private  Chain nextInChain;

        @Override
        public void setNextChain(Chain nextChain) {

                nextInChain = nextChain;

        }

        @Override
        public void calculate(Numbers request) {

                if(request.getCalcWanted() == "sub"){

                        System.out.print(request.getNumber1() + " - " +
                            request.getNumber2() + " = "+
                                        (request.getNumber1()-request.getNumber2()));

                } else {

                        nextInChain.calculate(request);

                }

        }
}

//====================================
// TestCalcChain.java
public class TestCalcChain {

        public static void main(String[] args){

                // Here I define all of the objects in the chain

                Chain chainCalc1 = new AddNumbers();
                Chain chainCalc2 = new SubtractNumbers();
                Chain chainCalc3 = new MultNumbers();
                Chain chainCalc4 = new DivideNumbers();

                // Here I tell each object where to forward the
                // data if it can't process the request
```

```
                chainCalc1.setNextChain(chainCalc2);
                chainCalc2.setNextChain(chainCalc3);
                chainCalc3.setNextChain(chainCalc4);

                // Define the data in the Numbers Object
                // and send it to the first Object in the chain

                Numbers request = new Numbers(4,2,"add");

                chainCalc1.calculate(request);

        }
}
```