Question 1

(1) Layered Architecture uses separation of concerns because it separates the scope of the project into four layers: presentation layer, business layer, persistence layer, and database layer. It also uses abstraction, because the highest layer where the graphical user interface is located presents the data and hides the complexities of the other layers. Lastly, it uses modularity, because each layer is broken down into components or modules.

(2) Client-Server Architecture uses separation of concerns because it separates processes between the client and server. It also uses abstraction because the client side is only shown the user interface(web app, mobile app) and never aware of the requests on the server side. Lastly, it uses generality, because the architecture can be used in multiple contexts(e.g email, gaming, web, file).

(3) MVC Architecture separation of concerns because the aspects of the software are separated into three categories: model, view, and controller. It also uses abstraction because the client will only see the data presented in the view while the manipulation of data between the controller and model is kept hidden.

Question 2

Explanation:

```
FileInputStream fin = new FileInputStream(filePath);
CompilationUnit cu = StaticJavaParser.parse(fin);
```

First, we read in the Java File, SimpleComparison.java, using FileInputStream. Then we parse the Java file into a CompilationUnit using JavaParser.

```
cu.accept(new ModifierVisitor<Void>() {
           public IfStmt visit(IfStmt n, Void arg) {
               if (n.getCondition() instanceof BinaryExpr) {
                   BinaryExpr condition = (BinaryExpr)n.getCondition();
                   if (condition.getOperator() == BinaryExpr.Operator.NOT_EQUALS &&
n.getElseStmt().isPresent()) {
                       condition.setOperator(BinaryExpr.Operator.EQUALS);
                       IfStmt newIfStmt = new IfStmt(condition, n.getElseStmt().get(),
n.getThenStmt());
                       return newIfStmt;
                   }
               }
               return n;
           }
       }, null);
```

Then, we modify the AST using a visitor design pattern to visit “if statement” nodes. We do this by checking if a conditional statement has two operands and an operator. After verifying that the condition is an instance of binary expression (BinaryExpr), we check if the operator is “!=” and that an “else statement” is associated with the “if statement”. If true, then we set the operator of the current “if statement” to “==” and create a new if statement (IfStmt) using the new operator and switching the else statement and then statement. We save all the changes into our CompilationUnit.

```
FileOutputStream fout = new
FileOutputStream("/Users/johangonzalez/hw-1/src/main/resources/SimpleComparisonModifie
d.java");
       String refactorOutput = cu.toString();
       fout.write(refactorOutput.getBytes());
       fout.close();
```

Next, we save our changes into a new file using FileOutputSteam by converting the CompilationUnit into a string and writing it into a file called SimpleComparisonModifed.java

```
YamlPrinter printer = new YamlPrinter(true);
       String yaml = printer.output(cu);
       FileOutputStream fo = new
FileOutputStream("/Users/johangonzalez/hw-1/src/main/resources/RefractoredAST.yaml");
       fo.write(yaml.getBytes());
       fo.close();
```

Lastly, we create a yaml file using YamlPrinter.

Commands:

```
mvn clean
```

```
mvn package
```

```
java -cp target/hw-1-1.0-SNAPSHOT-jar-with-dependencies.jar
com.w24.cs180.Main
```