Question 5) Design Principles

1) This segment of code violates the Single Responsibility Principle and also the Open/Closed Principle. It violates SRP because it allows multiple types of data to be passed to the class; a superior design would include separate classes for the MEMORY_RESOURCE, SOCKET_RESOURCE, and FILE_RESOURCE. It also violated OCP because adding a new memory type means that you have to modify the entire ResourceAllocator class, both for allocating and deallocating; this will make debugging and further modification much more difficult than necessary.

2) This segment of code clearly violates the Don't Repeat Yourself Principle. Rather than having three very similar addListener methods for three different additions; they all include the same try-catch inside the method bodies. We could have avoided this code duplication by adding a new method to handle adding listeners to a component: a simple helper method to add a listener to a given feature would have eliminated this duplication. For example, addListenerHelper(JComponent J) could have been called for each of these, with the duplicated code

```
for (Socket s : this.sockets) {
                    try {
                        OutputStreamWriter writer = new
OutputStreamWriter(s.getOutputStream());
                        writer.write(J.getText());

                    } catch (Exception ex) {
                        /* -- Error Handling -- */
                    }
                }
```

inside of this method.

3) This segment of code violates the Liskov Substitution Principle. Since Set is a subclass of Multiset, it is supposed to have at least all of the functionality of the Multiset. However, it is missing some of the operations necessary. To fix this, we would need to add the appropriate override methods to the Set class (count and remove).

4) This segment of code could have been better at using the Don't Repeat Yourself and Liskov Substitution Principles. The createUser class seems to be completely unnecessary, as the constructor is Public and can always be called instead of this. This creates problems if we were to create any subclasses of User, because the createUser method would not return an instance of the child class. This is a minor detail, but a better design would not have included this extra method. Also, there could probably be better design for the e-mailing system using an e-mail helper class which will be called anytime it is necessary to send an e-mail to the User, and then it could check which type of e-mail to send based on the registered flag for the User.