Homework #3
Design Decisions

Nathan Balon
CIS 550
Object Oriented Programming

Program three built on the existing classes that were created in program two. In the assignment, five new classes were created they are: `PersonAdapter`, `PersonAdapterImpl`, `PersonList`, `PersonListImpl` and `personListClient`. The purpose of the assignment is to create remote object references and then add the references to a list on a remote server.

`PersonAdapter` is an interface that defines the methods that are available to a class that implements this interface. The `PersonAdapter` extends `java.rmi.Remote` which is serves basically as a marker interface used to indicate that objects that implement this interface may be invoked remotely. Also, each method in the interface must declare that it throws a `RemoteException`. The methods declared in `PersonAdapter` get information about a person; a few examples are `getName` and `printDescription`.

`PersonList` is another interface that extends `java.rmi.Remote`. This interface is used by the `PersonListImpl`. This interface is used to add and remove people from the remote list that is contained on the server and to print a description of all the people that are contained on the server. The purpose of this interface is similar to the `PersonAdapter` interface.

The remaining new classes for the university application are concrete classes. `PersonAdapterImpl` is an adapter class which uses the adapter design pattern. The adapter is used so that the existing classes can be used with no modification. This class is needed because the existing classes of type `Person` already extend other classes and java only supports single inheritance. `PersonAdapterImpl` extends `UnicastRemoteObject` and implements `PersonAdapter`. The class has a data member of type `Person` and uses this data member to call the methods available to `Person` object.

`PersonListImpl` is another class that extends `UnicastRemoteObject`. This class is used as the server for the program. `PersonListImpl` implements the methods `addPerson`, `removePerson` and `listAll`. The methods `addPerson` and `RemovePerson` are used to add and remove remote references to the list on the server. The `ListAll` method is used to iterate through the list on the server and return a list of all people contained on the server. The class also has `main` method. In main a new `PersonListImpl` is created and bound to the rmiregistry. The `PersonListImpl` is bound to the registry by the name “PersonList”. To simplify the program the person was bound to the localhost. After call the `Naming.rebind` method the server is ready to use by the client program.

The final portion of the program is `personListClient`. This is the client portion of the program. The `personListClient` class has `main` method which is the entry point to client application. The first thing that is done in main is to use `Naming.lookup` to return a remote reference to the server’s `PersonListImpl` object. Once a reference has been returned to the client program at this point the user has a number of options concerning what they would like to do with the program.

The options available to the users of the program are to add a person, list all people, change a persons name and exit the program. The first option that a user has is they can

create a new `Person` by using the `PersonFactory` that was created earlier. Once a person object is created it is passed as an argument to the `PersonAdapterImpl` constructor to create a `UnicastRemoteObject`. After the adapter is created the `PersonAdapterImpl` object is then added to the server by calling the `addperson` method on the server. The second option that the user of the program has is to list all of the people that are contained on the server. This is implemented by calling a remote method on the server. The third option is used to show that the remote objects will remain consistent when a change is made. The user can change the name of a person by calling the method `setName` of `PersonAdapter`. The program will search through a list that is contained locally on the client. Once the person has been found in the list the name of the person will then be changed to a new name. If after the name is changed the user selects to display all of the people that are contained remotely on the server the changes will have propagated to the server, since the server is holding remote references to a `PersonAdapter` object. This is one of the benefits of using remote references, if the object were serialized and sent to the server, then change made on the client would not affect the object on the server since they would be copies instead of references. The last option that the user has is to exit the program. Exiting the program performs the necessary clean up by removing the reference from the local machine from the server when the program terminates.

Using Java RMI allows for methods to be called on remote objects. The one benefit that can be shown in this program is that if a change is made to a remote object's data member then the same value will be seen everywhere. For instance, if a remote reference on of `PersonApadter` changes the name of a person then when `listAll` is called on the server the new value of the name will be displayed there also. Using remote objects makes it easier to keep the objects in a consistent state. If a `Person` object on the other hand were passed to the server by value, then a change made on the client would not affect the object that is held on the server. It would be up to the programmer to keep the values consistent. A problem ran into when designing the application is if an object with a remote reference was created on the client and passed to the server and the program terminated and the server was still active and another client application was started that tried to display all of the `PersonAdpater` objects on the server then an exception would be thrown since these objects were garbage collected when the first program terminated. The reference in this case would have a null value, which would result in the exception being thrown. For this reason, when a client program terminates the remote references that were created by the client are removed from the server. By removing the references from the server's list avoids the problem of the server throwing exceptions. Another possible design decision would have been to send the `Person` objects by value to the server and then create the remote references to these objects on the server.