MODULARIZE YOUR CODE!

Modularization

- Separation of application and protocol layer is *extremely* helpful for:
  - Extension of application or protocol independently (e.g., not rebuilding from scratch for projects 2 and 3)
  - Writing clearer code that is easier to debug
  - Easier thread organization:
    - 1 or 2 for I/O per socket, probably
    - any number for app

Modularization in Java

- Force yourself to separate app and protocol
- Use separate packages!
- Example of one way to do this:

App

```
package App;

public class App {
  Helper helper;
  boolean newData;
  Object[] data;
  public App(){
    newData = false;
    helper = new Helper("128.208.6.43", 9999, this);
    helper.start();
    mainLoop();
  }
…
}
```

Protocol

```
package Protocol;

public class Helper implements Runnable{
  Socket socket;
  BufferedInputStream input;
  BufferedOutputStream output;

  public Helper(String ip, int port, App app){
    this.app = app;
    socket = new Socket(ip, port);
    input = new BufferedInputStream(socket.getInputStream());
    output = new BufferedOutputStream(socket.getOutputStream());
  }
…
}
```

Protocol

```
public void run() {
  while (true){
    try {
      int bytesAvailable = input.available();
     if (bytesAvailable > 0){
       byte[] buffer = new byte[bytesAvailable];
       input.read(buffer);
       processRead(buffer);
     }
     …
}

public void processRead(byte[] buffer){
  // TODO: translate buffer into objects
  this.app.dataWasRead(objects);
}
```

App

```
public void dataWasRead(Object[] objs){
  data = objs;
  newData = true;
}

public void mainLoop(){
  while (true){
    if (newData){
      doThings(data);
      newData = false;
    }
  }
}
```

Abstracting An Example

- Email
- How should we separate protocol and application layers for an email client?

Abstracting An Example

- If we properly separate application and protocol, does either one affect the other?
- Could we implement email using HTTP as its protocol instead of SMTP?
- Should we?

Note for Project 1

- Client should automatically re-register before the time runs out:

```
Timer timer = new Timer();
timer.schedule(new TimerTask(){
    @override public void run(){
        reregister();
    }
}, timeout - 1000);
```