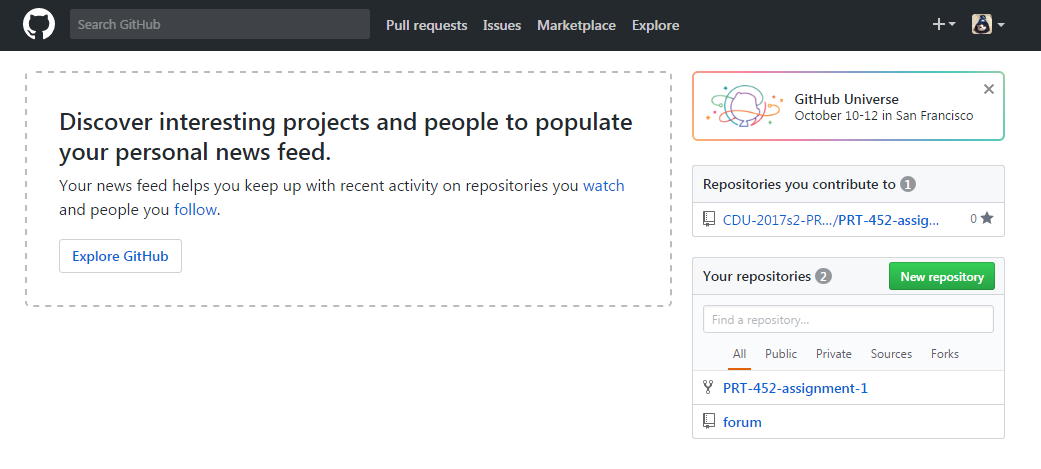
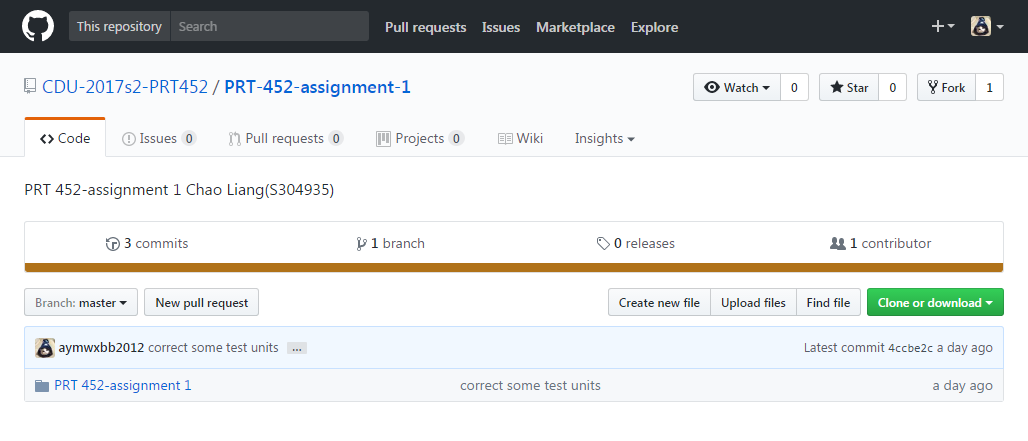
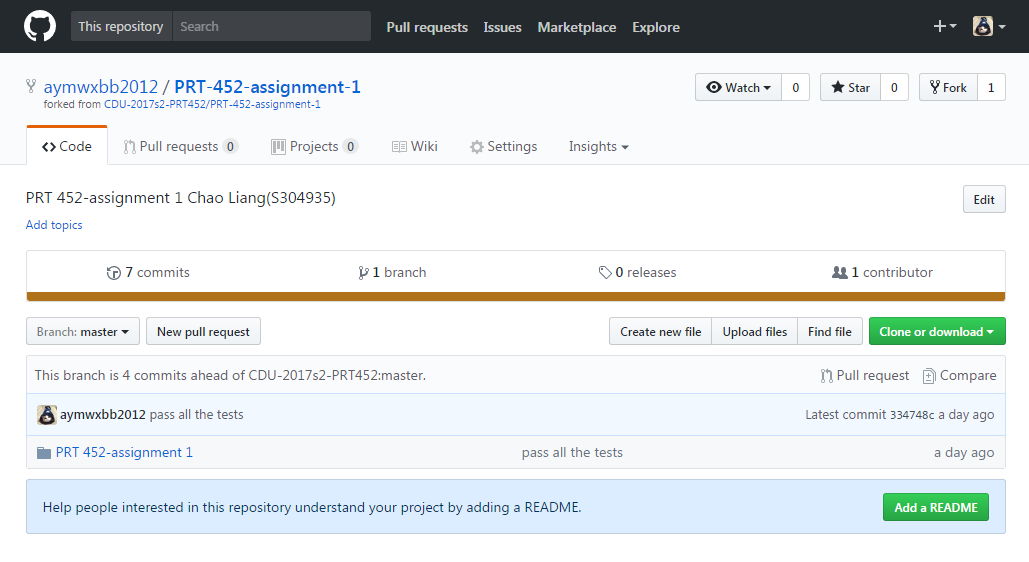
**Question1**

Answer:





At first, I submitted my assignment (including programming code) into this organization repository.

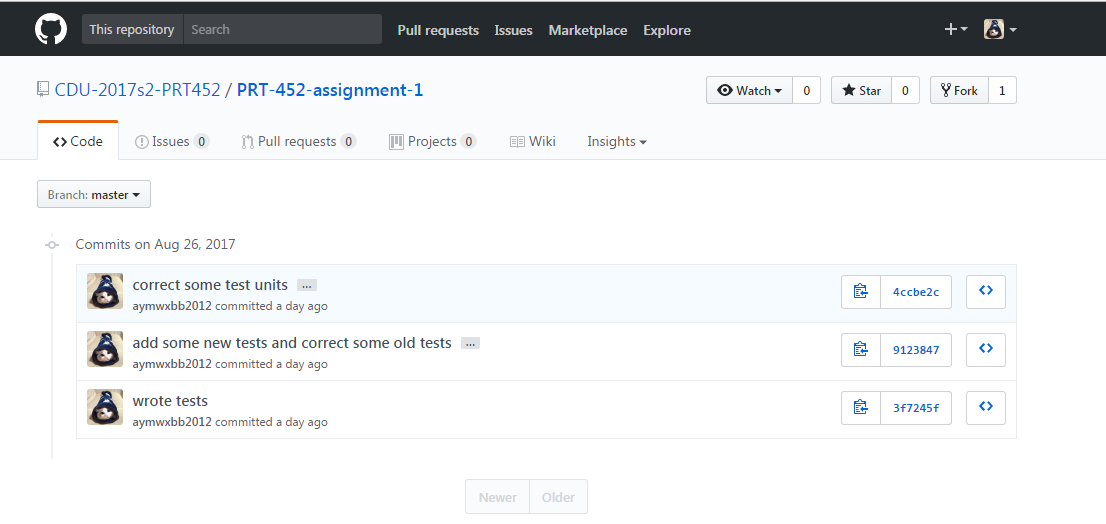


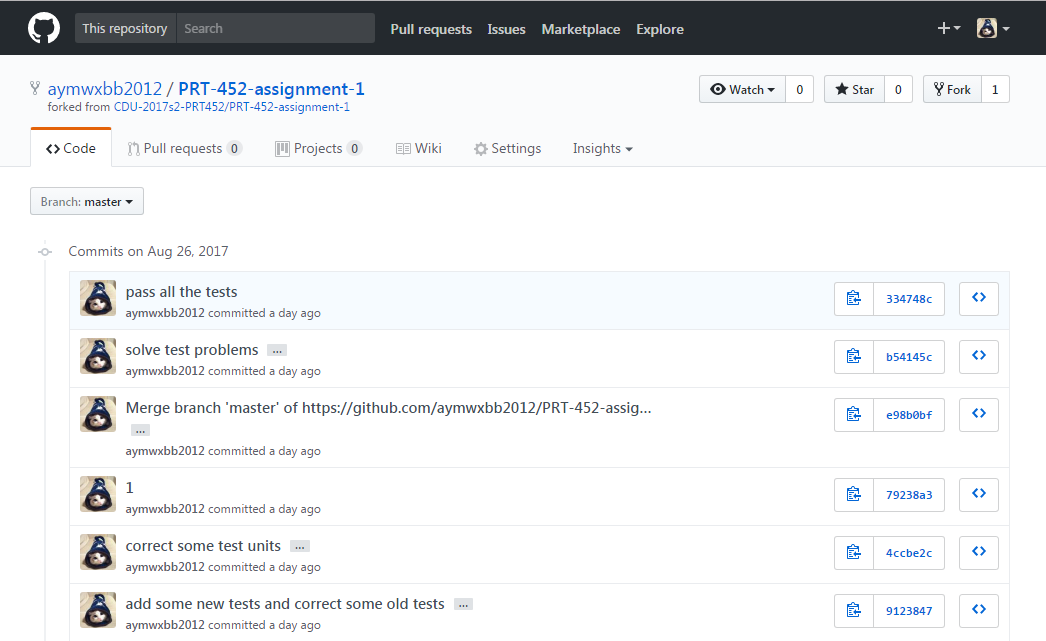
Then, I fork them into my own repository and finished them.

<https://github.com/aymwxbb2012/PRT-452-assignment-1>

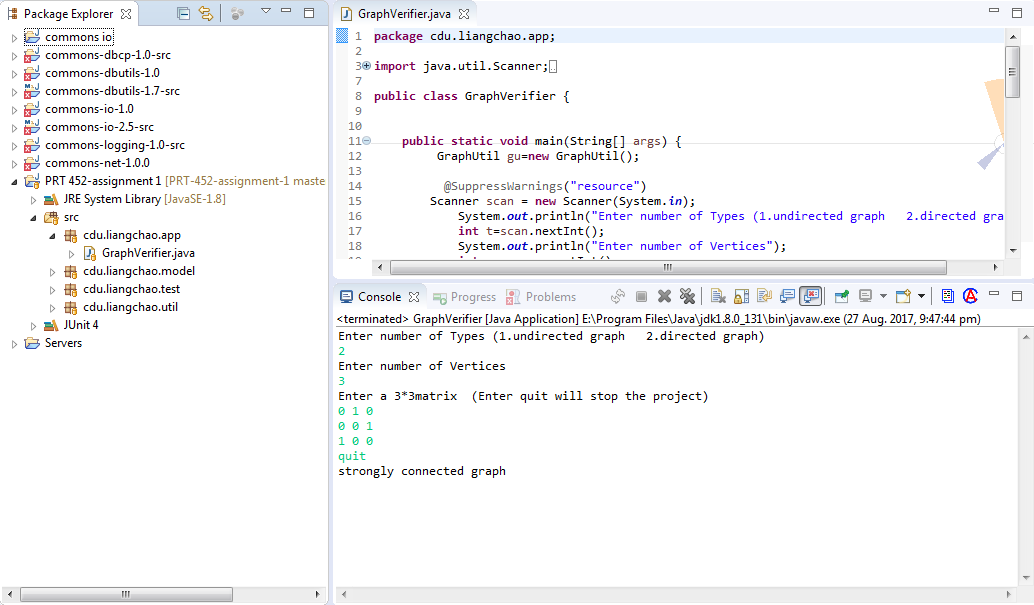
**Question2**

Answer:





<https://github.com/aymwxbb2012/PRT-452-assignment-1>



**Question3**

Answer:

Example 1: Switch statements

**commons-io-2.5-src/src/main/java/org/apache/commons/io/FileSystemUtils.java**

**long** freeSpaceOS(**final** String path, **final** **int** os, **final** **boolean** kb, **final** **long** timeout) **throws** IOException {

**if** (path == **null**) {

**throw** **new** IllegalArgumentException("Path must not be null");

}

**switch** (os) {

**case** ***WINDOWS***:

**return** kb ? freeSpaceWindows(path, timeout) / FileUtils.***ONE\_KB*** : freeSpaceWindows(path, timeout);

**case** ***UNIX***:

**return** freeSpaceUnix(path, kb, **false**, timeout);

**case** ***POSIX\_UNIX***:

**return** freeSpaceUnix(path, kb, **true**, timeout);

**case** ***OTHER***:

**throw** **new** IllegalStateException("Unsupported operating system");

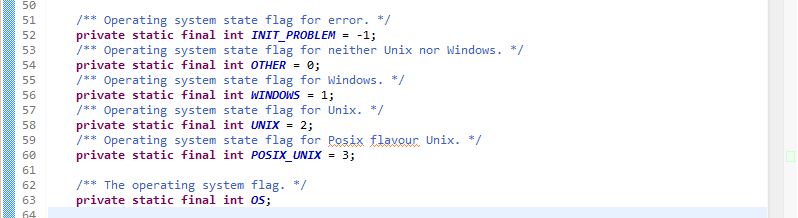
**default**:

**throw** **new** IllegalStateException(

"Exception caught when determining operating system");

}

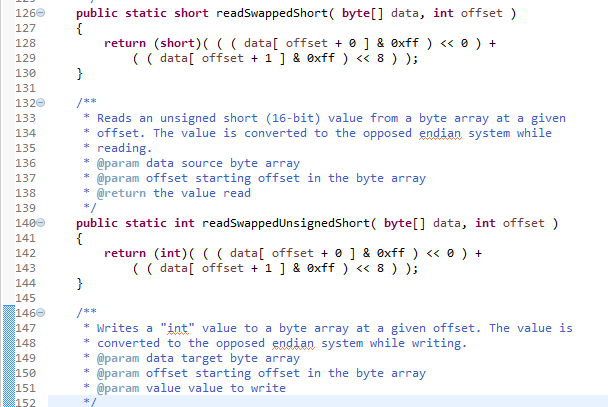
}

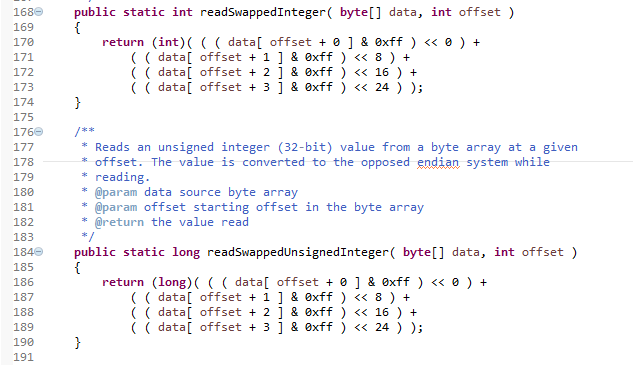


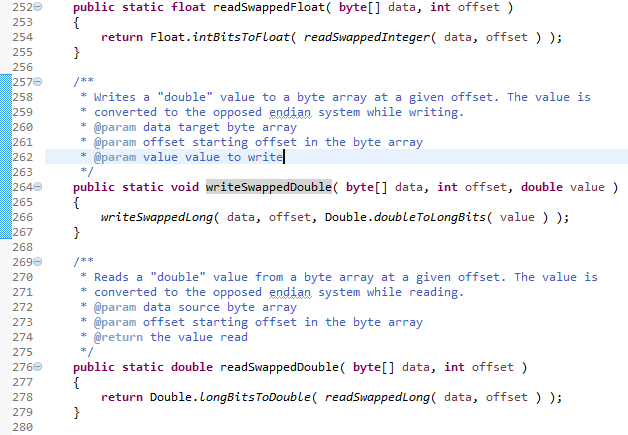
From my point of view, this is switch statements. “WINDOWS”, “UNIX”, “POSIX\_ UNIX” and “OTHER” are a list of type code. When a new condition is added, people have to find all the switch code and modify it. People can think of polymorphism and use Replace Type Code with Subclasses or Replace Type Code with State/Strategy to modify it. Because there are not too many conditions, polymorphism will be superfluous. People can separate this method into multiple smaller methods by using Replace Parameter with Explicit Methods and change the switch accordingly.

Example 2: Data clumps

**/commons-io-1.0/src/java/org/apache/commons/io/EndianUtils.java**







I think that this is data clumps. “**byte**[] data” and “**int** offset” often appear together in the code. People can use Introduce Parameter Object to set them off and become a new class so as to improve understanding and organization of code and reduce the size of code. But if people move only these parameters to a new class, this will become a Data Class.

Example 3: Message chains

**/commons-dbcp-1.0-src/src/java/org/apache/commons/dbcp/PoolingDriver.java**

**synchronized** **public** ObjectPool getPool(String name) {

ObjectPool pool = (ObjectPool)(*\_pools*.get(name));

**if**(**null** == pool) {

InputStream in = **this**.getClass().getResourceAsStream(String.*valueOf*(name) + ".jocl");

**if**(**null** != in) {

JOCLContentHandler jocl = **null**;

**try** {

jocl = JOCLContentHandler.*parse*(in);

} **catch**(Exception e) {

**throw** **new** DbcpException(e);

}

**if**(jocl.getType(0).equals(String.**class**)) {

pool = getPool((String)(jocl.getValue(0)));

**if**(**null** != pool) {

registerPool(name,pool);

}

} **else** {

pool = ((PoolableConnectionFactory)(jocl.getValue(0))).getPool();

**if**(**null** != pool) {

registerPool(name,pool);

}

}

}

}

**return** pool;

}

This is a message chain. JOCLContentHandler requests PoolableConnectionFactory, and PoolableConnectionFactory requests Pool. This means that JOCLContentHandler has to rely on the navigation between PoolableConnectionFactory and Pool. If any modification happens in these relationships, people have to change JOCLContentHandler. People can reduce the message chain by using Hide Delegate. By deleting message chain, the client code does not need to know too many details of the relationship of objects and the code size can be reduced. However, if people use too many Hide Delegate, understanding the functionality and operation of the code will become difficult, and may cause the Middle Men problem.

Example 4: Speculative generality

**Commons NetNET-242:**

**Method createServerSocket of FTPSSocketFactory will never be called and thus UseClientMode is incorrect in a secured ftp transfer using active mode.**

<https://issues.apache.org/jira/browse/NET-242?jql=project%20%3D%20NET%20AND%20status%20in%20(Resolved%2C%20Closed)%20AND%20text%20~%20%22delete%22%20ORDER%20BY%20key%20DESC>

Index: FTPSSocketFactory.java

===================================================================

--- FTPSSocketFactory.java (revision 712101)

+++ FTPSSocketFactory.java (working copy)

@@ -20,13 +20,11 @@

import java.io.IOException;

import java.net.InetAddress;

-import java.net.ServerSocket;

import java.net.Socket;

import java.net.UnknownHostException;

import javax.net.SocketFactory;

import javax.net.ssl.SSLContext;

-import javax.net.ssl.SSLServerSocket;

/\*\*

@@ -63,20 +61,5 @@

return this.context.getSocketFactory().createSocket(address, port, localAddress, localPort);

}

- public ServerSocket createServerSocket(int port) throws IOException {

- return this.init(this.context.getServerSocketFactory().createServerSocket(port));

- }

-

- public ServerSocket createServerSocket(int port, int backlog) throws IOException {

- return this.init(this.context.getServerSocketFactory().createServerSocket(port, backlog));

- }

-

- public ServerSocket createServerSocket(int port, int backlog, InetAddress ifAddress) throws IOException {

- return this.init(this.context.getServerSocketFactory().createServerSocket(port, backlog, ifAddress));

- }

-

- public ServerSocket init(ServerSocket socket) throws IOException {

- ((SSLServerSocket) socket).setUseClientMode(true);

- return socket;

- }

+

}

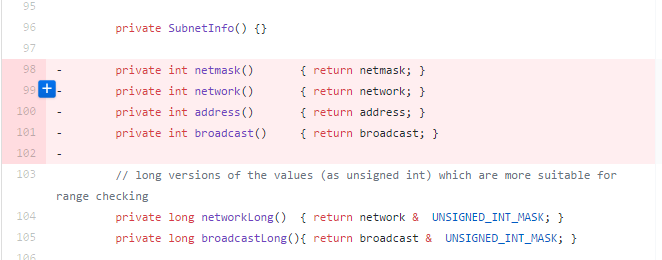
This bug is speculative generality. According to the description in the web site, the method createServerSocket is never used, because people can create ServerSockets by using a ServerSocketFactory. People can use Inline Method to get rid of unused class, methods and parameters, or just delete them.

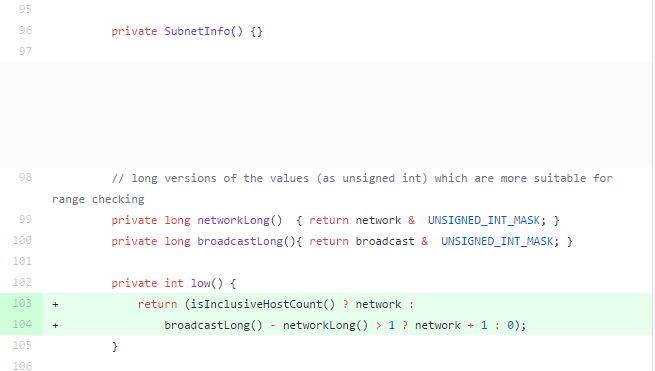
Example 5: Speculative generality

**Commons NetNET-621**

**SubnetUtils#SubnetInfo - remove unnecessary accessors**

[**https://issues.apache.org/jira/browse/NET-621?jql=project%20in%20(DBCP%2C%20IO%2C%20DBUTILS%2C%20LOGGING%2C%20NET)%20AND%20text%20~%20%22remove%20unnecessary%22**](https://issues.apache.org/jira/browse/NET-621?jql=project%20in%20(DBCP%2C%20IO%2C%20DBUTILS%2C%20LOGGING%2C%20NET)%20AND%20text%20~%20%22remove%20unnecessary%22)





This is Speculative generality. According to the description in the web site, The methods, including “private int netmask() { return netmask; }”,”private int network() { return network; }”,”private int address() { return address; }”,”private int broadcast() { return broadcast; }”, do not actually work. These methods is created for the future, but not necessary. These methods will make the code hard to understand and support.

**References:**

FOWLER, M., BECK, K., BRANT, J., OPDYKE, W. & ROBERTS, D. (1999) Refactoring: Improving the Design of Existing Code, Addison Wesley.

GAMMA, E., HELM, R., JOHNSON, R. & VLISSIDES, J. (1995) Design patterns : elements of reusable object-oriented software, Reading, Mass., Addison-Wesley.