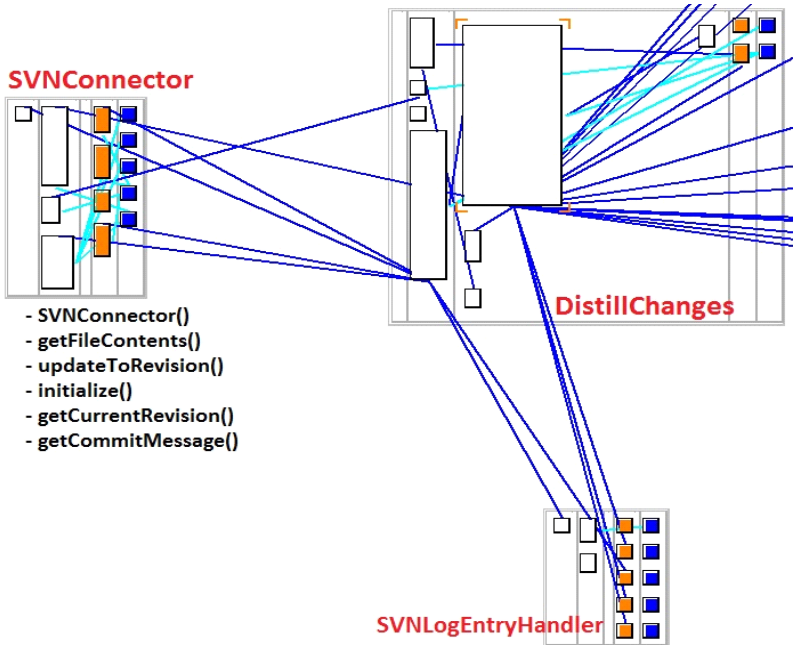
RFC: Refactory plan for ChEOPSJ

Introduction

After looking at the class blueprints and their interactions in codecity, we learned that the classes SVNConnector and SVNLogEntryHandler are only being used by the class DistillChanges (more precisely only in its methods: ExtractChangesFromJavaFiles(), iterateRevisions() and updateOneRev(). We also noticed that both SVN related classes almost have no interaction with each other!



We can see that there are 11 methods being used (6 of the SVNConnector class, 5 of the SVNLogEntryHandler) by the DistillChanges class, these will be provided in common interfaces.

For making our refactoring succesful and preventing risks, we won't use a big bang integration. In one of the SVNConnector methods (getCommitMessage) we see that it uses SVNLogEntryHandler. Therefore we will first focus on refactoring the SVNLogEntryHandler and afterwards refactor the SVNConnector. It's very important to work in schematic well ordered steps!

1 Refactoring of LogEntryHandler

Aspects

- LogEntryHandler Interface with shared functions that are currently in the SVNLogEntryHandler Class
- For each kind of repository (Git, SVN, …) a logentryhandler class (SVNLogEntryHandler, GITLogEntryHandler, …) which implements the LogEntryHandler interface.
- Factory pattern to create the LogEntryHandler of choice
- Singleton pattern, because there can be only one factory object, SVNLogEntryHandler object, GITLogEntryHandler object, ...
- The use of a property file instead of hardcoded configurations.

  Possible ways to handle this: ordered from worst to best:

  1) Hardcoded Configs (currently)
  2) Config XML/property file
  3) Own Plugin with preferences (however this goes out of the scope of our time frame and project description)

Package Structure

The package 'svnconnection' will be renamed to connections.

```
–  be.ac.ua.ansymo.cheopsj.distiller.connections
   -- RepositoryFactory.java (factory)
   -- connections (package)
         --- RepositoryConnector (interface)
         --- impl (package)
                ---- GITConnector (class)
                ---- SVNConnector (class)
   -- loghandler (package)
         --- RepositoryLogHandler (interface)
         --- impl (package)
                ---- GITLogEntryHandler (class)
                ---- SVNLogEntryHandler (class)
```

Properties file

Because the location of a file is rather hard to find at runtime when using plugins, the location to the file can be passed to the factory, this way the factory can read what kind of object it should return, it will be passed further on to the LogEntryHandler class itself, so those can read the url, username and passwords out of it!

The structure would be something like:

```
type_in_use = SVN
SVN_username =  abc
SVN_password = def
SVN_url = http:...

GIT_username = xyz
GIT_password = klm
GIT_url = http:...
```

LogEntryHandler <interface>

```
interface LogEntryHandler{
   getChangedPaths();
   String getMessage();
   String getUser();
   Date getDate();

}
```

SVNLogEntryHandler & GITLogEntryHandler <class>

```
class SVNLogEntryHandler implements LogEntryHandler{
   SVNLogEntryHandler();
   //implementation of the 5 functions of the interface
   LogEntryHandler getLogEntryHandler(File configFile);
   //will be explained below

}

class GITLogEntryHandler implements LogEntryHandler{
   GITLogEntryHandler();
   //implementation of the 5 functions of the interface
   LogEntryHandler getLogEntryHandler(File configFile);
   //will be explained below
```

```
}
```

Possible improvement for the future

In case we notice that there is a lot of shared common functionality between all LogEntryHandler classes, we could add an abstract class which gets extended, this way common private functions can be used. However our first goal is making it work, then make it better!

RepositoryFactory

The factory will provide the function:
`LogEntryHandler RepositoryFactory.getLogEntryHandler(File configFile);`

which returns an object that implements the interface LogEntryHandler.

Singleton

The repositoryFactory (as well as the Handlers themselves) are singletons, therefore only one object can be created out of it. This is done as follows:

```
public class RepositoryFactory{

    static private RepositoryFactory singleton;

    private RepositoryFactory();
    //this is made private so other classes can't create a new object by just calling
    new RepositoryFactory();

    private static RepositoryFactory getFactory(){
           if(singleton == null){
                  singleton = new RepositoryFactory();
           }
           return singleton;
    }

    public static LogEntryHandler getLogEntryHandler(File configFile){
           RepositoryFactory factory = RepositoryFactory.getFactory(File configFile);
           LogEntryHandler LEH = factory.instantiateHandler();
           return LEH;
    }

    private LogEntryHandler instantiateHandler(File configFile){
           String type = fetchFromPropertiesFile(File,type_in_use);

           if(type.equalsIgnoreCase("svn"){
                  return SVNLogEntryHandler.getLogEntryHandler();
           }else if(type.equalsIgnoreCase("git"){
                  return GITLogEntryHandler.getLogEntryHandler();
           }
    }

}
```

SVNLogEntryHandler getLogEntryHandler function (same for GITLogEntryHandler)

```
public static LogEntryHandler getLogEntryHandler(){
    if(singleton == null){
           singleton = new SVNLogEntryHandler();
    }
```

```
    return singleton;
}
```

Singleton

Like said above, the LogEntryHandler classes themselves are singletons too, therefore they should also have:

```
static private SVNLogEntryHandler singleton;
private SVNLogEntryHandler();
```

2 Refactoring of connector

Aspects

- RepositoryConnector Interface with shared functions that are currently in the SVNConnector
- For each kind of repository (Git, SVN, ...) a connector class (SVNConnector, GITConnector, ..) which implements the RepositoryConnector interface.
- Factory pattern to create the connector of choice
- Singleton pattern, because there can be only one factory object, SVNConnector object, GITConnector object, ...

RepositoryConnector <interface>

```
interface RepositoryConnector{
    void initialize();
    void updateToRevision(File, long, IProgressMonitor);
    long getCurrentRevision(File);
    void getCommitMessage(File , long , LogEntryHandler);
    String getFileContents(String, long );

}
```

SVNConnector & GITConnector <class>

```
class SVNConnector implements RepositoryConnector{
    SVNConnector();
    //implementation of the 5 functions of the interface
    RepositoryConnector getConnector(File configFile);
    //will be explained below

}

class GITConnector implements RepositoryConnector{
    GITConnector();
    //implementation of the 5 functions of the interface
    RepositoryConnector getConnector(File configFile);
    //will be explained below

}
```

Possible improvement for the future

In case we notice that there is a lot of shared common functionality between all Connector/LogEntryHandler classes, we could add an abstract class which gets extended, this way common private functions can be used. However our first goal is making it work, then make it better!

RepositoryFactory

Add functionality to the factory to create Connectors:

```
RepositoryConnector RepositoryFactory.getConnector(File configFile);
```

which returns an object that implements the interface RepositoryConnector. This will be done in the same way as mentioned above.

Singleton

```
public class RepositoryFactory{

    static private RepositoryFactory singleton;

    private RepositoryFactory();
    //this is made private so other classes can't create a new object by just calling
    new RepositoryFactory();

    private static RepositoryFactory getFactory(){
           if(singleton == null){
                  singleton = new RepositoryFactory();
           }
           return singleton;
    }

    public static RepositoryConnector getConnector(File configFile){
           RepositoryFactory factory = RepositoryFactory.getFactory();
           RepositoryConnection RC = factory.instantiateConnection(File configFile);
           return RC;
    }

    private RepositoryConnector instantiateConnection(File configFile){
           String type = fetchFromPropertiesFile(File,type_in_use);

           if(type.equalsIgnoreCase("svn"){
                  return SVNConnector.getConnector(File configFile);
           }else if(type.equalsIgnoreCase("git"){
                  return GITConnector.getConnector(File configFile);
           }
    }
```

SVNConnector's getConnector function (same for GITConnector)

```
public static RepositoryConnector getConnector(File configFile){
    if(singleton == null){
           String username = fetchFromPropertiesFile(File,SVN_username);
           String password = fetchFromPropertiesFile(File,SVN_password);
           singleton = new SVNConnector(username,password);
    }
    return singleton;
}
```

Singleton

Like said above, the Connector classes itself are singletons too, therefore they should also have:

```
static private SVNConnector singleton;
private SVNConnector(username,password);
```