University of Central Lancashire
UCLan

CO3409 Distributed Enterprise Systems

Lab : Reflection

Summary

Use reflection to examine features of a class, to link components to create an application, and to create a mock object to allow behavioural rather than functional unit testing.

Before continuing with this practical read this tutorial First!

Purpose

On completion of this, you should be able to

- Use reflection to examine features of a class
- Explain how a container could use reflection to implement dependency injection
- Create a mock object that imitates an instance of a class that implements an interface.

Things to Remember

Some things to remember...

- Run Netbeans as "Administrator".
- When running your application, if you are using a database please ensure your database is running first (see previous lab worksheets).

Activities

1. Examining Properties of a Class

1. Create a new Java Application from the "Java with Ant" category.

2. Create a new class `Account`, in the package `reflectionDemo`.

3. Replace the class with the contents of **Appendix 1**. Fix the imports. Add a getter for the balance field.

4. Create a new main class, `ReflectionDemo` in the package `reflectionDemo`.

   Replace the Main method with the contents of **Appendix 2**. Fix the imports.

5. Use the following documentation to extract and display some more information (e.g. the modifiers of the fields)
   - https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html
   - https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Method.html
   - https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Field.html

6. Use the following documentation and that of Field to directly set the private field on the object whose balance was set. Display the modified balance.
   - https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AccessibleObject.html

2. Creating a (very) simple container

1. In the package `reflectionContainer`, create appropriate interfaces and classes from **Appendices 3-7**. Fix the imports.

2. Create a class with a main method. Replace the main method with the code found at **Appendix 8**. Fix any imports.

3. Create a `Calculator` class that implements the `ICalculator` interface and extends `Component`. Implement the methods.

   > Hint: light bulb, then brain.

4. Run the application by right-clicking the file containing a standard Main function and choosing **Debug File**.

5. Read the application and explain how it works.

   > Hint: http://www.cs.sjsu.edu/faculty/pearce/modules/lectures/ooa2/cbd/implementation/index.htm

6. Look at carefully at the `Container` class from Activity 2. Explain the bug mentioned in the comments.

7. Add the `MultiMap` Class to the project. Replace the appropriate `Map` with a `MultiMap` and amend the `findProviders()` method to use the `MultiMap`.

3. Creating a Mock object to support behavioural testing

1. Create a new folder called `mockReflection` in your source packages folder.

2. Add the classes and interfaces from **Appendix 10, 11 and 12**.

3. Add getters and setters to `FileUser`.

4. Fix any imports.

5. Create a new class with a main method. Replace the main method with the code in **Appendix 13**.

6. Run the program. The test fails. Fix the `FileUser` class so that the test passes.

7. Amend the mock object to keep track of whether `openFile` is called before `readFile` and `closeFile` is called last.

8. Add a `getResults()` method that displays the results of the test (e.g. "test failed because `openFile()` not called". Make sure that the test fails. Fix the `FileUser` class.

9. Explain the program.

> Hint: http://www.kdgregory.com/index.php?page=junit.proxy)

4. Fixing the container to handle components requiring the same interface

1. Look at carefully at `Container` class from **Activity 2**. Explain the bug mentioned in the comments.

2. Add the `MultiMap` Class from **Appendix 9** to the project.

3. Replace the appropriate `Map` with a `MultiMap` and amend the `findProviders()` method to use the `MultiMap`.

Appendices

Appendix 1

```
@Retention(RetentionPolicy.RUNTIME)
@interface TestStatus {
    String   value() default "untested";
}

class Account {

    private Double balance = 0.0;

    public void withdraw(Double amt) {
        balance -= amt;
    }

    @TestStatus
    public void deposit(Double amt) {
        balance += amt;
    }

}
```

Appendix 2

```
public static void main(String[] args) {
    try {
      Class someClass = Account.class;

      Field[] someFields = someClass.getDeclaredFields();
      for (int i = 0; i < someFields.length; i++) {
        System.out.println(someFields[i].getName() + ": " +
someFields[i].getType());
      }

      Method someMethod = someClass.getMethod("deposit", Double.class);
      System.out.println("Method: " + someMethod.getName());
      Parameter[] params = someMethod.getParameters();
      for (int i = 0; i < params.length; i++) {
        System.out.println("Parameter "+ i +" " + params[i].getName() + ": "
                                          + params[i].getType().getName());
      }

      Annotation[] anns = someMethod.getAnnotations();
      for (Annotation a : anns) {
        System.out.println("Annotation type: " + a.annotationType().getName()
            + " as string: " + a.toString() + " value: " + ((TestStatus)
a).value());
      }

      System.out.println("depositing £100");
      Object someObject = someClass.newInstance();
      someMethod.invoke(someObject, 100.0);

      System.out.println("displaying balance");
      someMethod = someClass.getMethod("getBalance", (Class[]) null);
```

```
      System.out.println("balance = £" + someMethod.invoke(someObject,
(Object[]) null));

    } catch (IllegalAccessException | IllegalArgumentException |
InstantiationException
        | NoSuchMethodException | SecurityException | InvocationTargetException
e) {
      System.out.println(e);
    }
  }
```

Appendix 3

```
public interface App {
    void main() throws Exception;
}
```

Appendix 4

```
public interface ICalculator {

    public Double add(Double x, Double y);
    public Double mul(Double x, Double y);
    public Double sub(Double x, Double y);
    public Double div(Double x, Double y);
}
```

Appendix 5

```
public class Component {

    private Set<Class<?>> requiredInterfaces;
    private Set<Class<?>> providedInterfaces;
    private Map<Class<?>, Field> fields;
    protected Container container;

    public Component() {
         fields = new HashMap<Class<?>, Field>();
         requiredInterfaces = computeRequiredInterfaces();
         providedInterfaces = new HashSet<Class<?>>();
         Class<?>[] interfaces = this.getClass().getInterfaces();
         for(int i = 0; i < interfaces.length; i++){
             providedInterfaces.add(interfaces[i]);
         }
         container = null;
    }

    public Set<Class<?>> getProvidedInterfaces() {
        return providedInterfaces;
    }
```

```
    public Set<Class<?>> getRequiredInterfaces() {
        return requiredInterfaces;
    }

    public Container getContainer() {
        return container;
    }

    public void setContainer(Container container) {
        this.container = container;
    }

    public boolean addRequiredInterface(Class<?> e) {
        return requiredInterfaces.add(e);
    }

    public Iterator<Class<?>> iterator() {
        return providedInterfaces.iterator();
    }

    private Set<Class<?>> computeRequiredInterfaces() {
        Set<Class<?>> result = new HashSet<Class<?>>();
        Field[] fieldArray = this.getClass().getDeclaredFields();
        for(int i = 0; i < fieldArray.length; i++) {
            Class<?> fieldType = fieldArray[i].getType();
            if (fieldType.isInterface()) {
                result.add(fieldType);
                this.fields.put(fieldType, fieldArray[i]);
            }
        }
        return result;
    }

    // calls client.setField(provider)
    public void setProvider(Class<?> intf, Component provider) throws Exception
{
        Field field = fields.get(intf);
        if (field != null) {
            String name = field.getName();
            Character c = name.charAt(0);
            String setter = "set" + Character.toUpperCase(c) +
name.substring(1);
            Method m = this.getClass().getMethod(setter, field.getType());
            m.invoke(this, provider);
        }
    }
}
```

Appendix 6

```
public class Container {

  private Map<Class<?>, Component> providedInterfaces =
                                          new HashMap<Class<?>, Component>();

  // map values should be sets of components needing the interface:
  private Map<Class<?>, Component> requiredInterfaces =
```

```
                                        new HashMap<Class<?>, Component>();

  public void addComponent(Component component) throws Exception {
         component.setContainer(this);
         for(Class<?> intf: component.getProvidedInterfaces()) {
       providedInterfaces.put(intf,  component);
         }
         for(Class<?> intf: component.getRequiredInterfaces()) {
       requiredInterfaces.put(intf,  component);
         }
         findProviders();
  }

  private void findProviders() throws Exception {
    Set<Class<?>> reqInterfaces = requiredInterfaces.keySet();
    for(Class<?> intf: reqInterfaces) {
      Component client = requiredInterfaces.get(intf); // this should be a set
      Component provider = providedInterfaces.get(intf);
      if (client != null && provider != null) {
        client.setProvider(intf,  provider);
        requiredInterfaces.remove(intf);
      }
    }
  }

  public Component findProvider(Class<?> intf) {
    return providedInterfaces.get(intf);
  }


  public void launch() throws Exception {
    Component c = findProvider(App.class);
    ((App) c).main();
  }

}
```

Appendix 7

```
public class StatsCalculator extends Component implements App {

    private ICalculator arithmeticCalculator;

    public ICalculator getArithmeticCalculator() {
        return arithmeticCalculator;
    }

    // required by container:
    public void setArithmeticCalculator(ICalculator arithmeticCalculator) {
        this.arithmeticCalculator = arithmeticCalculator;
    }

    public Double mean(List<Double> data) throws Exception {
        Double sum = 0.0;
        for (Double val : data) {
            sum = arithmeticCalculator.add(sum, val);
```

```
        }
        Double avg = arithmeticCalculator.div(sum, (double) data.size());
        return avg;
    }

    public void main() throws Exception {
        List scores = new LinkedList();
        for (int i = 0; i < 100; i++) {
            scores.add((double) i);
        }
        Double avg = mean(scores);
        System.out.println("Average = " + avg);
    }
}
```

Appendix 8

```
public static void main(String[] args) {
        try {
            Container container = new Container();
            container.addComponent(new StatsCalculator());
            container.addComponent(new Calculator());
            container.launch();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

Appendix 9

```
public class MultiMap <K, V> {

    private final Map<K, Set<V>> body = new HashMap();

    Set <V> get(K key) {
    /* Returns the value to which the specified key is mapped,or null if this
       map contains no mapping for the key.
    */
       return body.get(key);
    }

    Set <V> put(K key, V value) {
    /* Associates the specified value with the specified key in this
       map(optional operation).
    */
      Set <V> val = body.get(key);
      if ( val == null){
          val = new HashSet<>();
      }
       val.add(value);
       return body.put(key, val);
    }
```

```
    Set<K> keySet() {
        return body.keySet();
    }

    void remove(K intf) {
         body.remove(intf);
    }
}
```

Appendix 10

```
public class FileProxy implements InvocationHandler
{
        public IFile toStub()
    {
        return (IFile)Proxy.newProxyInstance(
                        this.getClass().getClassLoader(),
                        new Class[] { IFile.class },
                        this);
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable
    {
        if (method.getName().equals("closeFile"))
        {
            return true;
        }
        if (method.getName().equals("openFile"))
        {
            return 999;
        }
        if (method.getName().equals("readFile"))
        {
            if (args.length != 1)
                throw new Exception("readFile Invalid number of parameters:
"+Integer.toString(args.length));
            if ((int)args[0] != 999)
                return null;
            return "hello world".toCharArray();
        }
        throw new UnsupportedOperationException(method.getName());
    }
}
```

Appendix 11

```
public interface IFile {

    int openFile(String fileName);
    char [] readFile(int handle);
    boolean writeFile(char [] data);
    boolean closeFile();

}
```

Appendix 12

```
public class FileUser {

    IFile file;

    FileUser(){ }

    int countChar(){
        int handle = file.openFile();
        char [] data = file.readFile(91);
        if (data == null)
            return 0;
        return data.length;
    }
}
```

Appendix 13

```
public static void main(String[] args) {
        FileProxy fp = new FileProxy();

        IFile af = (IFile)fp.toStub();
        FileUser fu = new FileUser("thefile");
        fu.setFile(af);
        int count = fu.countChar();
        if (count == "hello world".length())
            System.out.println("success");
        else
            System.out.println("fail");
    }
```