Politecnico di Milano, A.A. 2015/2016

Software Engineering 2: **C**ode **I**nspection

Belotti Nicola 793419
Chioso Emanuele 791621
Colombo Andrea 853381

January 2, 2016

Contents

1 Introduction

ciao

2 Assigned class and methods

All the methods we were assigned belong to the same class, the DeploymentDescriptorModel class

2.1 First Method: getConstructor

```
/** Returns a wrapped constructor element for the specified
    argument types in the class with the specified name. If the
    specified class name is a persistence-capable key class name
    which corresponds to a bean with an unknown primary key class a
    dummy constructor will also be returned. Types are specified as
    type names for primitive type such as int, float or as fully
    qualified class names.
    @param className the name of the class which contains the
        constructor to be checked
    @param argTypeNames the fully qualified names of the argument
        types
    @return the constructor element
    @see #getClass
    */

public Object getConstructor (final String className, String[]
    argTypeNames)
   {
      Object returnObject = null;

      if ((NameMapper.PRIMARY_KEY_FIELD ==
         getPersistenceKeyClassType(className)) &&
         Arrays.equals(argTypeNames, NO_ARGS))
      {
         returnObject = new MemberWrapper(className, null,
             Modifier.PUBLIC,
            (Class)getClass(className));
      }

      if (returnObject == null)
      {
         returnObject = super.getConstructor(className,
             argTypeNames);

         if (returnObject instanceof Constructor)  // wrap it
            returnObject = new
                MemberWrapper((Constructor)returnObject);
      }

      return returnObject;
   }
```

2.2 Second Method: getMethod

```
/** Returns a wrapped method element for the specified method name
    and argument types in the class with the specified name. If the
    specified className represents a persistence-capable class and
    the requested methodName is readObject or writeObject, a dummy
    method will be returned. Similarly, if the specified class name
    is a persistence-capable key class name which corresponds to a
    bean with an unknown primary key class or a primary key field
    (in both cases there is no user defined primary key class) and
    the requested method is equals or hashCode, a dummy method will
    also be returned. Types are specified as type names for
    primitive type such as int, float or as fully qualified class
    names. Note, the method does not return inherited methods.
    @param className the name of the class which contains the
    method to be checked @param methodName the name of the method
    to be checked
    @param argTypeNames the fully qualified names of the argument
        types
    @return the method element
    @see getClass
    */

   public Object getMethod (final String className, final String
       methodName,
      String[] argTypeNames)
   {
      int keyClassType = getPersistenceKeyClassType(className);
      Object returnObject = null;

      if (isPCClassName(className))
      {
         if ((methodName.equals("readObject") &&  // NOI18N
                   Arrays.equals(argTypeNames, getReadObjectArgs()))
                       ||
            (methodName.equals("writeObject") &&  // NOI18N
                      Arrays.equals(argTypeNames,
                          getWriteObjectArgs())))
         {
            returnObject = new MemberWrapper(methodName,
               Void.TYPE, Modifier.PRIVATE,
                   (Class)getClass(className));
         }
      }
      if ((NameMapper.UNKNOWN_KEY_CLASS == keyClassType) ||
         (NameMapper.PRIMARY_KEY_FIELD == keyClassType))
      {
         if (methodName.equals("equals") &&  // NOI18N
                   Arrays.equals(argTypeNames, getEqualsArgs()))
```

```
        {
            returnObject = new MemberWrapper(methodName,
                Boolean.TYPE, Modifier.PUBLIC,
                    (Class)getClass(className));
        }
        else if (methodName.equals("hashCode") &&  // NOI18N
                  Arrays.equals(argTypeNames, NO_ARGS))
        {
            returnObject = new MemberWrapper(methodName,
                Integer.TYPE, Modifier.PUBLIC,
                    (Class)getClass(className));
        }
    }

    if (returnObject == null)
    {
        returnObject = super.getMethod(className, methodName,
            argTypeNames);

        if (returnObject instanceof Method)  // wrap it
            returnObject = new MemberWrapper((Method)returnObject);
    }

    return returnObject;
}
```

2.3 Third Method: getFields

```
/** Returns a list of names of all the declared field elements in
    the class with the specified name. If the specified className
    represents a persistence-capable class, the list of field names
    from the corresponding ejb is returned (even if there is a
    Class object available for the persistence-capable).
    @param className the fully qualified name of the class to be
        checked
    @return the names of the field elements for the specified class
    */
   public List getFields (final String className)
   {
      final EjbCMPEntityDescriptor descriptor =
          getCMPDescriptor(className);
      String testClass = className;

      if (descriptor != null)  // need to get names of ejb fields
      {
         Iterator iterator =
             descriptor.getFieldDescriptors().iterator();
         List returnList = new ArrayList();

         while (iterator.hasNext())
            returnList.add(((FieldDescriptor)iterator.next()).getName());

         return returnList;
      }
      else
      {
         NameMapper nameMapper = getNameMapper();
         String ejbName =
            nameMapper.getEjbNameForPersistenceKeyClass(className);

         switch (getPersistenceKeyClassType(className))
         {
            // find the field names we need in the corresponding
            // ejb key class
            case NameMapper.USER_DEFINED_KEY_CLASS:
               testClass = nameMapper.getKeyClassForEjbName(ejbName);
               break;
            // find the field name we need in the abstract bean
            case NameMapper.PRIMARY_KEY_FIELD:
               return Arrays.asList(new String[]{
                  getCMPDescriptor(ejbName).
                  getPrimaryKeyFieldDesc().getName()});
            // find the field name we need in the persistence capable
            case NameMapper.UNKNOWN_KEY_CLASS:
               String pcClassName =
```

```
                nameMapper.getPersistenceClassForEjbName(ejbName);
            PersistenceFieldElement[] fields =
                getPersistenceClass(pcClassName).getFields();
            int i, count = ((fields != null) ? fields.length : 0);

            for (i = 0; i < count; i++)
            {
                PersistenceFieldElement pfe = fields[i];

                if (pfe.isKey())
                    return Arrays.asList(new
                        String[]{pfe.getName()});
            }
            break;
        }
    }

    return super.getFields(testClass);
}
```

2.4 Fourth Method: getField

```
/** Returns a wrapped field element for the specified fieldName in
    the class with the specified className. If the specified
    className represents a persistence-capable class, a field
    representing the field in the abstract bean class for the
    corresponding ejb is always returned (even if there is a Field
    object available for the persistence-capable). If there is an
    ejb name and an abstract bean class with the same name, the
    abstract bean class which is associated with the ejb will be
    used, not the abstract bean class which corresponds to the
    supplied name (directly).
    @param className the fully qualified name of the class which
        contains the field to be checked
    @param fieldName the name of the field to be checked
    @return the wrapped field element for the specified fieldName
    */
public Object getField (final String className, String fieldName)
   {
      String testClass = className;
      Object returnObject = null;

      if (className != null)
      {
         NameMapper nameMapper = getNameMapper();
         boolean isPCClass = isPCClassName(className);
         boolean isPKClassName = false;
         String searchClassName = className;
         String searchFieldName = fieldName;

         // translate the class name & field names to corresponding
         // ejb name is abstract bean equivalents if necessary
         if (isPCClass)
         {
            searchFieldName = nameMapper.
               getEjbFieldForPersistenceField(className, fieldName);
            searchClassName = getEjbName(className);
         }
         else // check if it is a pk class without a user defined
             key class
         {
            String ejbName =
               nameMapper.getEjbNameForPersistenceKeyClass(className);

            switch (getPersistenceKeyClassType(className))
            {
               // find the field we need in the corresponding
               // abstract bean (translated below from ejbName)
               case NameMapper.PRIMARY_KEY_FIELD:
```

```
            testClass = ejbName;
            searchClassName = ejbName;
            isPKClassName = true;
            break;
         // find the field we need by called updateFieldWrapper
         // below which handles the generated field for the
         // unknown key class - need to use the
         // persistence-capable class name and flag to call that
         // code, so we configure it here
         case NameMapper.UNKNOWN_KEY_CLASS:
            testClass = nameMapper.
               getPersistenceClassForEjbName(ejbName);
            isPCClass = true;
            isPKClassName = true;
            break;
      }
   }

   if (nameMapper.isEjbName(searchClassName))
   {
      searchClassName = nameMapper.
         getAbstractBeanClassForEjbName(searchClassName);
   }

   returnObject = super.getField(searchClassName,
       searchFieldName);

   if (returnObject == null) // try getting it from the
       descriptor
      returnObject = getFieldWrapper(testClass,
          searchFieldName);
   else if (returnObject instanceof Field)  // wrap it
      returnObject = new MemberWrapper((Field)returnObject);

   if (isPCClass)
   {
      returnObject = updateFieldWrapper(
         (MemberWrapper)returnObject, testClass, fieldName);
   }
   // when asking for these fields as part of the
   // persistence-capable is key class, we need to represent
       the
   // public modifier which will be generated in the inner
       class
   if (isPKClassName && (returnObject instanceof
       MemberWrapper))
      ((MemberWrapper)returnObject)._modifiers =
          Modifier.PUBLIC;
```

```
    }

    return returnObject;
}
```

2.5 Fifth Method: getFieldType

```
/** Returns the field type for the specified fieldName in the class
    with the specified className. This method is overrides the one
    in Model in order to do special handling for non-collection
    relationship fields. If it's a generated relationship that
    case, the returned MemberWrapper from getField contains a type
    of the abstract bean and it's impossible to convert that into
    the persistence capable class name, so here that case is
    detected, and if found, the ejb name is extracted and used to
    find the corresponding persistence capable class. For a
    relationship which is of type of the local interface, we do the
    conversion from local interface to persistence-capable class.
    In the case of a collection relationship (generated or not),
    the superclass' implementation which provides the java type is
    sufficient.
    @param className the fully qualified name of the class which
        contains the field to be checked
    @param fieldName the name of the field to be checked
    @return the field type for the specified fieldName
   */

     public String getFieldType (String className, String fieldName)
   {
     String returnType = super.getFieldType(className, fieldName);

     if (!isCollection(returnType) && isPCClassName(className))
     {
        NameMapper nameMapper = getNameMapper();
        String ejbName =
           nameMapper.getEjbNameForPersistenceClass(className);
        String ejbField =
           nameMapper.getEjbFieldForPersistenceField(className,
               fieldName);

        if (nameMapper.isGeneratedEjbRelationship(ejbName,
            ejbField))
        {
           String[] inverse =
              nameMapper.getEjbFieldForGeneratedField(ejbName,
                  ejbField);

           returnType = nameMapper.
              getPersistenceClassForEjbName(inverse[0]);
        }

        if (nameMapper.isLocalInterface(returnType))
        {
```

```
            returnType =
                nameMapper.getPersistenceClassForLocalInterface(
                classNameName, fieldName, returnType);
        }
    }

    return returnType;
}
```

2.6 Sixth Method: getFieldWrapper

```
private MemberWrapper getFieldWrapper (String className, String
    fieldName)
   {
      EjbCMPEntityDescriptor descriptor =
          getCMPDescriptor(className);
      MemberWrapper returnObject = null;

      if (descriptor != null)
      {
         PersistenceDescriptor persistenceDescriptor =
           descriptor.getPersistenceDescriptor();

         if (persistenceDescriptor != null)
         {
            Class fieldType = null;

            try
            {
               fieldType =
                   persistenceDescriptor.getTypeFor(fieldName);
            }
            catch (RuntimeException e)
            {
               // fieldType will be null - there is no such field
            }

            returnObject = ((fieldType == null) ? null :
               new MemberWrapper(fieldName, fieldType,
               Modifier.PRIVATE, (Class)getClass(className)));
         }
      }

      return returnObject;
   }
```

3 Functional Roles

3.1 DeploymentDescriptorModel class

The main function of this class is to augment the java metadata for a non-existent persistence-capable java/class file using the deployment descriptor. It is primarily used at ejbc time, though it could be used at any time as long as sufficient mapping and deployment descriptor information is available.

3.2 MemberWrapper class

This class is a utility class used to wrap an element into an object.

3.3 getConstructor method

The role of this method is to provide a wrapped constructor element for the class identified by the className parameter, this works only if the class isn't a persistence-capable key class that corresponds to a bean with unknown primary key, in this case a dummy constructor is returned. For a more detailedm description and implementation see the javadoc related to this method.

3.4 getMethod method

This method, similarly to the previous one, is used to get a wrapped constructor element for the class identified by the className parameter. Also in this case if the class is a persistence capable key class which corresponds to a bean with unknown primary key which or a primary key field, and the method identified is a equals or hashCode name a dummy method is returned. A dummy method is returned also in the case where the class is persistence capable and the method is a read or writed object method. This method will never returns inherited methods, for this purpose is used the getInheritedMethod method.

3.5 getFields method

The role of this method is to return a list of names of all the declared field elements in the class identified by the className parameter. If the class represents a persistence-capable class, the list of field names from the corresponding ejb is returned.

3.6 getField method

The role of this method is similar to the previous one but instead this one return only one wrapped field element identified by the fieldName parameter.

3.7 getFieldType method

This method works as the previous one but instead this returns the field type (as a string) of a field element identified by the fieldName parameter in the specified class.

3.8 getFieldWrapper method

The role of this method is to provide a functionality to get a field wrapper given a field's name and type.

4 Issues found

In this section we will list all the issues found by applying the checklist provided to the methods and classes we were assigned.

4.1 DeploymentDescriptorModel class issues

- In several lines of the class provided the indentation is done by tab and not by space, although it is equal to 4 spaces in every line.
- The bracing style chosen is Allman style and is consistent in every line except lines 111, 112, 918 to 922 and 927, where fact curly braces and statement between them is written in one line only.
- No line exceed the 80 characters limit except line 95 and line 276, although they not exceed the 120 characters limit.
- The DeploymentDescriptorModel.java file contains the public class DeploymentDescriptorModel that is the first class declared, and the private class MemberWrapper.
- MemberWrapper class is not documented properly.
- The javadoc is missing for the following methods: getEjbName, getPersistenceKeyClassType, getFieldWrapper, updateFieldWrapper.
- Variables are declared in the wrong order, first are declared non-static instance variables and then static class variables.
- Methods are grouped properly by functionalities.
- getField and updateFielWrapper methods are too long, they should split in two or more auxiliary methods.

4.2 getConstructor method issues

- Line 285: if statement with only one statement to execute without curly braces.
- Lines are wrapped properly
- The method is documented properly
- There isn't commented out code.
- All declarations and initializations are done in the correct manner.
- All methods are called correctly.
- Method doesn't make use of arrays.

- Objects are compared correctly.
- There are no computations. All assignments are done correctly and there are no implicit casts.
- There are switch statements or loop.
- There is no use of files.

4.3 getMethod method issues

- Line 349: if statement with only one statement to execute without curly braces.
- Lines are wrapped properly
- The method is documented properly
- There isn't commented out code.
- All declarations and initializations are done in the correct manner.
- All methods are called correctly.
- Method doesn't make use of arrays.
- Objects are compared correctly.
- There are no computations. All assignments are done correctly and there are no implicit casts.
- There are switch statements or loop.
- There is no use of files.

4.4 getFields method issues

- Line 403: while loop with only one statement to execute without curly braces.
- Line 435: if statement with only one statement to execute without curly braces.
- Lines are wrapped properly
- The method is documented properly
- There isn't commented out code.
- All declarations and initializations are done in the correct manner.

- All methods are called correctly.
- Arrays are implemented correctly, there shouldn't be any indexes error since an iterator is used.
- Objects are compared correctly.
- There are no computations. All assignments are done correctly and there are no implicit casts.
- All cases are addressed with break statement.
- There is no use of files.

4.5 getField method issues

- This method is too long, it should be split in two or more auxiliary methods.
- Line 539-541: if-else statement with only one statement to execute without curly braces.
- Line 552, if statement with only one statement to execute without curly braces.
- Lines are wrapped properly
- The method is documented properly
- There isn't commented out code.
- All declarations and initializations are done in the correct manner.
- All methods are called correctly.
- Objects are compared correctly.
- There are no computations. All assignments are done correctly and there are no implicit casts.
- All cases are addressed with break statement.
- There is no use of files.

4.6 getFieldType method issues

- Lines are wrapped properly
- The method is documented properly
- There isn't commented out code.
- All declarations and initializations are done in the correct manner.
- All methods are called correctly.
- Objects are compared correctly.
- There are no computations. All assignments are done correctly and there are no implicit casts.
- There are no switch case statements
- There is no use of files.

4.7 getFieldWrapper method issues

- Line 539-541: if-else statement with only one statement to execute without curly braces.
- Line 552, if statement with only one statement to execute without curly braces.
- Lines are wrapped properly
- The method is documented properly
- There isn't commented out code.
- All declarations and initializations are done in the correct manner.
- All methods are called correctly.
- Objects are compared correctly.
- There are no computations. All assignments are done correctly and there are no implicit casts.
- Line 798 to 801,
- All cases are addressed with break statement.