Document Refactorings

Soumya Mudiyappa
West Chester University
fs926226@wcupa.edu

Swetchha Shukla
West Chester University
ss928947@wcupa.edu

Yung-Chen Cheng
West Chester University
yc917559@wcupa.edu

I. Rename Refactorings

Rename Refactoring provides an easy way to change the name of identifiers for code symbols without changing its behavior. There are five types of Rename Refactoring:

1) Rename Class Declarations
2) Rename Method Declarations
3) Rename Field Declarations
4) Rename Local Variables
5) Rename Package Declarations

II. Preconditions of Rename Class Refactoring

Rename Class Refactoring (RcR) changes the name of the class and all references to that class to the new name without changing its behavior. There are certain preconditions required for RcR.

1) The target class cannot be duplicate with any existing class within same package after rename.
2) The target class cannot be duplicate with any imported class from different package after rename.
3) If a parent class imports a class from different package, the target child class within same java file cannot be duplicate with that imported class after rename.

A. The target class cannot be duplicate with an existing class within same package after rename.

When we try to rename a class with an existing class name, the Eclipse produces syntax error: "Please choose another name". [1] The classes will be conflicted if we rename the target class using the name of an existing class in the same package. So we can not have duplicate class names in the same package.

For example, a package *p* contains class: *A*, *B* and *C*, now we want to refactor the class name *A* to *B*:

```
package p;
class A{}
class B{}
class C{}
```

(a) Before Rename Refactoring Class A

```
package p;
class B{}
class B{}
class C{}
```

(b) After Rename Refactoring Class A

Fig. 1. Example of Rename Class Refactoring

Then the java compiler shows up the error that B.java already exists as figure 2:

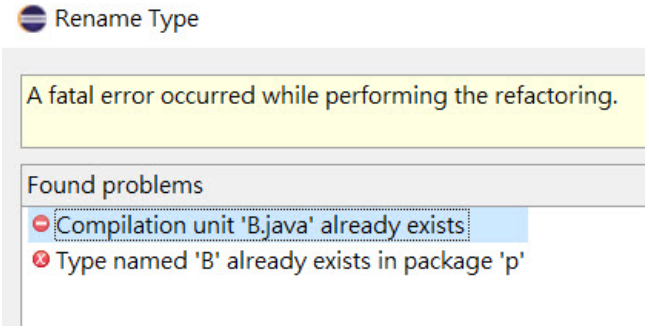

Fig. 2. The error of using same class name for refactoring

Furthermore, this precondition is applicable to nested classes. The examples below show that we can not use the same name either as inner or as outer class for nested classes:

- Example 1: The rename refacrtoring of the inner class can not be the same name as other inner classes' name.

```
package p;

public class A{
    class M{}
    class N{}
}
```

(a) Before Rename Refactoring Inner Class M

```
package p;

public class A{
    class N{}
    class N{}
}
```

(b) After Rename Refactoring Inner Class M to N

Fig. 3. Example 1 of Nested Class Rename Refactoring

Then the java compiler shows up the error as figure 4:

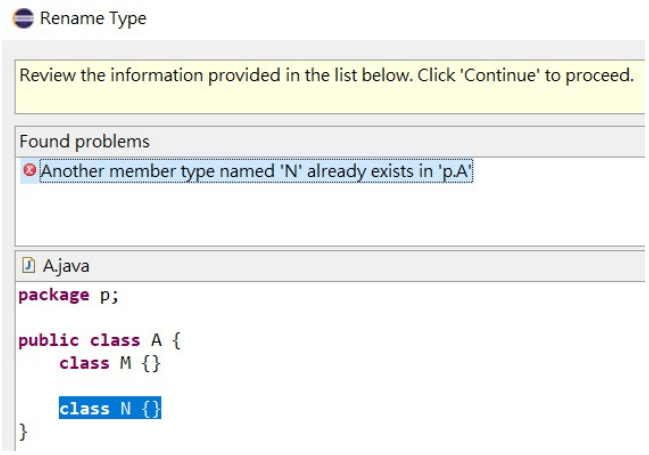

Fig. 4. The error of using same inner class name for refactoring

- Example 2: The rename refacrtoring of the outer class can not be the same name as the inner classes' name.

```
package p;

public class A{
    class M{}
    class N{}
}
```

(a) Before Rename Refactoring Outer Class A

```
package p;

public class M{
    class M{}
    class N{}
}
```

(b) After Rename Refactoring Outer Class A to M

Fig. 5. Example 2 of Nested Class Rename Refactoring

Then the java compiler shows up the error as figure 6:

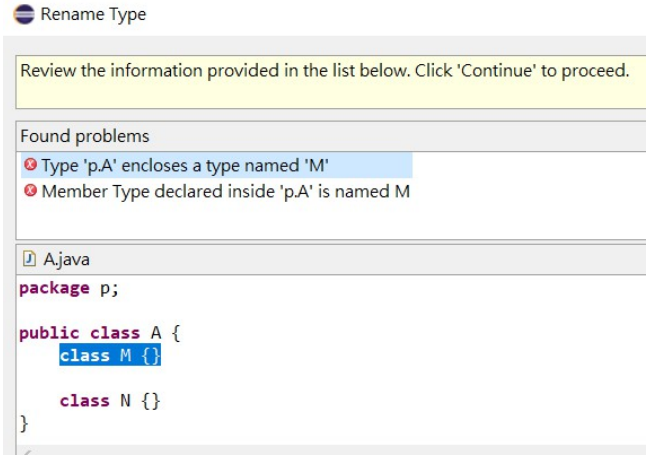

Fig. 6. The error of using same inner class name for refactoring

- Example 3: The rename refacrtoring of the inner class can not be the same name as the outer class name.

```
package p;

public class A{
    class M{}
    class N{}
}
```

(a) Before Rename Refactoring Inner Class M

```
package p;

public class A{
    class A{}
    class N{}
}
```

(b) After Rename Refactoring Inner Class M to A

Fig. 7. Example 3 of Nested Class Rename Refactoring

Then the java compiler shows up the error as figure 8:

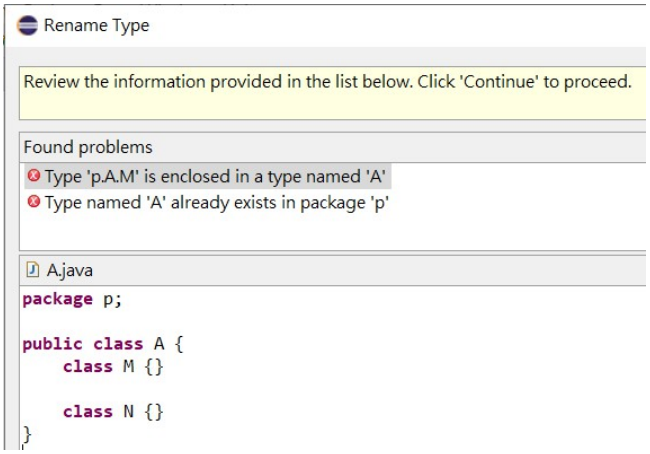

Fig. 8. The error of using same outer class name for refactoring

Also, this precondition is applicable even if one or the other file is empty. So checking whether a class with the same name already exists in a package should be the first job we have to do for RcR.

B. The target class cannot be duplicate with any imported class from different package after rename.

If a class is imported from different package, we have to pre-check that the new name of the target class is not duplicate with the imported class after rename refactoring.

```
package q;
import p.C;

class A{
}

class B{
}
```

(a) Before

```
package q;
import p.C;

class A{
}

class C{
}
```

(b) After

Fig. 9. **RcR from B to C**

In Fig. 9 (a), we see that class B is not duplicate with class A and we can implement RcR on class B to any other name except 'A' as mentioned in section II-A. However, in Fig. 9 (b), when we try to implement RcR from *B to C*, java generates compile error *"a compilation unit must not import and declare a type with the same name"* [1] as shown in Fig. 10. This is because the compiler cannot distinguish between the *imported class C* of package 'p' and the *existing class C* of package 'q'.

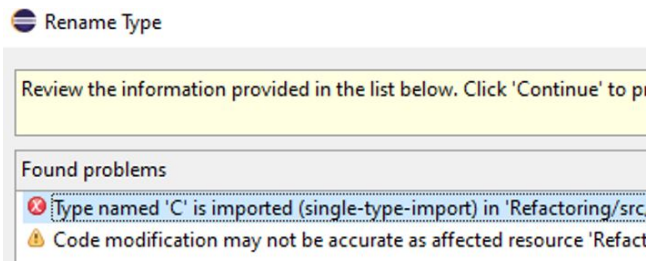

Fig. 10. **Compilation Unit Error**

Therefore, it is essential to pre-check that the target class should not have duplicate name with any of imported class after RcR.

C. If a parent class imports a class from different package, the target child class within same java file cannot be duplicate with that imported class after rename.

If a parent class imports a class from different package and if we try renaming a child class with the same name as the class imported into its parent class within the same java file, compiler produces the error as 'a compilation unit must not import and declare a type with the same name' [1].This precondition can be explained by the following example.

A.java

```
package q;
import p.C;
class A{
}
class B extends A{
}
class D extends B{
}
```

(a) Before renaming Subclass B

A.java

```
package q;
import p.C;
class A{
}
class C extends A{
}
class D extends B{
}
```

(b) After Renaming Child class B to C

Fig. 11. Precondition for Renaming Child class within the same java file

From the above figure 11, We see that if a parent class A imports a class C from package p and if we try to rename the child class B to C or D to C, java generates compile error as given in the figure 12. As mentioned in section II-B, the same precondition also holds good for renaming a sub-class. This precondition is applicable for all ancestor class and we have to trace back and check if any of the parent class is importing a class with same name before renaming the child class within the same java file.

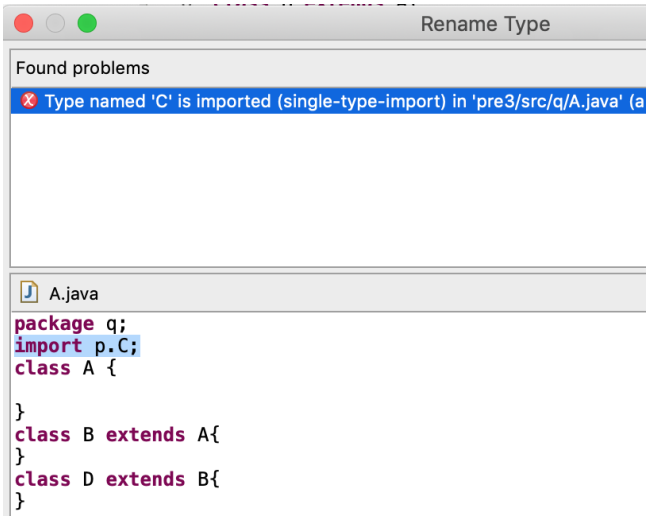

Fig. 12. Error produced after renaming the sub-class.

If a parent class imports a class from different package and if the child class is defined in a separate java file, then in that case we can refactor and rename the child class to the imported class name.

III. Code Change Rules

References

[1] "Eclipse Refactorings Properties," https://git.eclipse.org.