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 any 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:

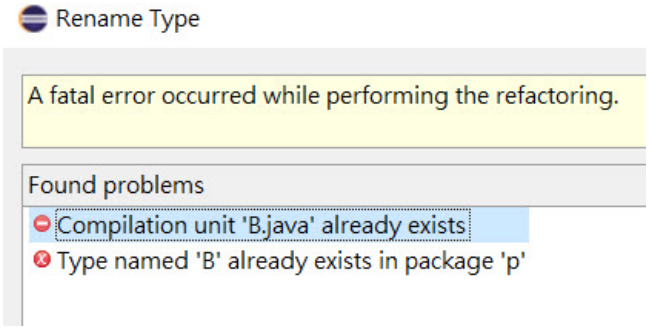


Fig. 2. The error of using same class name for refactoring

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 rename refactoring.

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 does not match or duplicate with the imported class after rename refactoring.

```
package q;
import p.C;

class A{
}

class B{
}
```

(a) Before Rename Refactoring Class B

```
package q;
import p.C;

class A{
}

class C{
}
```

(b) After Rename Refactoring Class B to C

Fig. 3. **Precondition when importing a class**

In Fig. 3 (a), we see that class B is not duplicate with class A and class B can be rename refactored to any other name except A as mentioned in section II-A. However, in Fig. 3 (b), when we try to rename refactor the class *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. 4. This is because the compiler cannot distinguish between the *imported class C* of package 'p' and the *existing class C* of package 'q'. Therefore, a conflict in the duplicate names arises as shown in Fig. 5

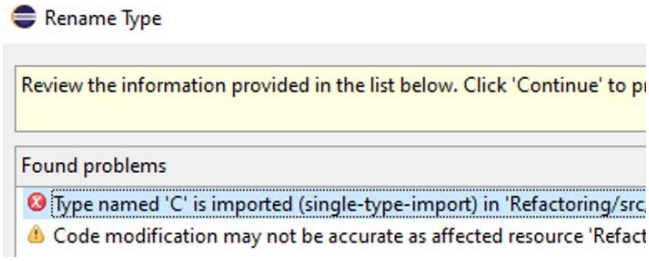


Fig. 4. **Compilation Unit Error**

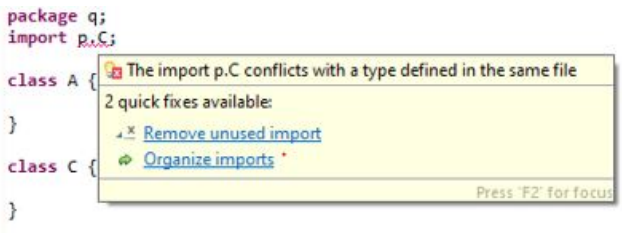
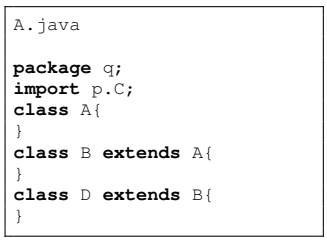


Fig. 5. **Duplicate name conflict**

Hence, 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.

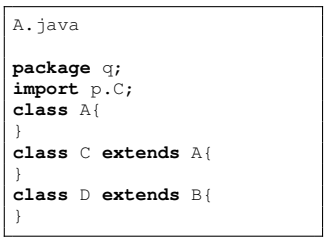
If a parent class imports a class from different package and if we try renaming a child class with the same name as the class imported into its parent class within the same java file, compiler produces the error as 'a compilation unit must not import and declare a type with the same name' [1].This precondition 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. 6. Precondition for Renaming Child class within the same java file

From the above figure 6, We see that if one has imported class C from package p to the parent Class A and if we try to rename the child class B to C or D to C, Compiler produces the error as given in the figure 7. 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, so one has 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.

If a parent class imports a class from different package and if the child class is defined in a separate java file. At that time

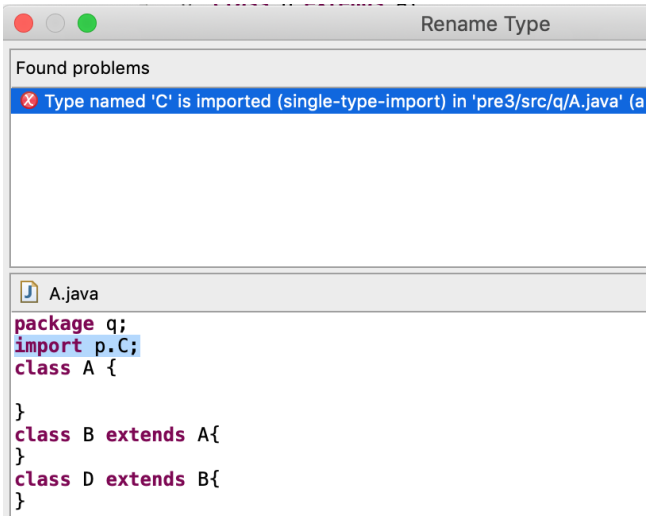
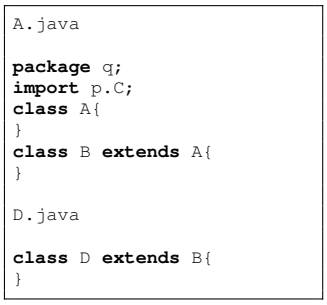


Fig. 7. Error produced after renaming the sub-class.

we can refactor and rename the child class to the imported class name. In the below figure 8, We see that child class of B (class D)is defined in separate java file and this time we can easily refactor and rename the child class D to C.



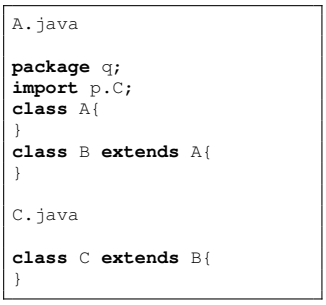
```
A.java

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

D.java

class D extends B{
}
```

(a) Before renaming Child class D



```
A.java

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

C.java

class C extends B{
}
```

(b) After Renaming Child Class D to C

Fig. 8. Precondition for Renaming Child Class in separate java file

III. Code Change Rules

References

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