

Document Refactorings

Soumya Mudiappa
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 any existing class name, the Java compiler produces syntax error “Please choose another name”. [1] The classes will be conflicted if we rename refactor 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, if we want to rename refactor the class name from A to B as in Fig. 1, then the Java compiler produces an error that B.java already exists as shown in Fig. 2.

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

(a) Before

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

(b) After

Fig. 1. Example of RcR from A to B

Rename Type

A fatal error occurred while performing the refactoring.

Found problems

- Compilation unit 'B.java' already exists
- Type named 'B' already exists in package 'p'

Fig. 2. Compile error for using same class name after RcR

Also, this precondition is applicable even if the classes are defined in separate java files or any file is empty but are within the same folder. In Fig. 3, we try to apply RcR from A to C or B to C where C.java is an empty file, the compiler produces an error as shown in Fig. 4.

```
A.java  
  
public class A{  
}  
  
class B{  
}  
  
C.java  
//empty file
```

(a) Before

```
A.java  
  
public class A{  
}  
  
class C{  
}  
  
C.java  
//empty file
```

(b) After

Fig. 3. RcR from B to C even if C.java is empty

Rename Type

Found problems

- Compilation unit 'C.java' already exists

Fig. 4. Compile error for same file name after RcR

Furthermore, this precondition is also applicable to nested classes. The examples below show that for RcR we can not

use the same name either as inner or as outer class for nested classes like Fig. 5.

```
package p;

public class A{

    class M{
    }

    class N{
    }
}
```

Fig. 5. Nested Class before RcR

Example 1: In order to implement RcR for inner class, we should pre-check that we do not use the same name as any of other inner class name. From Fig. 6, when we try to rename refactor the inner class from M to N, the Java compiler produces an error as shown in Fig. 7.

```
package p;

public class A{

    class N{
    }

    class N{
    }
}
```

Fig. 6. Example 1 for Nested Class after RcR from M to N

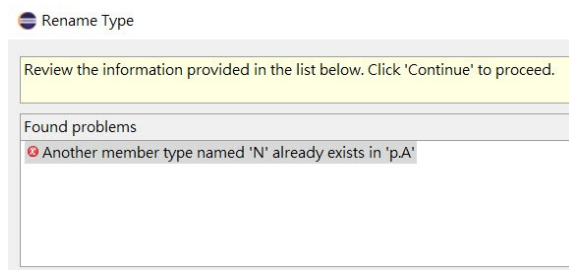


Fig. 7. Compile error for duplicate inner class name after RcR

Example 2: In order to implement RcR for outer class, we should pre-check that we do not use the same name as any of the inner class name and vice-versa. From Fig. 8, when we try to use same name for outer class and inner class, the Java compiler produces an error as shown in Fig. 9 and Fig. 10.

```
package p;

public class M{

    class M{
    }

    class N{
    }
}
```

```
package p;

public class A{

    class A{
    }

    class N{
    }
}
```

(a) After RcR on Outer Class A to M

(b) After RcR on Inner Class N to A

Fig. 8. Example 2 of Nested Class after RcR

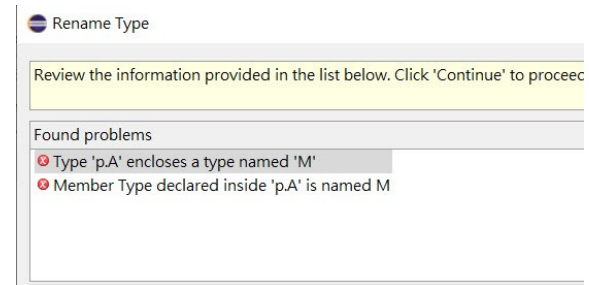


Fig. 9. Compile error for RcR outer class same as inner class name

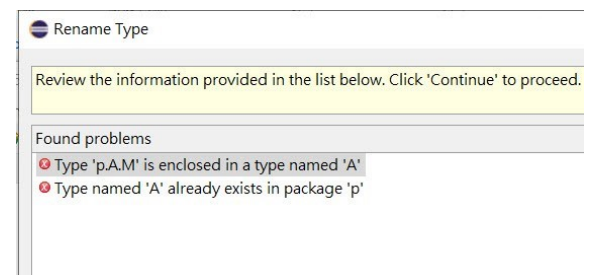


Fig. 10. Compile error for RcR inner class same as outer class name

Therefore, checking whether a class with the same name already exists in a package should be the first precondition 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{
}
```

```
package q;
import p.C;

class A{
}

class C{
}
```

(a) Before

(b) After

Fig. 11. RcR from B to C

In Fig. 11 (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. 11 (b), when we try to implement RcR from B to C, Java compiler produces an error “*a compilation unit must not import and declare a type with the same name*” [1] as shown in Fig. 12. This is because the compiler cannot distinguish between the *imported class C* of package ‘p’ and the *existing class C* of package ‘q’.

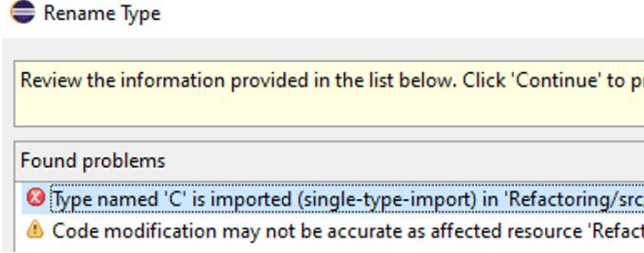


Fig. 12. Compilation Unit Error

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 an 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.

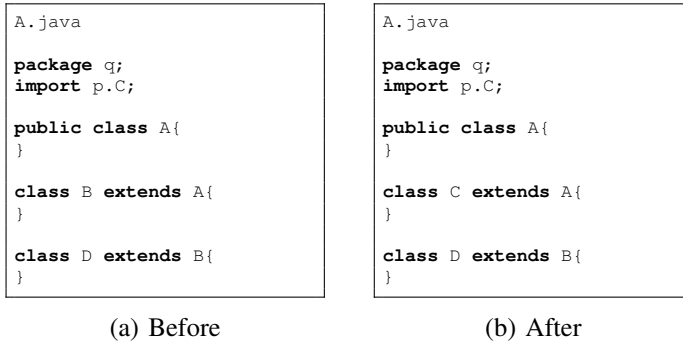


Fig. 13. RcR from B to C

From the above Fig. 13, we see that if a parent class A imports a class C from package ‘p’ and if we try to rename the child class from B to C or D to C, Java compiler produces an error as shown in the Fig. 14. As mentioned in section II-B, the same precondition also holds good for renaming a child 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.

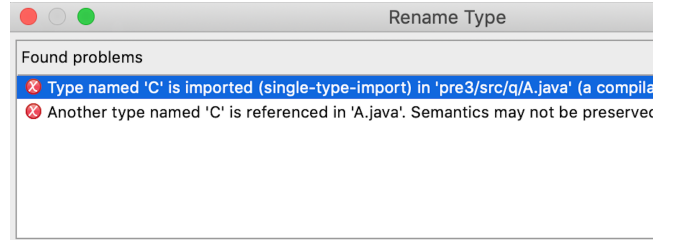


Fig. 14. Compile error for RcR child class to imported class

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/>.