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) The target class file name cannot be duplicate with any existing java file name within same package after rename.

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

When we rename a class with any existing class name, the Java compiler produces error message: *"duplicate class: p.B"*. The classes will be conflicted if we run rename refactor on 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 run rename refactor on 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.

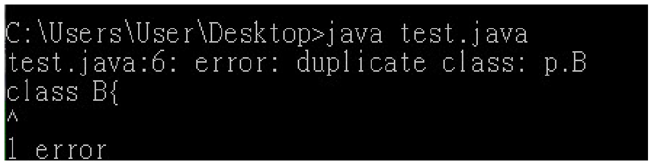
```
package p;

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**



```
C:\Users\User\Desktop>java test.java
test.java:6: error: duplicate class: p.B
class B{
^
1 error
```

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

Furthermore, the same situation occurs in 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. 3.

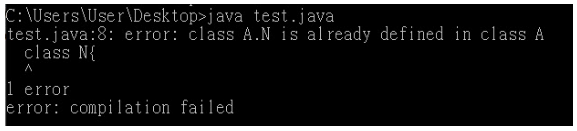
```
package p;

public class A{

  class M{
  }

  class N{
  }
}
```

Fig. 3. **Nested Class before RcR**

Example 1: In order to apply RcR for inner class, we should pre-check that we do not use the same name as any of other inner class name. As shown in Fig. 4, when we run RcR on the inner class from M to N, the Java compiler produces an error as shown in Fig. 5.

```
package p;

public class A{

  class N{
  }

  class N{
  }
}
```

Fig. 4. **Example 1 for Nested Class after RcR from M to N**



```
C:\Users\User\Desktop>java test.java
test.java:8: error: class A.N is already defined in class A
  class N{
  ^
1 error
error: compilation failed
```

Fig. 5. **Compile error for duplicate inner class name after RcR**

Example 2: In order to apply 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. As shown in Fig. 6, when we use same name for outer class and inner class, the Java compiler produces an error as shown in Fig. 7 and Fig. 8.

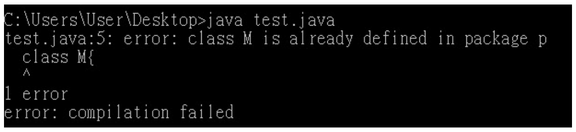
```
package p;

public class M{

  class M{
  }

  class N{
  }
}
```

(a) After RcR on Outer Class A to M

```
package p;

public class A{

  class M{
  }

  class A{
  }
}
```

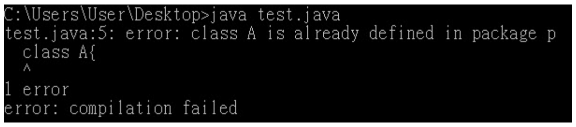
(b) After RcR on Inner Class N to A

Fig. 6. **Example 2 of Nested Class after RcR**



```
C:\Users\User\Desktop>java test.java
test.java:5: error: class M is already defined in package p
  class M{
  ^
1 error
error: compilation failed
```

Fig. 7. **Compile error for RcR outer class same as inner class name**



```
C:\Users\User\Desktop>java test.java
test.java:5: error: class A is already defined in package p
  class A{
  ^
1 error
error: compilation failed
```

Fig. 8. **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.

In Fig. 9(a), we see that class B is not duplicate with class A and we can apply RcR on class B to any other name except 'A' as mentioned in section II-A. However, in Fig. 9(b), when we apply RcR from *B to C*, Java compiler produces an error *"C is already defined in this compilation unit"*. This is because the compiler cannot distinguish between the *imported class C* of package 'p' and the *existing class C* of package 'q'.

This precondition also holds good for renaming a child class. Suppose a Java file has a class and its children classes as shown in Fig. 10. We see that if parent class A imports a class C from package 'p' and if we apply RcR on the child class from B to C or D to C, Java compiler produces an error *"C is already defined in this compilation unit"*. This precondition is applicable for all ancestor class and we have to trace back

```
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**

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.

```
//A.java

package q;
import p.C;

public class A{
}

class B extends A{
}

class D extends B{
}
```

(a) Before

```
//A.java

package q;
import p.C;

public class A{
}

class C extends A{
}

class D extends B{
}
```

(b) After

Fig. 10. **RcR for child class B to C**

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 apply RcR on child class to the imported class name.

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

C. The target class file name cannot be duplicate with any existing java file name within same package after rename.

If classes are defined in separate java files, we have to pre-check that the new name of the target class declared in its class file name is not duplicate with any existing Java file name within the same package.

```
//A.java

public class A{
}

class B{
}


//C.java
//empty file
```

(a) Before

```
//C.java

public class C{
}

class B{
}


//C.java
//empty file
```

(b) After

Fig. 11. **RcR from A to C if A is declared in file A.java**

For example, from Fig. 11, when we apply RcR from *A to C* where C.java is an empty file, the Java compiler produces an error *"Compilation unit 'C.java' already exists"*. This is because after RcR the Java file also gets renamed from *A.java to C.java* and therefore creates a conflict for duplicate file name as *C.java* already exists in the same package.

However, we can apply RcR from *B to C* as class B is not declared within its class file name.

References