The “Extract method” JAVA refactoring

Program Transformation Project

Bogdan Dumitriu, 0402044
Jose Magalhaes, F031331

April 2005

Project description

Extract Method is a program refactoring which extracts a piece of code to a new method:

```
void printOwing() {
  printBanner();

  //print details
  System.out.println("name:   "
             + _name);
  System.out.println("amount "
           + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails (double
outstanding) {
  System.out.println ("name: "
                    + _name);
  System.out.println ("amount "
                    + outstanding);
}
```

Extract Method - why?

The *Extract Method* refactoring has two main advantages:

- Increases readability – methods become shorter and meaningful names help understanding the code

- Reusability made easier – the extracted method may be useful somewhere else

Restrictions

Extract Method cannot be done if the fragment:

- contains any control flow instruction (break, continue, labels or returns);
- contains assignments to more than one variable declared outside the fragment.

Extraction – Case analysis (1)

Fragment type: does not use any variable
Solution: simply move code and call the new method

```
public class TestCase02 {
  public void testMethod() {
     System.out.print("header");

     @ emTestMethod
     System.out.print("some text");
     System.out.print("some more text");
     @

     System.out.print("footer");
  }
}
```

```
public class TestCase02 {
  public void testMethod() {
    System.out.print("header");
    emTestMethod();
    System.out.print("footer");
  }

  private void emTestMethod() {
    System.out.print("some text");
    System.out.print("some more text");
  }
}
```

Extraction – Case analysis (2)

Fragment type: uses variables from the parent method, but doesn't change them

Solution: pass them as parameters

```
public class TestCase03 {
  public void testMethod() {
     int a, b, c;

     @ emTestMethod
     int i;

     for (i = 0; i < 5; i++) {
       System.out.print(a + b + c + i);
     }
     @
  }
}
```

```
public class TestCase03 {
  public void testMethod() {
    int a, b, c;
    emTestMethod(a, b, c);
  }

  private void emTestMethod(int a, int b, int c) {
    int i;
    for(i = 0; i < 5; i++) {
      System.out.print(a + b + c + i);
    }
  }
}
```

Extraction – Case analysis (3)

Fragment type: changes one variable
Solution: return it (in the new method) and assign to it (in the parent method)

```
public class TestCase05 {
  public void testMethod() {
     System.out.print("header");
     ArrayList x;

     @ emTestMethod
     x = new ArrayList();
     @

     System.out.print(x);
     System.out.print("footer");
  }
}
```

```
public class TestCase05 {
  public void testMethod() {
    System.out.print("header");
    ArrayList x;

    x = emTestMethod(x); // assigns the return
              // value to the changed variable

    System.out.print(x);
    System.out.print("footer");
  }

  private ArrayList emTestMethod(ArrayList x) {
    x = new ArrayList();
    return x; // changed parameter is returned
  }
}
```

Extraction – Case analysis (4)

Fragment type: has variable declarations which are used after the fragment

Solution: move those declarations up (carefully)

```
public class TestCase13 {
  public void testMethod(int a, boolean
b, ArrayList al) {
      int x = 2;

      @ emTestMethod
      x = 1;
      String aa, bb = "r", z = "text",
            cc = "t";

      System.out.print(z);
      System.out.print(a);
      System.out.print(al);
      @

      z = “”;
   }
}
```

```
public class TestCase13 {
  public void testMethod(int a, boolean b,
ArrayList al) {
    int x = 2;
    String z; // moved up
    x = emTestMethod(x, a, al, z);
    z = “”; // is still valid
  }

  private int emTestMethod(int x, int a, ArrayList
al, String z) {
    x = 1;
    String aa;
    String bb = "r";
    z = "text"; // transformed into assign
    String cc = "t";
    System.out.print(z);
    System.out.print(a);
    System.out.print(al);

  }
}
```

Extraction – Case analysis (5)

Fragment type: throws non-runtime exceptions
Solution: add the *throws* clause – check if the fragment doesn't already catch some exceptions

```
public class TestCase15 {
  public void testMethod(String name)
                throws IOException {
    File f = new File(name);
    FileInputStream fis = new
                  FileInputStream(f);
    int c;
    @ emTestMethod
    while ((c = fis.read()) != -1) {
         System.out.write(c);
    }
    @
    fis.close();
  }
}
```



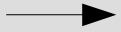
```
public class TestCase15 {
  public void testMethod(String name) throws
                                  IOException {
    File f = new java.io.File(name);
    FileInputStream fis = new FileInputStream(f);
    int c;
    c = emTestMethod(c, fis);
    fis.close();
  }

  private int emTestMethod(int c,
    FileInputStream fis) throws IOException {
    while((c = fis.read()) != -1) {
      System.out.write(c);
    }
    return c;
  }
}
```

Extraction – Case analysis (6)

```
public class TestCase16 {
  public void testMethod(String name)
              throws IOException {
    File f = new File(name);
    FileInputStream g = new
              FileInputStream(f);
    int c;
    @ emTestMethod
    f.clone();
    try {
      f.toURL();
      while ((c = g.read()) != -1) {
        System.out.write(c);
      }

      try {
         new URI("uri").
             parseServerAuthority();
         g.read();
      }
      catch (IOException e) {
      }
    }
    catch (MalformedURLException e) {
    }
    catch (URISyntaxException e) {
    }
    @
    g.close();
  }
}
```



```
public class TestCase16 {
  public void testMethod(String name) throws
                                  IOException {
    File f = new File(name);
    FileInputStream g = new FileInputStream(f);
    int c;
    c = emTestMethod(f, c, g);
    g.close();
  }

  private int emTestMethod(File f, int c,
      FileInputStream g) throws IOException,
            CloneNotSupportedException {
    f.clone();
    try {
      f.toURL();
      while((c = g.read()) != -1) {
        System.out.write(c);
      }

      try {
        new URI("uri").parseServerAuthority();
        g.read();
      }
      catch(IOException e) { }
    }
    catch(MalformedURLException e) { }
    catch(URISyntaxException e) { }
    return c;
  }
}
```

Implementation in Stratego

- We've seen that we need to collect quite a lot of information (variables used inside and outside the fragment, variable types, new method name, exceptions thrown, etc.)

- Ideal solution: a single traversal collects all necessary information and generates dynamic rewrite rules to apply in the correct places.

Extract Method on a single traversal – skeleton (1)

```
strategies

  prepare-rules =
    rules (UsedVars : () -> [])
    ; rules (AfterUserVars : () -> [])
    ; rules (AssignedToVars : () -> [])

  extract-method =
       ?ClassDec(_, _)
       ; handle-class

    <+ ?MethodDec(_, _)
       ; handle-method

    <+ ?Block(_)
       ; handle-block

    <+ ?Extract(_, _)
       ; handle-extract

    <+ where(var-dec; handle-var-dec)

    <+ where(var-use; handle-var-use)

    <+ where(var-assign; handle-var-assign)

    <+ all(extract-method)
```

Extract Method on a single traversal – skeleton (2)

```
strategies

  prepare-rules =
    rules (UsedVars : () -> [])
    ; rules (AfterUserVars : () -> [])
    ; rules (AssignedToVars : () -> [])

  extract-method =
       ?ClassDec(_, _)
       ; handle-class

    <+ ?MethodDec(_, _)
       ; handle-method

    <+ ?Block(_)
       ; handle-block

    <+ ?Extract(_, _)
       ; handle-extract

    <+ where(var-dec; handle-var-dec)

    <+ where(var-use; handle-var-use)

    <+ where(var-assign; handle-var-assign)

    <+ all(extract-method)
```

```
handle-var-dec =
  map(declare-var-dec-rules)

declare-var-dec-rules =
  ?(x, (t, init))
  ; rules(VarType : x -> (t, init))
  ; try(<InExtract>(); rules (IsDeclared : x -> x))

handle-var-use =
  ?x
  ; try(
     <InExtract>();
     <not(IsDeclared)>x;
     <UsedVars>() => uv;
     rules (UsedVars : () -> [x | uv])
    )
  ; try(
     <InAfterExtract>();
     <IsDeclared>x;
     <AfterUsedVars>() => auv;
     rules (AfterUsedVars : () -> [x | auv])
    )

handle-var-assign =
  ?x
  ; try(
     <InExtract>();
     <not(IsDeclared)>x;
     <AssignedToVars>() => av;
     rules (AssignedToVars : () -> [x | av])
    )
```

Extract Method on a single traversal – skeleton (3)

```
strategies

  prepare-rules =
    rules (UsedVars : () -> [])
    ; rules (AfterUserVars : () -> [])
    ; rules (AssignedToVars : () -> [])

  extract-method =
       ?ClassDec(_, _)
       ; handle-class

    <+ ?MethodDec(_, _)
       ; handle-method

    <+ ?Block(_)
       ; handle-block

    <+ ?Extract(_, _)
       ; handle-extract

    <+ where(var-dec; handle-var-dec)

    <+ where(var-use; handle-var-use)

    <+ where(var-assign; handle-var-assign)

    <+ all(extract-method)
```

```
handle-extract =
  {| InExtract :
    ?Extract(Id(x), stmts)
    ; rules(ExtractedCode : () -> (x, stmts))
    ; rules(InExtract : () -> ())
    ; rules(ContainsExtract : () -> ())
    ; all(extract-method)
    ; rules(InAfterExtract : () -> ())
    ; where(<(AssignedToVars; length)>() => nrAv)
    ; where(
        <UsedVars>();
        map(\ x -> ExprName(Id(x)) \) => e*
      )
    ; (    <eq>(nrAv, 0)
           ; ![ x(e*); ]|
        <+ <eq>(nrAv, 1)
           ; <AssignedToVars>() => [y]
         ; ![ y = x(e*); ]|
      )
  |}
```

Extract Method on a single traversal – skeleton (4)

```
strategies

  prepare-rules =
    rules (UsedVars : () -> [])
    ; rules (AfterUserVars : () -> [])
    ; rules (AssignedToVars : () -> [])

  extract-method =
       ?ClassDec(_, _)
       ; handle-class

    <+ ?MethodDec(_, _)
       ; handle-method

    <+ ?Block(_)
       ; handle-block

    <+ ?Extract(_, _)
       ; handle-extract

    <+ where(var-dec; handle-var-dec)

    <+ where(var-use; handle-var-use)

    <+ where(var-assign; handle-var-assign)

    <+ all(extract-method)
```

```
handle-class =
  {| RewriteClass :
    try(RewriteClass)
  |}

handle-method =
  {| VarType :
    all(extract-method)
  |}

handle-block =
  {| InBlock, CurLabel, IsDeclared,
     ContainsExtract, InAfterExtract, ExtractedCode :
    where(new => label)
    ; rules(InBlock+label)
    ; rules(CurLabel : () -> label)
    ; all(extract-method)
    ; try(
        <ContainsExtract>();
        define-rewrite-class-rule;
        rewrite-block
      )
  |}
```

Extract Method on a single traversal – skeleton (5)

```
strategies

  prepare-rules =
    rules (UsedVars : () -> [])
    ; rules (AfterUserVars : () -> [])
    ; rules (AssignedToVars : () -> [])

  extract-method =
       ?ClassDec(_, _)
       ; handle-class

    <+ ?MethodDec(_, _)
       ; handle-method

    <+ ?Block(_)
       ; handle-block

    <+ ?Extract(_, _)
       ; handle-extract

    <+ where(var-dec; handle-var-dec)

    <+ where(var-use; handle-var-use)

    <+ where(var-assign; handle-var-assign)

    <+ all(extract-method)
```

```
define-rewrite-class-rule =
  rules( RewriteClass :
     ClassDec(chead, ClassBody(cBody)) ->
     ClassDec(chead, ClassBody(newCBody))
     where
       <ExtractedCode>() => (x, stmts)
       ; <AssignedToVars>() => av
       ; (<eq>(<length>av, 1)
            < !av => [y];
              ; <VarType>y => (t, _)
              ; <conc>(stmts, [ |[ return y; ]| ])
            + !Void() => t
              ; !stmts
         ) => emBody
      ; <UsedVars>() => uv
      ; <AfterUsedVars>() => uav
      ; <union>(uv, uav)
      ; map(\x-> Param([], <(VarType;Fst)>x, Id(x)) \
          ) => param*
      ; <process-declarations>emBody => bstm*
      ; !|[ private t x(param*) { bstm* } ]| => em
      ; <conc>(cBody, [em]) => newCBody
  )
```

Handling exceptions

```
get-exceptions =
  define-exception-rules
  ; <union>(<bagof-TempException>(), <bagof-PermException>())

define-exception-rules =
  ?Try(block, catches)
  ; {| TempException, InTry :
       rules(InTry : () -> ())
       ; Try(define-exception-rules, id)
       ; where(
           <map(?Catch(Param(_, <id>, _), _))>catches
           ; eliminate-caught-exceptions
           ; ?exceptions
         )
    |}
  ; Try(id, define-exception-rules)
  ; where(
     <InTry>()
     < <map(\x->x where rules(TempException :+ () -> x ) \)>exceptions
     + <map(\x->x where rules(PermException :+ () -> x ) \)>exceptions
    )

 <+ ?Try(block, catches, finally)
    // same as above

 <+ ?Invoke(methodid, args)
    ; Invoke(id, define-exception-rules)
    ; where(get-exceptions-of-invocation)

 <+ all(define-exception-rules)
```

Possible improvements

- Disallowing all control flow is a bit harsh, since in some cases it can be allowed.

- Handling the changing of more than one variable in the extracted fragment could be supported.

- ... and certainly there might be some other details/cases we may have missed.