Module Road Map

Refactoring

- **Why Refactoring?**
- **Examples**
- **Common Refactorings**

What is Refactoring?

- Refactoring is the process of changing a software system so that
  - the external behavior is not altered, but
  - the internal structure is improved.
- Refactoring (http://www.refactoring.com/) is a "behavior-preserving transformation."
  - Small changes per transformation
  - ⇒ less likely to go wrong
  - System works after each change

Organizing Java Code

- Eclipse comes with extensive support for organizing and refactoring Java code
- It is possible to:
  - Generate getters and setters for the fields
  - Organize missing import statements
  - Move fields, methods, classes
  - Rename methods, classes, packages

Why Refactoring?

- Methods might no longer do (only) what their name suggests.
- Functionality that should be in two different classes might be in the same class.
- Functionality that should be in one class might be duplicated in two or more classes.
- Improve the design of existing code.
- Gain a better understanding of the code.

Refactoring » Example

- Consider a method for computing the room charge for a hotel:

```
public double getRoomCharge()
{
 double roomCharge = 0.0;
 ... code to compute room charge...
 return roomCharge;
}
```

- What other factors might go into computing the room charge?

Refactoring » Example

- Of course, to print out a bill for a customer, we also need to include incidentals and taxes …

```
public double getRoomCharge()
{
    double roomCharge = 0.0;
  ... code to compute room charge...
    // now add the incidentals to roomCharge
    ... code to add up incidentals ...
    // now add the tax for the room to the charge
    ...several lines of code to compute the tax...
    return roomCharge;
}
```

- What’s inelegant about this method now?
    - 3 sets of calculations in one function. Method does 3 things.
    - The name is not illustrative of what the method does.

Refactoring » Example

- Better: Changing the name of the method (for example, `calculateCustomerCharge`).
- Does this fix the problem?
  - No, We also need to change the name at all call sites.
  - We need to update the documentation.
  - If this method overrides a method in another class, the other name may need to be changed too. Ditto if this method implements an interface.
- This is known as the *Rename Method* refactoring.

Refactoring » Example

- Let’s refactor our getRoomCharge() method.

```
public double calculateCustomerCharge()
{
  double roomCharge = getRoomCharge();
  double incidentals = getIncidentals();
  double tax = getTax(roomCharge, incidentals);
  return roomCharge + incidentals + tax;
}
```

- What have we done?
  - We defined additional methods to compute incidentals, tax, etc.
  - In order to do this, we added local variables for the quantities that are being calculated in the new methods.
  - Some pre-existing local variables ended up being parameters to the new method.
  - The returned value is different from what was returned in the pre-existing method.

Refactoring » Common Refactorings

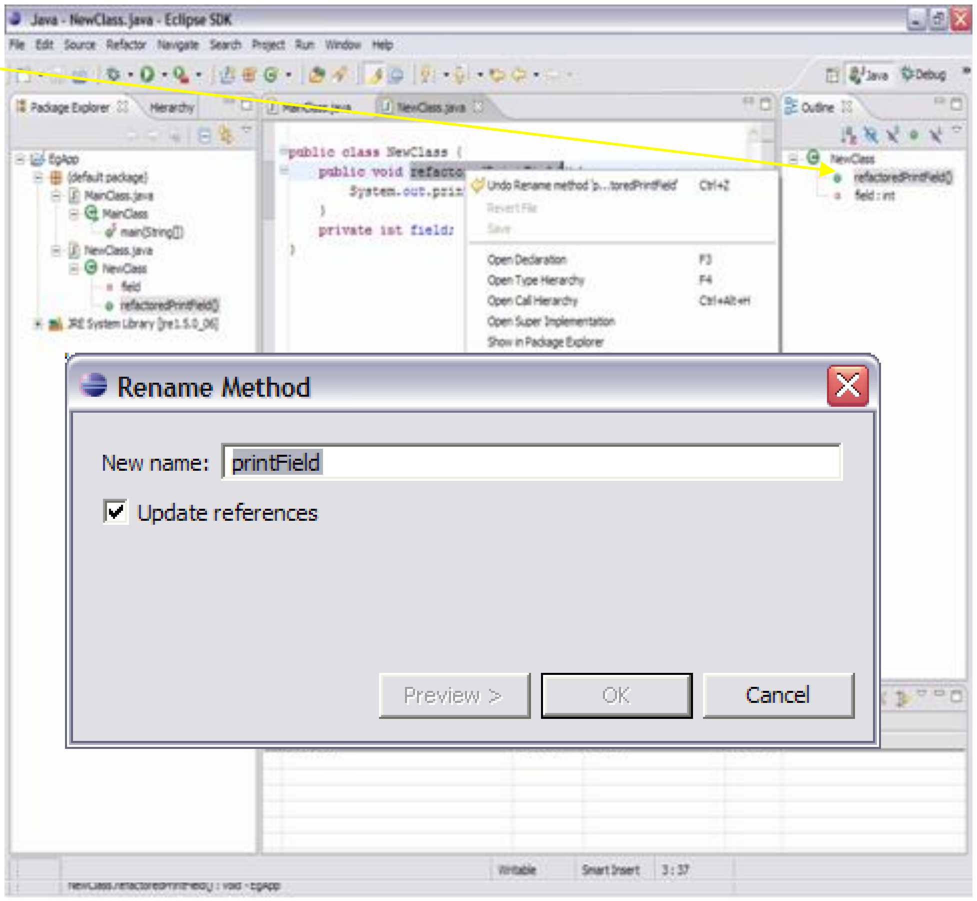
- *Rename*
  - *Methods*, *Fields*, *Packages*, *Projects*, *Parameters*, or *Local Variables*
- *Encapsulate Field* (generate getter and setter)
- *Pull up* a *Field* or *Method* (into superclass)
- *Push down* a *Field* or *Method* (into subclass)
- *Extract Method*, *Local Variable*, or *Constant* from an *Expression*
- *Change Method Signature*

Refactoring » Renaming a Method Using Eclipse

- In a Java view showing methods (e.g., the Outline view) select the method to be renamed.
- From the view's pop-up menu, select Refactor » Rename, or select Refactor » Rename from the global menu bar

or

- In a Java editor, select a reference to or the declaration of the method to be renamed.
- From the editor's pop-up menu, select Refactor » Rename, or select Refactor » Rename from the global menu bar.
- This pops up the Rename Method dialog box.
- Click Preview to preview the changes, or click OK to perform the refactoring.


Java - NewClass.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Hierarchy
MainClass.java
NewClass.java
Outline
Undo Rename method 'p...toredPrintField' Ctrl+Z
Revert File
Save
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Open Super Implementation
Show in Package Explorer
Rename Method
Preview >
OK
Cancel
Writable
Smart Insert
3 : 37

```
public class NewClass {
    public void refacto...
        System.out.prin...
    }
    private int field;
}
```

New name: printField

[✓] Update references

Refactoring » Encapsulating a Field

- The *Encapsulate Field* refactoring can be used to convert a public instance variable into a private instance variable with accessor functions.

- Example: Inelegant code—

```
public PublicFieldClass{
        public String name ;
}

public OtherClass{
   public static void main(String[] args){
    PublicFieldClass example = new PublicFieldClass();
    example.name = "Joe";
    System.out.println("My name is " + example.name);
   }
}
```

Refactoring » Encapsulating a Field

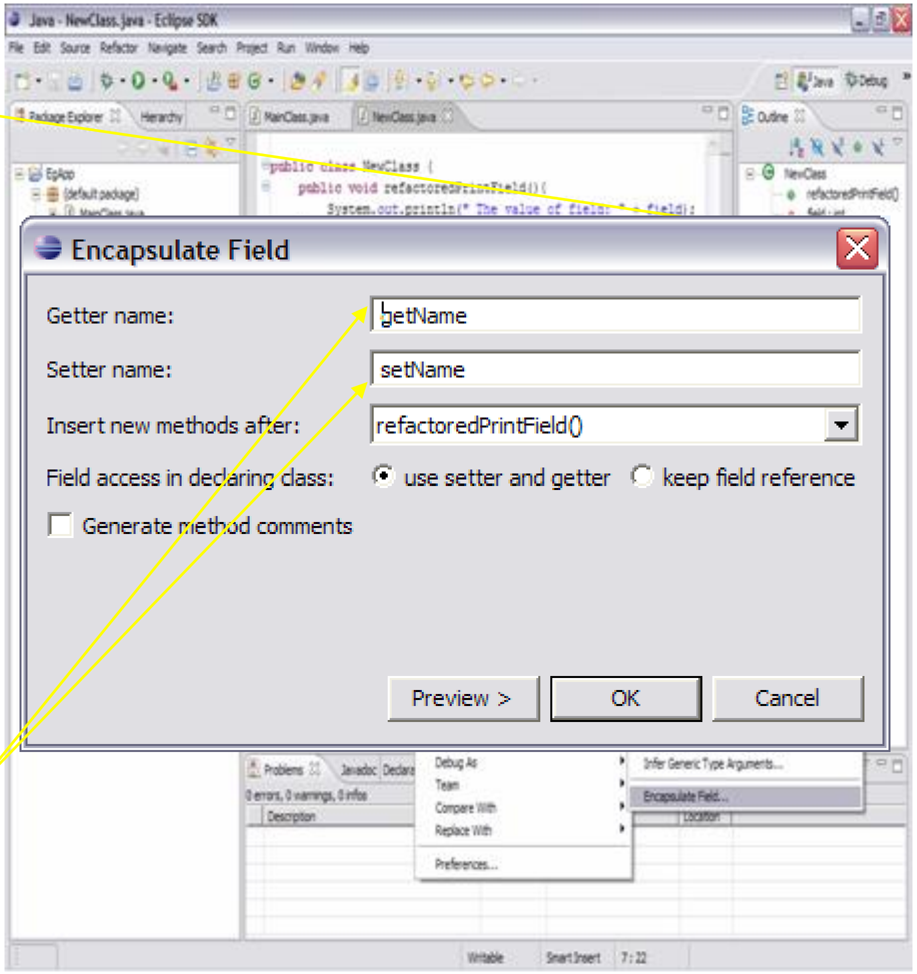
- After refactoring, we have …

```
public EncapsulatedFieldClass{
       private String name;
       public String getName(){
                  return name;
       }
       public setName(String newName){
                   name = newName;
       }
}

public OtherClass{
       public static void main(String[] args){
                  EncapsulatedFieldClass example =
                             new EncapsulatedFieldClass()
                  example.setName("Joe") ;
                  System.out.println("My name is " +
                              example.getName()) ;
        }
}
```

Refactoring » Encapsulating a Field Using Eclipse

- Select the field in one of the Java views (e.g., Outline, Package Explorer or Members view).
- From the field's pop-up menu, select Refactor » Encapsulate Field... , or from the menu bar, select Refactor » Encapsulate Field...
- Alternatively, in the Java editor, select the field.
- From the menu bar, select Refactor » Encapsulate Field... , or from the editor's pop-up menu, select Refactor » Encapsulate Field...
- This pops up the Encapsulate Field dialog.
- Type the names of the accessor routines in the Getter name and Setter name text fields.
- Click Preview to preview the changes or Click OK to perform refactoring.



```
public class NewClass {
    public void refactoredPrintField() {
        System.out.println(" The value of field: " + field);
```

Getter name: getName

Setter name: setName

Insert new methods after: refactoredPrintField()

Field access in declaring class: [✓ use setter and getter] keep field reference

☐ Generate method comments

Refactoring » Pull Up Method

- Moves a field or method to a superclass of its declaring class.
- Suppose you have the same method—or nearly the same method—in two different classes in your system. It may be a good idea to centralize the behavior in a superclass.

```
public class Employee extends Person {
     String getName() {
               ...
     }
}

public class Student extends Person {
     String getName() {
        ...
     }
}
```

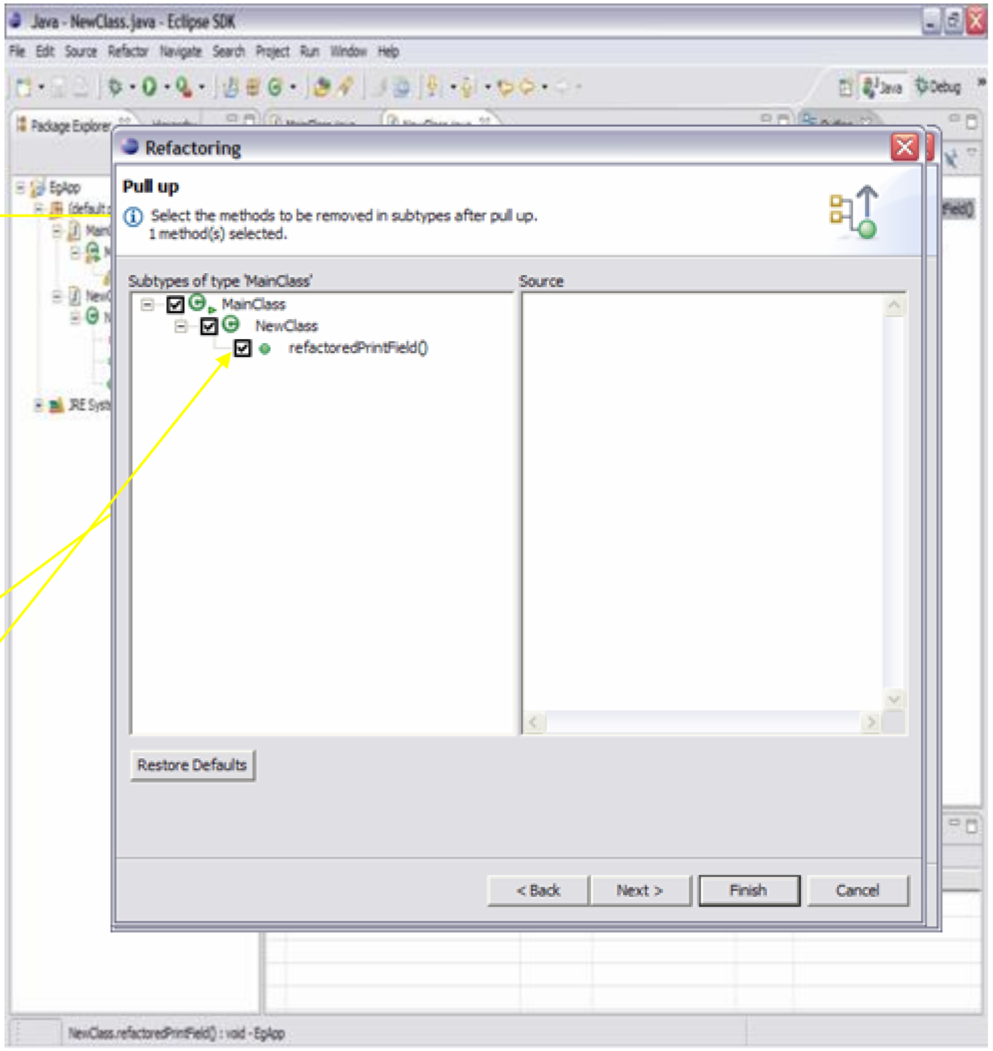
Refactoring » Pull Up Method

- After the Pull up Method refactoring is applied …

```
public class Person {
     String    getName() {
     ...
     }
}
```

Refactoring » Pull Up Method Using Eclipse

- In a Java view (e.g., Outline, Package Explorer, Members), select the members that you want to pull up.
- From the menu bar, select Refactor » Pull Up or from the pop-up menu, select Refactor » Pull Up.
- This pops up the Pull up dialog.
- Select the methods to pull up and their new declaring class. Click Next.
- Select the methods to be removed in the subtypes after pull up and click Next to review the changes.


Pull up

Select the methods to be removed in subtypes after pull up.
1 method(s) selected.

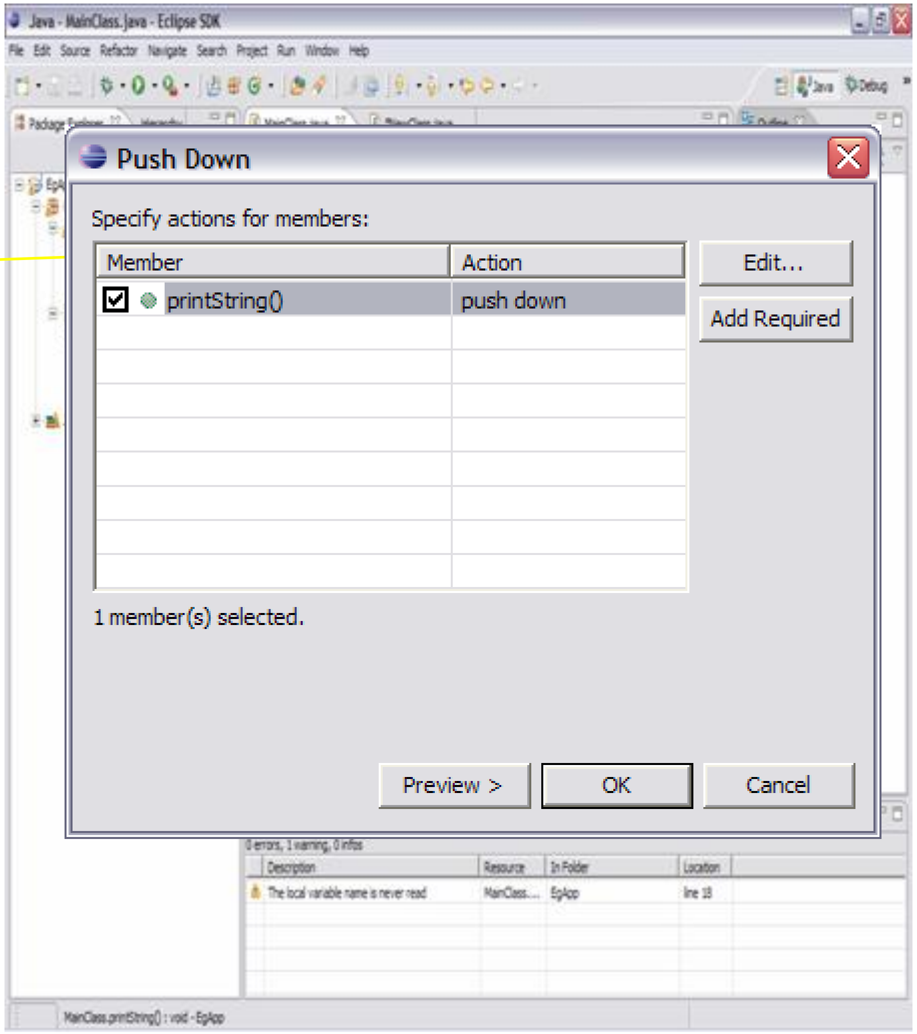
Java - NewClass.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Refactoring
Subtypes of type 'MainClass'
MainClass
NewClass
refactoredPrintField()
Source
Restore Defaults
< Back
Next >
Finish
Cancel
NewClass.refactoredPrintField() : void - EpApp

Refactoring » Push Down Method

- Reverse of Pull up Method.
- Moves a set of methods and fields from a class to its subclasses.
- Can be used when some of the subclasses do not use a method defined in the superclass.

Refactoring » Push Down Method Using Eclipse

- In a Java view (e.g., Outline, Package Explorer, Members), select the members that you want to push down.
- From the menu bar, select Refactor » Push Down or from the pop-up menu, select Refactor » Push Down.
- The Push Down dialog will open.
- Click Preview to preview the changes or click OK to perform the refactoring.


Java - MainClass.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Push Down
Edit...
Add Required
Preview >
OK
Cancel

Specify actions for members:

Member	Action
☑ ● printString()	push down

1 member(s) selected.

0 errors, 1 warning, 0 infos

Description	Resource	In Folder	Location
The local variable name is never read	MainClass...	EgApp	line 18

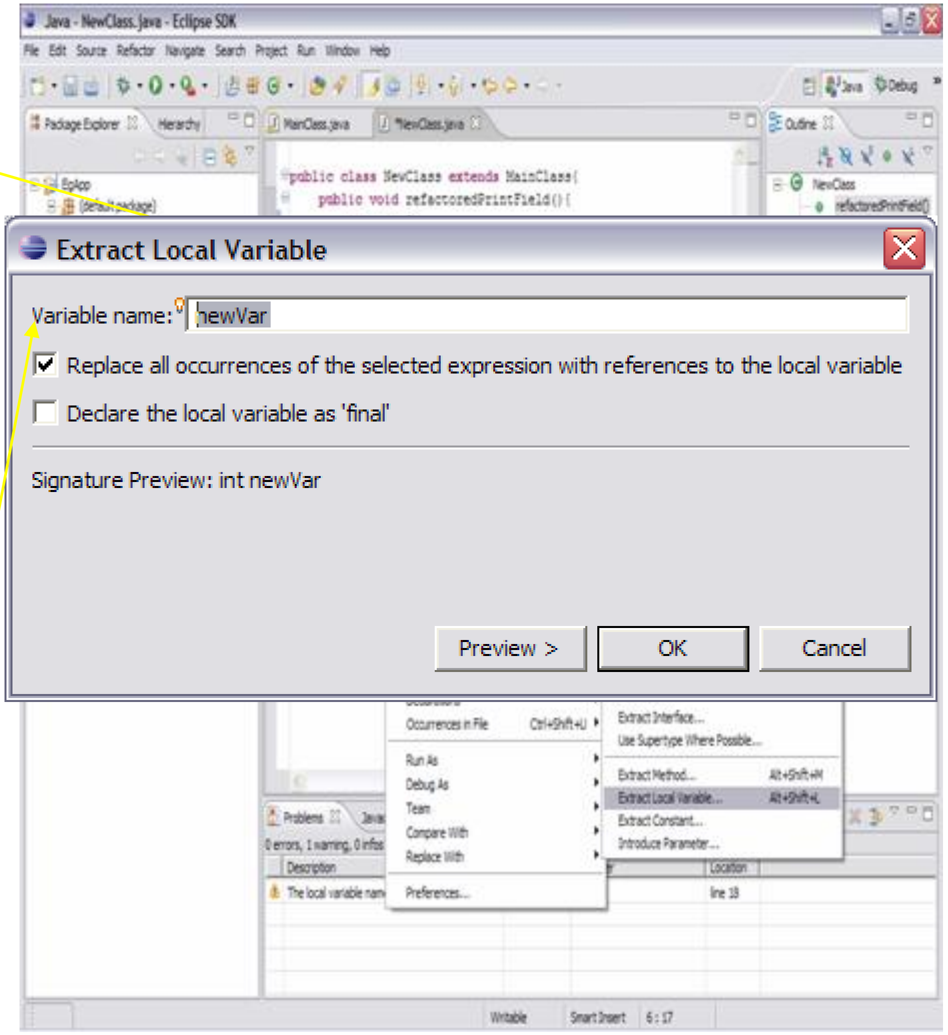
MainClass.printString() : void - EgApp

Extracting a Local Variable

- An expression that occurs in more than one place is replaced with a local variable, whose value is calculated only once.
- If a program needs to use the same value in multiple places, it can be calculated only once and then used wherever needed.
- Advantages
  - Makes the code more efficient.
  - Makes the code more readable.
  - Creates a single point of maintenance for the logic of computing the expression.

Refactoring » Extracting a Local Variable Using Eclipse

- In a Java editor, select the expression that you want to extract to a local variable.
- From the editor's pop-up menu, select Refactor » Extract Local Variable or from the menu bar, select Refactor » Extract Local Variable.
- This will open the Extract Local Variable dialog box.
- Type the name of the variable in the Variable name text field.
- Click Preview to preview the changes or click OK to perform the refactoring.



```
public class NewClass extends MainClass{
    public void refactoredPrintField(){
```

Extract Local Variable

Variable name: newVar

[✓] Replace all occurrences of the selected expression with references to the local variable

[ ] Declare the local variable as 'final'

Signature Preview: int newVar

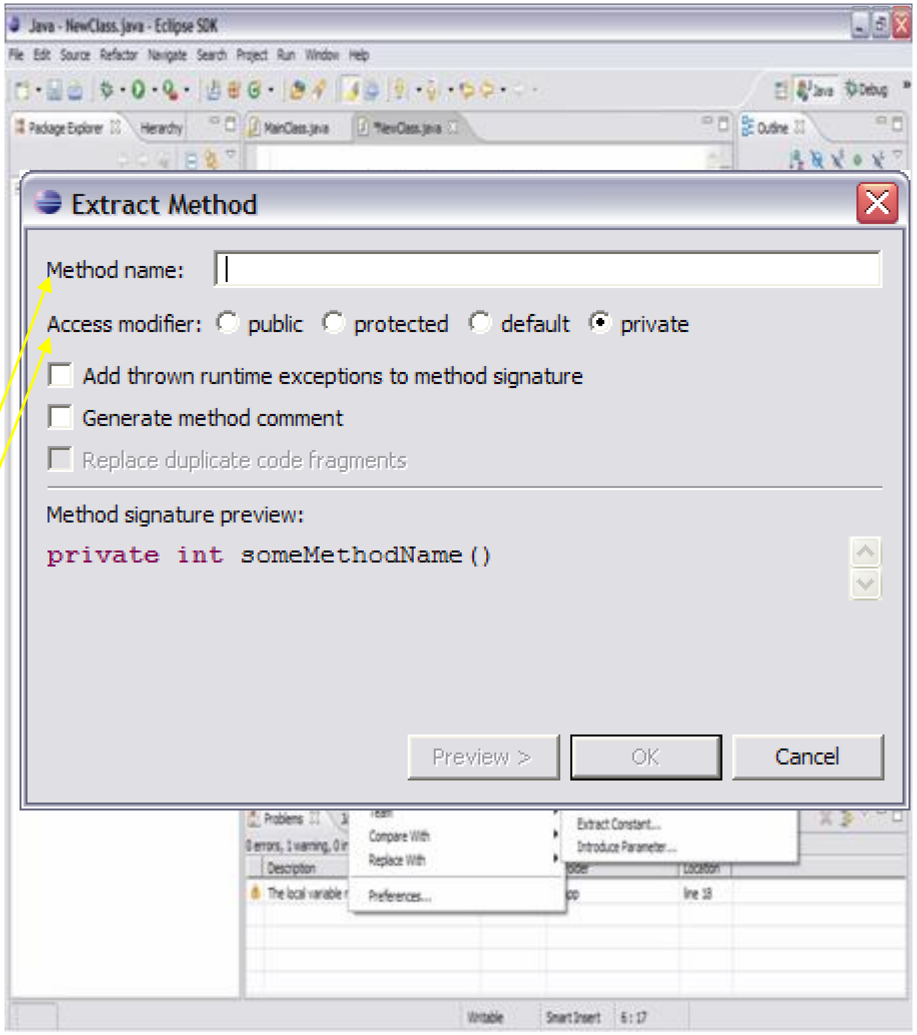
Preview > | OK | Cancel

Refactoring » Extracting a Method

- Creates a new method containing the statements or expression currently selected and replaces the selection with a reference to the new method.
- Advantages
  - Code readability
  - Minimize code duplication

Refactoring » Extracting a Method Using Eclipse

- In an editor, select a set of statements or an expression from a method body.
- From the pop-up menu in the editor, select Refactor » Extract Method from the menu bar, select Refactor » Extract Method.
- This opens the Extract Method dialog box.
- Type the method name in the Method name text field.
- In the Access Modifier list, specify the method's visibility (public, default, protected, or private).
- Click Preview to preview the changes or click OK to perform the refactoring.


Extract Method

Method name:

Access modifier: ( ) public ( ) protected ( ) default (●) private

☐ Add thrown runtime exceptions to method signature

☐ Generate method comment

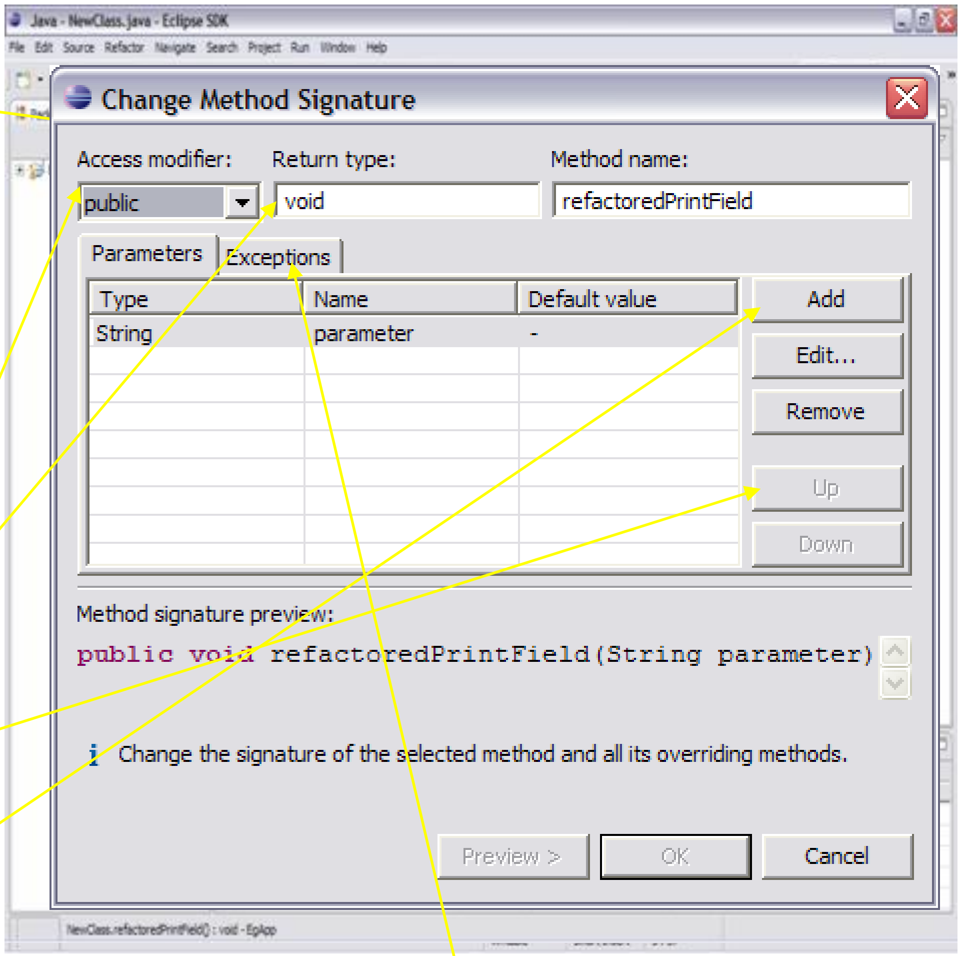
☐ Replace duplicate code fragments

Method signature preview:

```
private int someMethodName()
```

Refactoring » Change Method Signature

- Select the method in a Java view (e.g. Outline, Package Explorer, Members).
- From the menu bar, select Refactor » Change Method Signature or from the method's pop-up menu, select Refactor » Change Method Signature.
- This opens the Change Method Signature dialog box.
- Use the Access Modifier drop-down to control the method's visibility.
- Change the method's return type or name by editing the provided text fields.
- Select one or more parameters and use the Up and Down buttons to reorder the parameters (you can see a signature preview below the parameter list).
- Use the Add button to add a parameter; you can then edit its type, name and default value in the table.


Java - NewClass.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Change Method Signature
Parameters
Exceptions
Add
Edit...
Remove
Up
Down
Preview >
OK
Cancel

Access modifier: public
Return type: void
Method name: refactoredPrintField

Type	Name	Default value
String	parameter	-

Method signature preview:

```
public void refactoredPrintField(String parameter)
```

Change the signature of the selected method and all its overriding methods.

NewClass.refactoredPrintField() : void - EgApp

- Switch to the Exceptions tab to add or remove thrown exceptions. Click Preview to preview the changes

Other Refactorings Supported by Eclipse

- Renaming
  - a package
  - a compilation unit
  - a type
  - a local variable
  - method parameters
- Extracting
  - a constant
  - an interface from a type
- Inlining
  - a local variable
  - a method
  - a constant
  - static members between types
  - an instance method to a component
- Converting
  - a local variable to a field
  - an anonymous inner class to a nested class
  - a nested type to a top level type
- Replacing
  - references to a type with references to one of its supertypes
  - a single reference to a type with a reference to one of its supertypes
  - an expression with a method parameter
  - constructor calls with factory method invocations