**Assignment # 01**

**Group Members:**

**Muhammad Mauz 181400116**

[**Amina Shahbaz**](mailto:181400086@gift.edu.pk) **181400086**

[**Sania Abdul Rasheed**](mailto:181400088@gift.edu.pk) **181400088**

**Refactoring in IDE:**

Refactoring is the process of changing a software system in such a way that it does not alter the function of the code yet improves its internal structure.

**Refactoring using Eclipse:**

* Right clicking on a Java element in the Package Explorer view and selecting the Refactor menu item.
* Right clicking on a Java element in the Java editor and selecting the Refactor menu item.
* Selecting a Java element in either the Package Explorer view or Java Editor and clicking **Shift + Alt + T**.

**Does Eclipse support refactoring?**

**Compositional Refactoring**

This Eclipse bundle provides new Quick Assist actions to support refactorings in a compositional paradigm. In the compositional paradigm, the tool automates small, predictable steps of a refactoring

**Renaming**

**Renaming Variables and Methods:**

We can rename variables and methods by following these simple steps:

* Select the element
* Right-click the element
* Click the Refactor > Rename option
* Type the new name
* Press Enter

We can also perform the second and third steps by using the shortcut key, **Alt+Shift+R.**

When the above action is performed, Eclipse will find every usage of that element in that file and replace them all in place.

We can also use an advanced feature to update the reference in other classes by hovering over the item when the refactor is on and clicking on Options:

This will open up a pop-up where we can both rename the variable or method and have the option to update the reference in other classes

**Renaming Packages**

We can rename a package by selecting the package name and performing the same actions as in the previous example. A pop-up will appear right away where we can rename the package, with options like updating references and renaming subpackages.We can also rename the package from the Project Explorer view by pressing F2

Renaming Classes and Interfaces

We can rename a class or interface by using the same actions or just by pressing F2 from Project Explorer. This will open up a pop-up with options to update references, along with a few advanced options:

**Extracting**

Now, let's talk about extraction. Extracting code means taking a piece of code and moving it.

For example, we can extract code into a different class, superclass or interface. We could even extract code to a variable or method in the same class.

Eclipse provides a variety of ways to achieve extractions, which we'll demonstrate in the following sections.

**Extract Class:**

Now, suppose we want to extract out the driver details to a different class. We can do this by right-clicking anywhere within the class and choosing the Refactor > Extract Class option.This will open up a pop-up where we can name the class and select which fields we want to move, along with few other options We can also preview the code before moving forward. When we click OK, Eclipse will create a new class named Driver, and the previous code will be refactored to.

**Extract Interface:**

We can extract an interface by right-clicking anywhere within the class and choosing the Refactor > Extract Interface option, or we can use the **Alt+Shift+T** shortcut key command to bring up the menu directlyThis will open up a pop-up where we can enter the interface name and decide which members to declare in the interface.

**Extract Superclass:**

We may want to extract the non-employment-related properties to a Person superclass. To extract items to a superclass, we can right-click anywhere in the class and choose the Refactor > Extract Superclass option, or use **Alt+Shift+T** to bring up the menu directly

**Extract Method:**

To invoke the Extract Method wizard, we need to perform the following steps:

* Select the lines of code we want to extract
* Right-click the selected area
* Click the Refactor > Extract Method option
* The last two steps can also be achieved by **keyboard shortcut Alt+Shift+M**

**Extract Local Variables:**

we need to:

* Select the item
* Right-click and choose Refactor > Extract Local Variable
* The last step can also be achieved by the **keyboard shortcut Alt+Shift+L.**

**Extract Constant:**

we need to:

* Select the item
* Right-click and choose Refactor > Extract Constant

**Inlining:**

* Select the item we want to inline
* Right-click and choose the Refactor > Inline option
* The last step can also be achieved by **keyboard shortcut Alt+Shift+I:**

**Push Down and Pull Up:**

If we have a parent-child relationship (like our previous Employee and Person example) between our classes, and we want to move certain methods or variables among them, we can use the push/pull options provided by Eclipse.

As the name suggests, the Push Down option moves methods and fields from a parent class to all child classes, while Pull Up moves methods and fields from a particular child class to parent, thus making that method available to all the child classes.

For moving methods down to child classes, we need to right-click anywhere in the class and choose the Refactor > Push Down optionThis will open up a wizard where we can select items to push down.

Similarly, for moving methods from a child class to parent class, we need to right-click anywhere in the class and choose Refactor > Pull Up.This will open up a similar wizard where we can select items to pull up

**Changing a Method Signature**

we can follow a few simple steps:

* Select the method or place the cursor somewhere inside
* Right-click and choose Refactor > Change Method Signature
* The last step can also be achieved by keyboard **shortcut Alt+Shift+C.**

**Moving**

Sometimes, we simply want to move methods to another existing class to make our code more object-oriented.

* Select the lines you want to move
* Right-click and choose the Refactor > Move option
* The last step can also be achieved by **keyboard shortcut Alt+Shift+V:**

**Java Eclipse Code Refactoring Shortcuts and Tips:**

**1. Eclipse Refactor — Extract Class**

The 'Extract class...' refactoring (**from Alt+Shift+T**) extracts a group of fields into a separate class and replaces all occurrences to fit the new structure.

**2. Eclipse Refactor — Extract Interface**

Use the 'Extract interface...' refactoring technique to extract an interface out of a class (**Alt-Shift-T** to bring up a menu).

**3. Eclipse Refactor — Extract Superclass**

If you want to extract part of a class into a superclass, use **Alt+Shift+T** to bring up the menu and select, as shown below. There are a lot of customization options available in the dialog!

**4. Eclipse Refactor — Extract Method**

Use this **Alt + Shift + M** shortcut with the Eclipse key to extract a selection to a method. This helps you move a selected block of code to a separate method with ease.

**5. Eclipse Refactor — Push Down**

Ever wondered what 'Push down...' in the refactoring menu ( **Alt+Shift+T**) does? It moves methods and fields from a parent class to #all extenders. Quite handy!

**6. Eclipse Refactor — Extract Local Variable**

You can now declare a local variable's type as 'var' while extracting it using **Alt+Shift+L** (shortcut for Extract Local Variable refactoring).

**7. Eclipse Refactor — Rename**

Use the **Alt + Shift + R** shortcut to rename a variable, method, class, or even a package name. This is the most frequently used shortcut in code refactoring.

**8. Eclipse Refactor — Change Method Signature**

Use the **Alt + Shift + C** Eclipse shortcut key to change the signature of a method. Place the cursor inside a method or select the method name, and then, press this shortcut. The Change Method Signature dialog appears. You can change various elements of method signature, such as access modifier, return type, parameters, exceptions, etc

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Introduce Parameter Object** | Replaces a set of parameters with a new class, and updates all callers of the method to pass an instance of the new class as the value to the introduce parameter.   |  |  | | --- | --- | | Available: | Methods or on text selection resolving to a method | | Options: | Enable 'Keep original method as delegate to changed method' in the Introduce Parameter Object dialog to keep the original method. | |
| **Introduce Indirection** | Creates a static indirection method delegating to the selected method.   |  |  | | --- | --- | | Available: | Methods or on text selection resolving to a method | | Options: | Enable 'Redirect all method invocations' to replace all calls to the original method by calls to the indirection method. | |
| **Introduce Factory** | Creates a new factory method, which will call a selected constructor and return the created object. All references to the constructor will be replaced by calls to the new factory method.   |  |  | | --- | --- | | Available: | Constructor declarations | |
| **Introduce Parameter** | Replaces an expression with a reference to a new method parameter, and updates all callers of the method to pass the expression as the value of that parameter.   |  |  | | --- | --- | | Available: | Text selections that resolve to expressions | |
| **Encapsulate Field** | Replaces all references to a field with getter and setter methods.   |  |  | | --- | --- | | Available: | Field or a text selection resolving to a field.  This refactoring is also available as [quick assist](https://help.eclipse.org/latest/topic/org.eclipse.jdt.doc.user/reference/ref-java-editor-quickassist.htm) on field declarations and references selected in the editor. | |
| **Generalize Declared Type** | Allows the user to choose a supertype of the reference's current type. If the reference can be safely changed to the new type, it is.   |  |  | | --- | --- | | Available: | Type references and declarations of fields, local variables, and parameters with reference types | |
| **Infer Generic Type Arguments** | Replaces raw type occurrences of generic types by parameterized types after identifying all places where this replacement is possible.   |  |  | | --- | --- | | Available: | Projects, packages, and types | | Options: | 'Assume clone() returns an instance of the receiver type'. Well-behaved classes generally respect this rule, but if you know that your code violates it, uncheck the box.  Leave unconstrained type arguments raw (rather than inferring <?>)'. If there are no constraints on the elements of e.g. ArrayList a, uncheck this box will cause Eclipse to still provide a wildcard parameter, replacing the reference with ArrayList<?>. | |
| **Migrate JAR File** | Migrates a JAR File on the build path of a project in your workspace to a newer version, possibly using refactoring information stored in the new JAR File to avoid breaking changes.   |  |  | | --- | --- | | Available | JAR Files on build path | |
| **Create Script** | Creates a script of the refactorings that have been applied in the workspace. Refactoring scripts can either be saved to a file or copied to the clipboard. See *Apply Script*.   |  |  | | --- | --- | | Available | Always | |
| **Apply Script** | Applies a refactoring script to projects in your workspace. Refactoring scripts can either be loaded from a file or from the clipboard. See *Create Script*.   |  |  | | --- | --- | | Available | Always | |