Important Refactorings

General IntelliJ refactoring reference[1]

Rename

Change variable/method name

Simply changes the name of a variable consistently across all its uses.

For methods, this will also potentially change the name of methods defined in parent classes or interfaces.

Change method signature

You can change a method's signature, including:

- Change the parameter names and types
- Add new parameters
- Delete existing parameters
- Rearrange the order of the parameters
- Change the return type and/or method name

Extract and Inline

Extract and inline are opposites.

extract local variable / constant / field / parameter

Select an expression, and replace its usage with a newly created variable. If the expression appears multiple times, you can choose to replace all occurences.

```
totalAmount += item.price * item.quantity;
// Extract variable on "item.price * item.quantity;"
int itemTotal = item.price * item.quantity;
totalAmount += itemTotal;
```

Use to give names to expressions and describe their intent better. Also used in preparation of extract method.

Special case for "extract parameter": If the expression contains the only way that another parameter is used, the system will offer to remove that parameter:

[1] https://www.jetbrains.com/help/idea/refactoring-source-code.html

```
public void processItem(int[] array, int index) {
    System.out.print(array[index]);
}

public static void main(String[] args) {
    Processor processor = new Processor();
    int[] items = { 1, 3, 5};
    for (int i = 0; i < items.length; i++) {
        processor.processItem(items, i);
    }
}
```

When we "extract parameter" on array[index] we get:

```
public class Processor {
  public void processItem(int item) {
    System.out.print(item);
  }

  public static void main(String[] args) {
    Processor processor = new Processor();
    int[] items = { 1, 3, 5};
    for (int i = 0; i < items.length; i++) {
      processor.processItem(items[i]);
    }
  }
}
```

Extract method

Create a new method out of the selected statements.

- The selection must not change the value of local variables.
- Any local variables used in the selection become parameters to the function.
- Return value is appropriate for the selection.

```
    int total = 0;
    for (Item item : items) {
      total += item.price * item.quantity;
    }
```

We extract method for item.price * item.quantity; to get:

```
    for (Item item : items) {
      total += getItemTotal(item);
    }

// ... elsewhere ...
  private static int getItemTotal(Item item) {
    return item.price * item.quantity;
  }
```

Inline local variable / constant / field

Replace all occurences of the symbol with its value. Only works if symbol would always equal this value.

Inline method

Replace one or all occurences of a method's calls with the appropriately modified body. Optionally, delete the method.

Example same as "Extract Method" in reference

Class creation and alteration

These refactorings create new classes out of existing features, or otherwise manipulate the class structure.

Extract method object

Turns a whole selection into a new method in a new class. Any variables needed are passed as arguments to the class constructor.

```
int total = 0;
for (Item item : items) {
  total += item.price * item.quantity;
}
System.out.println(total);
```

If we "Extract method object" out of the first four lines, we get:

```
int total = new Totaller(items).invoke();
System.out.println(total);

// ...  elsewhere ...
private static class Totaller {
    private Item[] items;

    public Totaller(Item... items) {
      this.items = items;
    }

    public int invoke() {
      int total = 0;
      for (Item item : items) {
        total += item.price * item.quantity;
      }
      return total;
    }
  }
```

Extract parameter object

Combine a number of parameters to a method into a single object of a newly created class.Useful when the same parameters appear together often, or when we just want to reduce the number of parameters to a method.

```
total += getItemTotal(prices[i], quantities[i]);

// ...
private int getItemTotal(int price, int quantity) {
  return price * quantity;
}
```

Extracting a parameter object from the getItemTotal method, gives us a new Item class:

```
total += getItemTotal(new Item(prices[i], quantities[i]));

...
private static int getItemTotal(Item item) {
    return item.getPrice() * item.getQuantity();
}
...
private static class Item {
  private final int price;
  private final int quantity;

  private Item(int price, int quantity) {
    this.price = price;
    this.quantity = quantity;
  }

  int getPrice() {
    return price;
  }

  int getQuantity() {
    return quantity;
  }
}
```

Extract delegate

Extract some fields and methods into a new class, which the current class delegates calls to.

TODO: Find good example

Extract superclass/interface/subclass

Extract certain methods and/or fields into a subclass or a superclass, or use them to form an interface.

```
private static class Item {
    private final int price;
    private final int quantity;
```

```
    private Item(int price, int quantity) {
      ...
    }

    int getPrice() { return price; }
    int getQuantity() { return quantity; }
}
```

We can extract the Itemizable interface from the two methods getPrice and getQuantity.

```
private static class Item implements Itemizable {
    ...
}

...

public interface Itemizable {
  int getPrice();
  int getQuantity();
}
```

The system will also offer to search through all uses of the class Item and try to replace them with uses of the interface Itemizable instead.

Movement

A number of move refactorings are available. The system automatically adjusts references for us, and warns us about problem cases.

Convert to static method / Convert to instance method / Move method

Turn an instance method into a static method. The instance is then passed as a parameter, if needed.

Conversely, you can turn a static method into an instance method of one of its parameters. The method will then automatically move to the corresponding class.

```
    private static int getItemTotal(Itemizable item) {
      return item.getPrice() * item.getQuantity();
    }
```

Becomes:

```
    // In class Item    (possibly interface Itemizable?)
    int getItemTotal() {
        return getPrice() * getQuantity();
    }
```

Move class

Move a class to be an inner class of another class or to be an outer class, or move an outer class to an inner class, or move classes from one package to another.

pull members up / push members down

In the context of a class and it superclass or interface, move definitions and/or implementations "up" (towards the superclass) or "down" (towards the subclasses).

```
// In interface Itemizable
default int getItemTotal() {
  return getPrice() * getQuantity();
}
```

Becomes

```
// In interface Itemizable
int getItemTotal();

...
// In class Item
int getItemTotal() {
  return getPrice() * getQuantity();
}
```

Manual Refactorings

Move code

Some times it helps to move some lines of code elsewhere within the same method. This allows them to be grouped up with other lines of code, which might allow further refactorings (e.g. extract method).

Be careful when moving code around, to make sure you preserve the code's *semantics*.

Split loop

Oftentimes a loop is doing too much. It is often possible to split it in two loops, each doing some part of the work. This again might enable further refactorings.