Inline

IntelliJ IDEA provides the following inline refactorings:

- The *Inline Variable* refactoring replaces redundant variable usage with its initializer. See example. This refactoring is opposite to the Extract Variable refactoring.

  > The variable must be initialized at declaration. If the initial value is modified somewhere in the code, only the occurrences before modification will be inlined.

- *Inline Method* refactoring results in placing method's body into the body of its caller(s). You can opt to:
  - inline all occurrences of the method, and delete the method
  - inline only a single occurrence, and retain the method

  See example. This refactoring is opposite to the Extract Method refactoring.
- *Inline to Anonymous Class* refactoring allows replacing redundant class with its contents.
- *Inline Constructor* allows compressing a chain of constructors, if one of them is a special case of another. Seeexample.
- *Inline JSP/JSPX* works like a regular Java inline refactoring.
- *Inline Superclass* refactoring results in pushing superclass' methods into the class where they are used, and removing the superclass.

Examples

Inline variable

Before	After

`public void method() {`
`  int number = anotherClass.intValue();`
`  int b = a + number;`
`}`
	`public void meth`
	`  int b = a + anotherClass.intVa`
	`}`

`public void method() {`
`  AnotherClass.InnerClass aClass = anotherClass.innerClass;`
`  int a = aClass.i;`
`}`
	`public void meth`
	`  int a = anotherClass.innerCla`
	`}`

Inline method

Before	After

`public void method() {`
`  int c=add(a,b);`
`  int d=add(a,c);`
`}`
 	
`private int add(int a, int b) {`
`  return a+b;`
`}`
	`public void method() {`
	`  int c= a + b;`
	`  int d= a + c;`
	`}`

Before	After

`public ArrayList method() {`
`  String[] strings = {"a","b","c"};`
`  ArrayList list=add(strings);`
`  return list;`
`}`
 	
`private ArrayList add(String[] strings) {`
`  ArrayList list = new ArrayList();`
`  for (int i=0; i< strings.length; i++)`
`  {list.add(strings[i]);}`
`  return list;`
`}`
	`public ArrayList method() {`
	`  String[] strings = {"a","ab","abc"};`
	`  ArrayList list1 = new ArrayList();`
	`  for (int i=0; i< strings.length; i++)`
	`  {list.add(strings[i]);}`
	`  ArrayList list = list1;`
	`  return list;`
	`}`

Inline constructor

Before	After

`public class Class {`
`  public int`
`  varInt;`
`  public Class() {`
`  this(0);`
`  }`
 	
`  public Class(int i) {`
`  varInt=i;`
`  }`
 	
`  public void method() {`
`  Class aClass=new Class();`
`  ...`
`  }`
`}`
	`public class Class {`
	`  public int`
	`  varInt;`
	`  public Class(int i) {`
	`  varInt=i;`
	`  }`
	`  public void method() {`
	`  Class aClass=new Class(0);`
	`  ...`
	`  }`
	`}`

Inline superclass

Before	After

`public class Bar {`
`  ...`
`  int calculations1() { ... }`
`  int calculations2() { ... }`
`}`
 	
`class Foo extends Bar {`
`  int someMethod() {`
`  ...`
`  if (something > calculations1()) {`
`  ...`
`  return calculations2();`
`  }`
`  ...`
`  }`
`}`
	`class Foo {`
	`  ...`
	`  int someMethod() {`
	`  ...`
	`  if (something > calculations1()) {`
	`  ...`
	`  return calculations2();`
	`  }`
	`  ...`
	`  }`
	`  int calculations1() {...}`
	`  int calculations2() {...}`
	`}`

To perform the inline refactoring

1. Place the caret in the editor at the desired symbol to be inlined.
2. Do one of the following:
   - On the main menu or on the context menu of the selection, choose **Refactor | Inline**.
   - Press **Ctrl+Alt+N**.
3. In the Inline dialog , that corresponds to the selected symbol, specify the inlining options.
4. Preview and apply changes.

See Also

Procedures:

- Refactoring Source Code

Reference:

- Inline Method
- Inline Dialogs

Web Resources:

- Developer Community