

Programming by Intention

Introducing Intention Actions

IDEA 3.0 has a multitude of code completion features designed to ease the development process while at the same time improving the quality of the code you write. Features like Code Inspection, Smart Type completion, Live Templates, and code generators appear only when you specifically request them. However, another feature, *Intention Actions*, is always working silently in the background.


Do what I mean, not what I say!

Sometimes as programmers we can get ahead of ourselves. We reference classes we haven't yet imported, assign values to variables we haven't defined, make calls to methods we haven't written, and so forth. Technically these could be considered mistakes, but they are mistakes made with the best of intentions.

This is where intention actions come in to play. When IDEA suspects a possible problem with your code, it will do much more than simply bring the problem to your attention. If possible, it will also suggest a corrective solution. As an added bonus, IDEA can actually carry out its recommendations on your behalf. It can change the way variables are assigned, create missing references, and much more. Each of these operations is known as an *intention action*. An intention action is an action that IDEA will perform for you because it knows what you *intended* to do.

Working with intentions

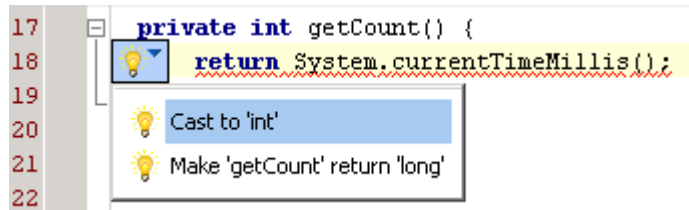
When IDEA encounters a syntax error or other portions of code that it believes it can assist you with, it will alert you with the intention action alert symbol, which is shown as a little yellow light bulb, flush left on the page. An example is shown below where we are trying to assign a value to an undeclared variable called `status`.

```
22 List list = new ArrayList();
23 list.add(new Integer(100));
24  status = false;
```

Because any given line of source code may trigger several different intention action alerts, the alert icon only appears when your cursor is on the effected region. In the above example, the alert is only displayed when your cursor is around the `status` variable. Note that although this alert is centered on a syntax error, there are situations in which IDEA will offer intention actions for code that is syntactically correct as well. For example, it may have a suggestion on how to optimize or improve code structure, or can logically deduce a common operation you might wish to perform.

Using intention actions to fix errors

The primary purpose of intention actions is to fix problems and oversights in your code. For example, if you try to return a `long` value from a method with a declared return type of `int`, IDEA will not only flag the syntax error before the compiler catches it, but will also offer two possible suggestions based on what you may have intended to do. In our example case, it will suggest that you either cast the returned value to an `int`, thereby satisfying the method signature, or change the method signature to return a `long` value as shown below.



Whichever action we take, IDEA will take care of actually implementing the change. We'll learn how to invoke these actions a little later.

Using intention actions as programming shortcuts

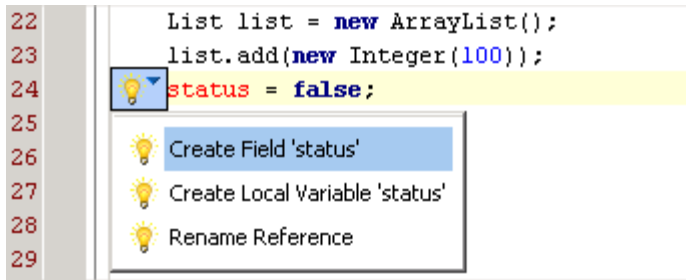
Intention actions do more than fix mistakes. Once you are familiar with the types of intention actions IDEA provides you can use them as programming shortcuts. For instance, I know that if I make a call to a method that does not yet exist, IDEA's intention actions will offer to create the method for me. With this in mind, I can now simply make a call to the method I wish I had, and let the intention actions stub it out for me. It is amazing how much repetitive work like defining variables, casting objects from collections, and importing packages can be eliminated by relying on the intention features of IDEA and having them do the grunt work for you.

Relying on intention actions quickly becomes a natural and efficient way to program. Instead of trying to understand all of the methods, instance variables, and other elements up front, I can let intention actions define them as I need them. It's kind of like not bothering to stop and lookup the spelling of a word in a dictionary while you are writing because you know that the spellchecker will fix it for you!

Selecting an intention action

When you see the intention action alert, your next step is to select which, if any, intention action you want to perform. IDEA will never try to fix the code for you without your approval. To review IDEA's suggestions and accept an intention action, left-click on the light bulb icon or press **Alt + Enter** to view the suggestion list. IDEA will present a list of possible intention actions via a drop down menu emitted from the light bulb icon. Only intention actions which will result in valid code will be shown in the list. Let's resume our earlier example where we were trying to assign a value to an undeclared field.

```
22 List list = new ArrayList();
23 list.add(new Integer(100));
24 status = false;
25
26
27
28
29
```





In this case, we see that there are four possible intention actions to select from. Use the mouse or keyboard to select one of them and IDEA will automatically implement the changes. While most conditions are applicable only to one or two intention actions, this one has four possible solutions. Briefly, the four options above will perform the following actions:

- **Create Field** Creates a new boolean field (instance) variable called `status`
- **Create Local Variable** Creates a new boolean local variable called `status`
- **Rename Reference** Changes the assignment to another, existing boolean in scope

Select the option that matches your intended use for the code. If none are appropriate, you can simply press escape to close the intention action list and leave the code unchanged.

Disabling intention alerts

If so desired, it is possible to disable the alert for any given type of intention action. You may want to do this if you find that one of IDEA's intention actions is not suited to your style of programming and that you do not wish to be continually alerted to its availability. To disable any action you encounter, bring up the suggestion list by clicking on the alert icon or by hitting the Alt + Enter shortcut. Left-click on the light bulb icon next to each action listed in the suggestion list to toggle its status from on  to off . Clicking the icon again will enable the action once again.

Once disabled, that intention action's availability will no longer cause the intention action alert icon to appear in the editor. You can however, still perform the intention action by using the Alt + Enter key combination while in the area that you suspect calls up the suggested action list. This will also allow you to turn suppressed alerts back on.

You can disable intention action alerts altogether with the Popup Alerts area of the status bar to turn off alerts. You can either left-click the area to toggle its status from On to Off, or right-click it and select a status from its context menu. This also disables the intention action to automatically import required classes.

Some common intentions actions

IDEA supports literally dozens of different intention actions that span all areas of development. There are intention actions for correcting errors, optimizing class structure,

working with exceptions, building EJBs, and much more. While I can't cover all them here, I'll highlight some of the ones you are most likely to encounter in your day to day programming operations. Knowledge of IDEA's support for these actions will allow you to anticipate their usage, allowing you to use them as shortcuts to performing common operations.

Exceptions

- **Move catch up** Reorganizes a catch statement amongst multiple catches so that the most specific exception is caught first. For example an `IOException` should be caught before its super class `Exception`.
- **Surround with try/catch** Available when an uncaught exception is detected. Generates a try catch block around the offending call, catching the appropriate exceptions.
- **Add exception to catch** Available inside a method with an existing try/catch block. Adds a catch block for a currently uncaught exception to the existing try/catch block.
- **Add exception to throws** Available when code inside a method can generate an uncaught exception. Adds the exception in question to the throws clause of the method signature.

Methods

- **Fix method return** Changes the method's signature to declare a return type matching that of the returned value.
- **Create method from usage** Available when your code attempts to call a nonexistent method. Create a new method whose parameters and return type are inferred from the context of the method call attempt.
- **Implement methods** Available in classes who implement an interface or whose super class is abstract and have not implemented all of the required methods. Creates stub method implementations. Equivalent to the command `Code | Implement Methods...`
- **Implement abstract method** Active when inside an abstract method of an abstract class. Locates classes that extend the abstract class and stubs out the methods in them if they have not implemented it already. This is useful when adding a new method to an abstract class with a number of existing concrete classes.

Variables and assignments

- **Split declaration** Available during a variable declaration and assignment operation, for example `int count = 0`. Splits the operation into two lines of code, declaration of the variable alone, followed by an assignment of the value to the variable on the next line.
- **Implement abstract class** Appears inside abstract classes only, when on the class declaration line. Creates a new class by extending the abstract class. You will be prompted for which of the abstract methods, if any, you wish to implement in the

new class. IDEA will create the new source file and add it to your version control system, if you are using one.

- **Variable type fix** Changes a variable's declared type to be compatible with an assignment operation. For example, if you try to assign a `boolean` to an `int` variable, this action will change the declared variable type to `boolean`.
- **Create field from parameter** Appears inside an object's constructor which has unused parameters. Creates a new field (an instance variable) based on the unused parameter and uses the constructor to initialize its value.
- **Create field/variable/parameter from usage** Introduces a new variable based on an assignment or usage in context. Depending on the context, you can create fields, local variables, and method parameters.

Other intention actions

- **Fetch external resource** Appears in the XML editor when the document references an external resource, such as a DTD. Downloads the resource to the local server so that IDEA may use it for validation purposes. You can review or edit the selection in the Resources area of the IDEA preferences.
- **Ignore external resource** As above, but tells IDEA not to attempt to validate against the resource nor warn you about it in the future. To unignore a resource, remove it from the ignore list in the Resources area of the IDEA preferences.
- **Invert if condition** Available on conditional statements (select the `if` keyword to activate). Inverts the condition and swaps the body of the conditional with the `else` block (or the code following the conditional). The resulting operation is logically equivalent.
- **Remove redundant cast** Removes a cast that is unnecessary because the cast value is already of the correct type.
- **Add type cast** Casts a reference to the expected type as allowed to satisfy the conditions. For example, if you are pulling a `String` from a collection and assigning it to a local variable, IDEA will cast the value from an object to a `String`.