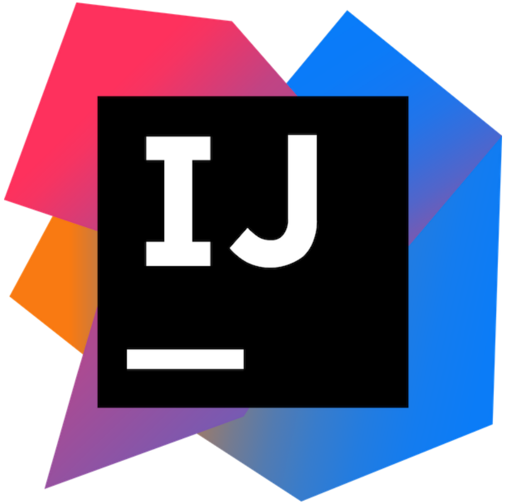
Características de un IDE

1. IntelliJ Idea
2. Características
3. Más características

IntelliJ

1. IntelliJ Idea



IDE

- Editor de texto
- Compilador/intérprete
- *Debugger*
- *File Browser*

Características

2. Características

Autoformatear

Autocompletar

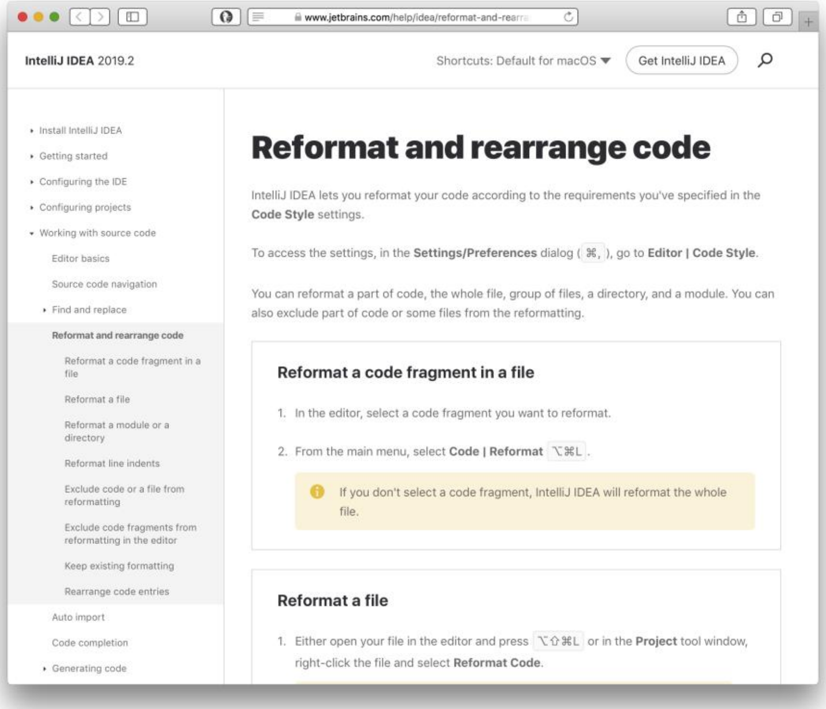
Generar código

Live templates

Comentarios

2. Características

Autoformatear



Reformat and rearrange code

IntelliJ IDEA lets you reformat your code according to the requirements you've specified in the **Code Style** settings.

To access the settings, in the **Settings/Preferences** dialog (⌘,), go to **Editor | Code Style**.

You can reformat a part of code, the whole file, group of files, a directory, and a module. You can also exclude part of code or some files from the reformatting.

Reformat a code fragment in a file

1. In the editor, select a code fragment you want to reformat.
2. From the main menu, select **Code | Reformat** ⌥⌘L .

If you don't select a code fragment, IntelliJ IDEA will reformat the whole file.

Reformat a file

1. Either open your file in the editor and press ⌥⇧⌘L or in the **Project** tool window, right-click the file and select **Reformat Code**.

[IntelliJ *Reformat and Rearrange*](IntelliJ Reformat and Rearrange)

2. Características

Autoformatear

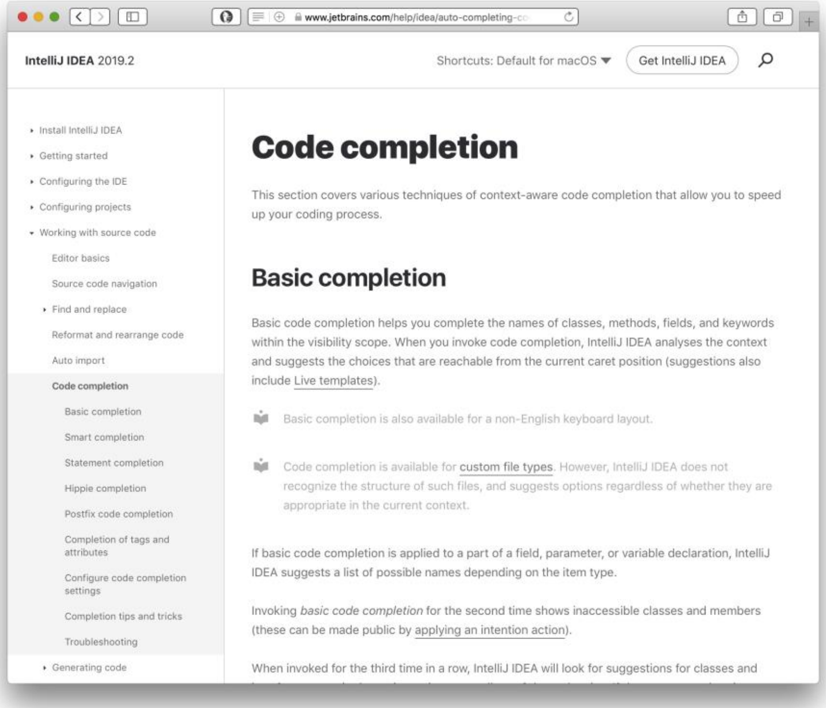
Autocompletar

Generar código

Live templates

Comentarios

Autocompletar



Code completion

This section covers various techniques of context-aware code completion that allow you to speed up your coding process.

Basic completion

Basic code completion helps you complete the names of classes, methods, fields, and keywords within the visibility scope. When you invoke code completion, IntelliJ IDEA analyses the context and suggests the choices that are reachable from the current caret position (suggestions also include Live templates).

Basic completion is also available for a non-English keyboard layout.

Code completion is available for custom file types. However, IntelliJ IDEA does not recognize the structure of such files, and suggests options regardless of whether they are appropriate in the current context.

If basic code completion is applied to a part of a field, parameter, or variable declaration, IntelliJ IDEA suggests a list of possible names depending on the item type.

Invoking *basic code completion* for the second time shows inaccessible classes and members (these can be made public by applying an intention action).

When invoked for the third time in a row, IntelliJ IDEA will look for suggestions for classes and

[IntelliJ *Code completion*](IntelliJ Code completion)

2. Características

Autoformatear

Autocompletar

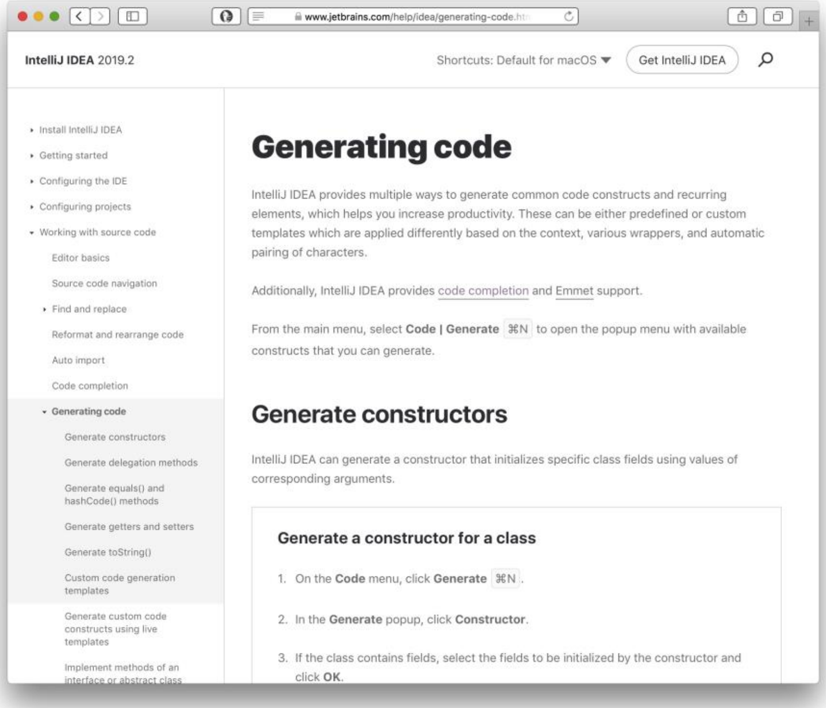
Generar código

Live templates

Comentarios

2. Características

Generar código



www.jetbrains.com/help/idea/generating-code.ht
IntelliJ IDEA 2019.2
Shortcuts: Default for macOS
Get IntelliJ IDEA
Install IntelliJ IDEA
Getting started
Configuring the IDE
Configuring projects
Working with source code
Editor basics
Source code navigation
Find and replace
Reformat and rearrange code
Auto import
Code completion
Generating code
Generate constructors
Generate delegation methods
Generate equals() and hashCode() methods
Generate getters and setters
Generate toString()
Custom code generation templates
Generate custom code constructs using live templates
Implement methods of an interface or abstract class

Generating code

IntelliJ IDEA provides multiple ways to generate common code constructs and recurring elements, which helps you increase productivity. These can be either predefined or custom templates which are applied differently based on the context, various wrappers, and automatic pairing of characters.

Additionally, IntelliJ IDEA provides code completion and Emmet support.

From the main menu, select **Code | Generate** ⌘N to open the popup menu with available constructs that you can generate.

Generate constructors

IntelliJ IDEA can generate a constructor that initializes specific class fields using values of corresponding arguments.

Generate a constructor for a class

1. On the **Code** menu, click **Generate** ⌘N.
2. In the **Generate** popup, click **Constructor**.
3. If the class contains fields, select the fields to be initialized by the constructor and click **OK**.

IntelliJ Generating Code

2. Características

Autoformatear

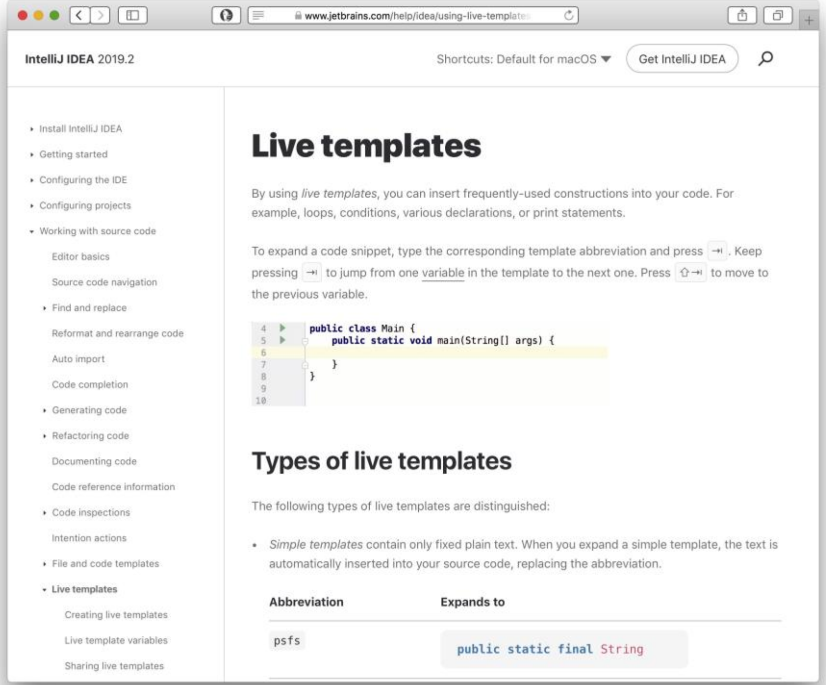
Autocompletar

Generar código

Live templates

Comentarios

Live templates



www.jetbrains.com/help/idea/using-live-template

IntelliJ IDEA 2019.2

Shortcuts: Default for macOS

Get IntelliJ IDEA

- Install IntelliJ IDEA
- Getting started
- Configuring the IDE
- Configuring projects
- Working with source code
  - Editor basics
  - Source code navigation
  - Find and replace
  - Reformat and rearrange code
  - Auto import
  - Code completion
  - Generating code
  - Refactoring code
  - Documenting code
  - Code reference information
  - Code inspections
  - Intention actions
  - File and code templates
  - Live templates
    - Creating live templates
    - Live template variables
    - Sharing live templates

Live templates

By using *live templates*, you can insert frequently-used constructions into your code. For example, loops, conditions, various declarations, or print statements.

To expand a code snippet, type the corresponding template abbreviation and press ⇥. Keep pressing ⇥ to jump from one variable in the template to the next one. Press ⇧⇥ to move to the previous variable.

```
public class Main {
    public static void main(String[] args) {

    }
}
```

Types of live templates

The following types of live templates are distinguished:

- *Simple templates* contain only fixed plain text. When you expand a simple template, the text is automatically inserted into your source code, replacing the abbreviation.

Abbreviation	Expands to
`psfs`	`public static final String`

[IntelliJ *Live Templates*]

2. Características

Autoformatear

Autocompletar

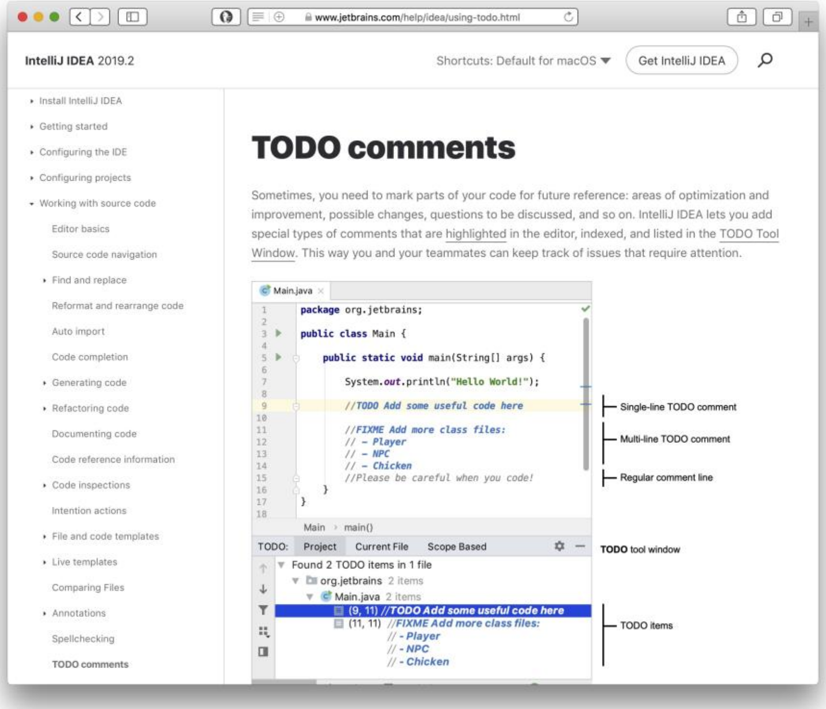
Generar código

Live templates

Comentarios

2. Características

Comentarios



TODO comments

Sometimes, you need to mark parts of your code for future reference: areas of optimization and improvement, possible changes, questions to be discussed, and so on. IntelliJ IDEA lets you add special types of comments that are highlighted in the editor, indexed, and listed in the TODO Tool Window. This way you and your teammates can keep track of issues that require attention.

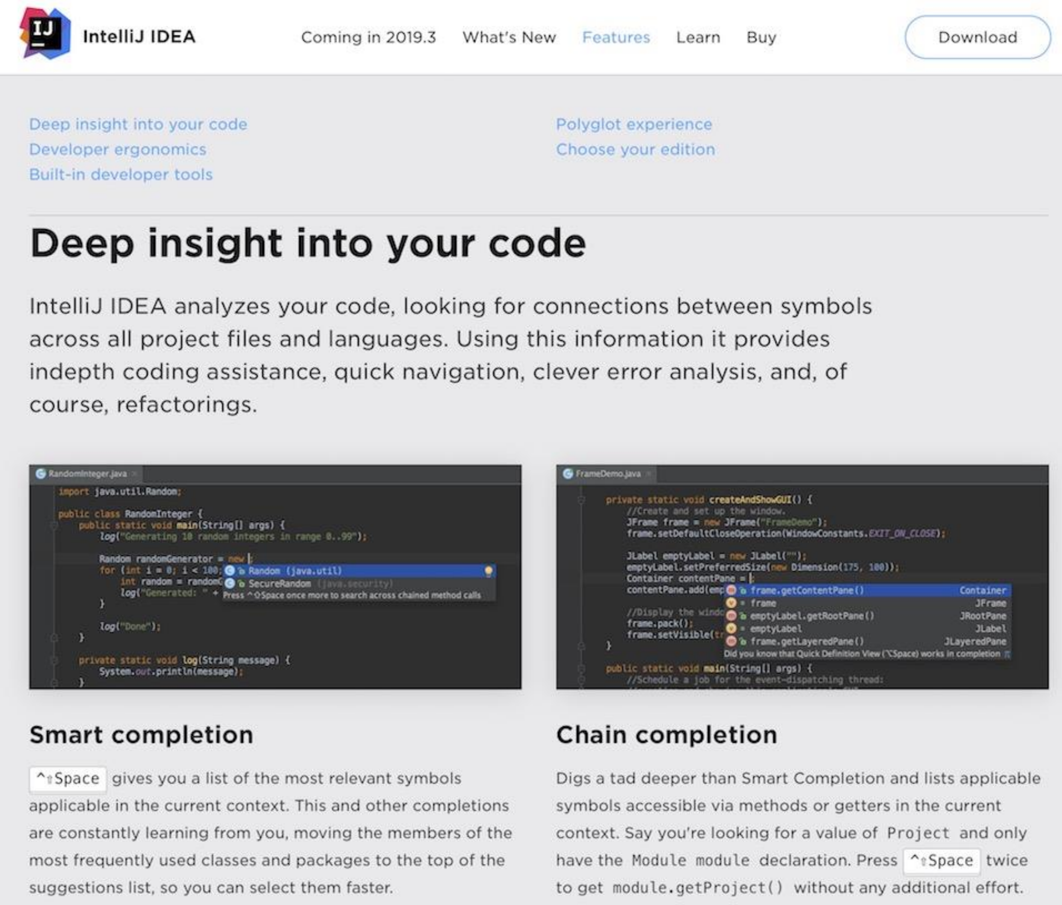
```
package org.jetbrains;

public class Main {

    public static void main(String[] args) {

        System.out.println("Hello World!");

        //TODO Add some useful code here

        //FIXME Add more class files:
        // - Player
        // - NPC
        // - Chicken
        //Please be careful when you code!
    }
}
```

```
Found 2 TODO items in 1 file
  org.jetbrains 2 items
    Main.java 2 items
      (9, 11) //TODO Add some useful code here
      (11, 11) //FIXME Add more class files:
               // - Player
               // - NPC
               // - Chicken
```

IntelliJ *Using TODO*

Más características

3. Más características



Deep insight into your code
Developer ergonomics
Built-in developer tools
Polyglot experience
Choose your edition

Deep insight into your code

IntelliJ IDEA analyzes your code, looking for connections between symbols across all project files and languages. Using this information it provides indepth coding assistance, quick navigation, clever error analysis, and, of course, refactorings.

RandomInteger.java

```
import java.util.Random;

public class RandomInteger {
    public static void main(String[] args) {
        log("Generating 10 random integers in range 0..99");

        Random randomGenerator = new |;
        for (int i = 0; i < 100;
            int random = randomG
            log("Generated: " +
        }

        log("Done");
    }

    private static void log(String message) {
        System.out.println(message);
    }
```

```
Random (java.util)
SecureRandom (java.security)
Press ^⇧Space once more to search across chained method calls
```

FrameDemo.java

```
    private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("FrameDemo");
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        JLabel emptyLabel = new JLabel("");
        emptyLabel.setPreferredSize(new Dimension(175, 100));
        Container contentPane = |;
        contentPane.add(emp

        //Display the windo
        frame.pack();
        frame.setVisible(tr
    }

    public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread:
```

```
frame.getContentPane()       Container
frame                           JFrame
emptyLabel.getRootPane()     JRootPane
emptyLabel                      JLabel
frame.getLayeredPane()    JLayeredPane
Did you know that Quick Definition View (⌥Space) works in completion
```

Smart completion

^⇧Space gives you a list of the most relevant symbols applicable in the current context. This and other completions are constantly learning from you, moving the members of the most frequently used classes and packages to the top of the suggestions list, so you can select them faster.

Chain completion

Digs a tad deeper than Smart Completion and lists applicable symbols accessible via methods or getters in the current context. Say you're looking for a value of `Project` and only have the `Module module` declaration. Press ^⇧Space twice to get `module.getProject()` without any additional effort.

IntelliJ *Features*

“There are two ways to write error-free programs; only the third one works. “

ALAN J. PERLIS, computer scientist

La Salle