AI Tooling for Scala Developers

What JetBrains Brings to the Table

Abstract

In the last few years, one of the leading new areas of development in JetBrains has been AI assistance. It started with simple access to an external large language model within IntelliJ IDEA—but we didn't stop there.
Today, JetBrains offers a wide range of new features and plugins that can make software developers work more quickly and productively. During this talk, we will go through the most important of them – AI Assistant, local and cloud completion, connecting to local LLMs, the integration of Model Context Protocol, and more. We will discuss how these tools can boost your productivity and improve your coding experience. We will also delve into… sorry, I mean, we will talk about JetBrains Junie, AI agents in general, and the future of AI assistance.
The talk will consist of animated infographics and live coding examples.

Notes about the abstract

- Due to time constraints and a loose connection to the rest of the talk, I will probably drop MCP.
- On the other hand, Junie gets more screen time – literally, there will be a video.
- Instead of animated infographics, I have recently preferred to make many slides that differ only by one small element. By clicking through them quickly, I make an impression of "animation." It's easier to control and takes less time to show, as I don't have to switch from slides to a video app.
    - Update: I can make animated gifs that will be played by the same app I use for static slides (ApolloOne).

- I will update the abstract and send it to the organizers.

Bio

I'm a Product Marketing Manager in the IntelliJ IDEA Scala team at JetBrains. I graduated from Warsaw University of Technology in 2005 with an MSc in artificial neural networks. I worked as a video game AI expert, a web tools and Android developer, and since two years ago a person who writes and talks publicly about features of IntelliJ IDEA with the Scala Plugin. In 2024, I became a Scala Ambassador.

Plan

- The slides with the agenda
  - "Hello from JetBrains", we are a Gold Sponsor, who am I
  - Topics we will go through: Code completion, AI Assistant, Mellum, Junie, licensing, and a few tips on how to use AI in coding
- The code example we will work on: a binary tree
  - A quick explanation of what it is and why I show it to the audience
  - Throughout the talk, I will switch between live coding and slides that explain what's happening in the code.
  - The binary tree code is already in a state where I can create a tree, add new elements to it, and print it out to the terminal.
  - GitHub repository: https://github.com/makingthematrix/binarytree
- Code completion
  - Mention traditional code completion (type inference, fields, and methods lookup)
  - Open Settings and go through the features under Code Completion and Inline Completion

        - Local completions
        - Cloud-based completions
        - Multi-line completions
        - A shortcut for AI code completions
        - Names suggestions
        - Sorting hints according to the AI Assistant's judgment
        - Completion policy (focused, creative)
        - Pop-up suggestions vs inline suggestions
    - **Note**: A lot of those features are called "... Completions". Be careful to explain the differences correctly. (This might help)
    - **Code example #1:** Write the "add" method for simultaneously adding many elements to the tree. Show inline code completion, name suggestions, and the in-editor prompt. Possibly write the same method twice using different features (it's a one-liner)
    - **Code example #2:** Write the "size" method, which recursively counts the number of elements in the tree. For now, use traditional code completion, e.g., match-exhaustive. We will come back to it.
    - Mention the **Next Edit** feature, show a short example
      (Next Edit is still in progress, let's come back to this point in August)
- Mellum
    - Mellum is already used for code completion in IntelliJ IDEA.
    - Show https://github.com/mellumai/mellum
    - Show https://github.com/JetBrains/mellum-sdk
    - Describe the main characteristics, and mention that Mellum is open source.
    - Show its page on HuggingFace.
    - Future plans for Mellum.

    - The possibility of fine-tuning Mellum for Scala.
- AI Assistant
    - How to install from Settings | Plugins.
    - Open the Open AI Chat window and go through the features:
    - How to include attachments.
    - How to manage different chats (create, rename, delete).
    - How to choose between models.
    - **Show a short video** on connecting to a local model through ollama.
    - Chat vs Edit mode: In the chat mode, let the AI Assistant explain the code in Tree.scala.
    - **Code example #3**: Rewrite the "size" method with the help of the AI Assistant in the Edit mode.
    - Make the AI Assistant generate a git commit message for the new code. (I might skip it due to time constraints)
    - How to learn more: Show the "Discover Features" panel.
    - Mention licensing, and that we will talk about it later.
- Junie
    - How to install Junie from AI Assistant (currently unavailable for IJ 2025.2 EAP)
    - Mention the webpage – Junie is in development, so it's important to check the page and the blog to learn about new features and changes.
    - Mention what LLM is used in Junie (Claude 3.7 and 4.0).
    - Mention Ask mode: works like the AI Assistant chat.
    - Why are there both AI Assistant and Junie? (Note that there is the Edit mode in AI Assistant)
    - Mention Brave mode: allows Junie to execute commands without asking for confirmation.

    - How to use Junie?
        - **Show a video** with Junie writing the code for iterating over the tree.
        - Show how the iterator should work (on the slides) and what is difficult about it.
        - The first solution is imperative and messy.
        - The second solution is Scala-idiomatic, but it's not optimal due to how the prompt was phrased: Junie coded the "toList" method internally and used it only to return its iterator.
- A few words on AI agents and LLM in general
    - Slides + a short discussion of limitations inherent to LMMs
    - Hints:
        - Be careful about prompts.
        - Check the results often.
        - Don't let AI generate tests for AI-generated code.
        - A good argument for using TDD: Generate tests first, based on prompts, and then generate the implementation.
- Licensing (leave for Q&A)
    - Questions:
        - Free access: Can people with IntelliJ IDEA Community Edition use AI Assistant and Junie? If yes, what are the limitations?
        - What are the differences between the access level for Ultimate and Community users?
        - Is there additional licensing for using AI Assistant and Junie, on top of the standard Ultimate license? If yes, how does it work?
        - Are there token quotas?
    - Answers:
        - https://www.jetbrains.com/ai-ides/

    - https://blog.jetbrains.com/blog/2025/04/16/jetbrains-ides-go-ai/#remark42__comment-71011913-2a9f-4922-86bd-32acc5495ca0 - some answers about the pricing
    - There will be an update on August 15

- The "thank you" slide.

Notes about the plan

After we decide on the contents, I need to go through the positioning document and think about how to include the slogans into the talk.

The following features are not currently in the plan due to time constraints but we can discuss if we should make changes to the plan to accommodate them.

- AI Assistant: Generate documentation
- AI Assistant: Explain errors in the runtime popup window
- AI Assistant: Explain the current branching situation in git (helpful for conflict resolution)
- AI Chat : /commands and #mentions
- Junie code example: Let it add a library and use it (e.g. use upickle to save the tree elements to json)
    - It can be either as a video or a quick live coding example
- A short discussion of MCP
    - https://plugins.jetbrains.com/plugin/26071-mcp-server
    - https://github.com/JetBrains/mcp-jetbrains

Research (private notes)

- Heuristic (traditional) line completion in IJ + SP
- JetBrains AI Assistant for prompts
- Local line completion
- Cloud-based line completion
- In-editor prompts
- Next Edit Suggestions (AI Assistant Demo: Next Edit Suggestions - 2025/07/17 10:51 CEST - Re...)
- Local models usage
- The theory behind AI agents
- Junie
- Model Context Protocol (https://modelcontextprotocol.io/introduction)
- The theory behind LLMs used for coding
- Instruct vs coding

Draft

Intro

Hello, everyone. My name is Maciej, and I work at JetBrains on the Scala Plugin team. As you surely know, the topic of this talk is AI tooling offered by JetBrains products. Specifically:

1. AI Assistant, our flagship tool
2. AI-powered Code completion in IntelliJ IDEA
3. Mellum, an open-source Large Language Model we offer
4. And Junie, our AI agent with capabilities that extend and augment those of AI Assistant

The talk will consist of slides and live coding. However, for Junie, I have a pre-recorded, edited video so that I can show you how it works within the 40-minute timeframe.

But first, let’s start with a quick quiz. Please raise your hand if you have:

1. Ever used any AI tooling while coding. By "AI tooling" I mean any form of help from AI features in your IDE of choice while coding, but also platforms like Mistral Chat or ChatGPT, or local Large Language Models run in the terminal.
2. Used AI Assistant in IntelliJ IDEA
3. Used Junie in IntelliJ IDEA
4. Used any other AI software

Snake

For the purpose of this talk, I coded a game of Snake in LibGDX and Scala 3. I don't want to show you examples that are too simple because that would significantly undersell what AI Assistant and Junie can do. However, it's also important that you understand exactly what's happening: what we're asking the AI to do, what the results are, and the reasoning behind them. So, after spending some time trying to find the right project, I decided the best solution was to write one myself.

[Run the game; remember to close the “run” window after each run]

I keep two versions of the code here. This is the finished version. I control the snake by hitting the left and right arrow keys. The Scala icons appear at random on empty tiles. The snake eats them and grows. If it goes over the edge, it reappears on the other side. And of course, if it collides with itself, it dies. We then see a popup with the score, and when we close it, the game ends. I like that I managed to make the board isometric as it makes the game a bit more difficult and also visually appealing. On the other hand, the snake is displayed simply as green tiles because actually drawing it turned out to be a bit too complicated.

And here's an "under construction" version, so to speak, from an earlier point in development. The snake crawls, and I can control it by hitting the keys, but that's all. It goes over the board and doesn't detect any collisions with the coins or with itself.

Let's write that missing code.

Code completions

A snake consists of a body, which is a list of positions on the board taken by the snake, and the direction it crawls in. Here's the snake's `crawl` method: We create a new `Pos2D` using the current head position and the direction of movement, and we create a new body by connecting that new head to the old body, but the old body loses its tail.

First, let's make the snake wrap around the board. To do that, we need to update the `crawl` method declaration by adding the board as a parameter. We need it to know where the board ends. Now, let's write new coordinates this way:

```
val x1 = body.head.x + snakeDir.x
val newX = if (x1 < 0) board.size - 1
           else if (x1 >= board.size) 0
           else x1
val x2 = body.head.y + snakeDir.y
val newY = if (x2 < 0) board.size - 1
           else if (x2 >= board.size) 0
           else x2
val newHead = Pos2D(newX, newY)
```

As you can see, while I’m writing, I get automatic code completions. They appear in the chosen color scheme but a little bit dimmed, and they can be accepted by hitting Tab. Let’s quickly run the code to see if it works, and we will talk more about it just after that. [run the code; remember to close the “run” window after each run]. Ok, it seems to work: the snake leaves the board on one side and reappears on the other.

In the top-right corner, we can see this whirlwind button. When we click it, the AI Widget opens up. We can use it to check your quota, and for quicker access to some of the most important settings in **Settings | Tools | AI Assistant**. From there to the **Inline Completion** settings page. Here, we can enable and disable local code completions and cloud code completions.

Local and cloud-based code completions

Local code completions tend to be simpler, but the Large Language Model that powers them runs entirely on your computer. All you need to do is download the model, and of course, you can then work without an internet connection.

On the other hand, cloud-based code completions are more powerful. As with local completions, you can choose the programming language they will be active in or even try out universal completion if your programming language is not on the list.

Another difference between them is latency. Local code completions are much faster. In fact, at JetBrains, we are now experimenting with a feature where you can see a few alternative suggestions at once and choose between them.

Other code completions settings

Next, we have the combobox for the completion policy. Code completions can be either focused, meaning AI Assistant will tend to suggest popular code snippets for a given situation. For example, if I write a `println` method and then a parameter, it will surely complete it with a call to the `.toString` method. Alternatively, we can switch it to "Creative" and see what happens, or choose something in between.

The checkbox, “Enable code completion indicator,” is purely decorative. The next one, “Enable automatic completions while typing” concerns how AI suggestions appear on the screen. Right now, I have them appearing automatically as I code. If you find it frustrating to see unnecessary code completions as you write, you can turn that off and instead use a shortcut – on Mac, it's **Option+Shift+Backspace** – to ask AI Assistant, "Okay, what do you think?" The checkbox, "Enable multi-line suggestions," simply allows AI Assistant to suggest longer and more complicated code. It works especially well in tandem with the Creative policy, as you give AI Assistant even more space to be creative.

AI Assistant chat window

The AI Assistant chat window is under the button here on the right. Its default purpose is to help you understand the code and help you come up with new ideas. It has access to the code in the editor, but you can attach more text files to it, or symbols (that is, classes or traits), or you can even prepare a prompt beforehand, add it to the prompt library, and then just choose it from there.

Your conversation with AI Assistant is remembered and used for the context of each new prompt. If you want to start from scratch, just click on “New Chat” here. Next to it, you have buttons to manage your chats collection.

In the bottom-right corner, you have an option to choose between many models on the backend. Also, if you or your company prefers to keep the code locally, or is worried about the quota costs, it's possible to connect to a local LLM from ollama, LM Studio, or OpenAI. I did some experiments with it, and even if you have a strong laptop, I don't recommend running an LLM on the same machine as IntelliJ IDEA. It works much better if you have it on a local server, plus the advantage is that everyone in the company can use it.

Let’s use Sonnet 4.5.

Now, let’s imagine that I wrote this method “crawl” a long time ago. Maybe I went on vacation, came back from a few weeks long bicycle trip, and remember nothing. So, let’s use AI Assistant to help me remember what this method was about. We can do it in two ways: either simply from the chat or by marking the code, hovering over the AI icon, and choosing to explain code. Both ways lead to the same action. [do it]

Edit code with AI Assistant

Now that we have a snake that moves around the board, let's use AI Assistant to write the logic that lets the snake eat the coin and grow. If I hit Control+Backspace, it opens a text field where I can write a prompt in-editor. Let's write:

"Create a new immutable field `hasCoin: Boolean` in the Snake class, and set it by default to false. In the `Board.update` method, if the snake's head is in the same position as a coin, update the snake in the board and change `setCoin` to true. In the `Snake.crawl` method, check if `hasCoin` is true. If it is, don't remove the last tuple from the snake's body, but instead change `hasCoin` back to false. Create methods for setting and getting `hasCoin` as well as updating the snake in the `Board` instance."

[run it; remember to close the “run” window after each run]

As you can see, the view is quite similar to a git diff view. We can see the changes, and although in this case the change is limited to the opened file, it is also possible that AI Assistant changed other files, and we can click through them. If we like what we see, we can accept it; if not, we can revert the changes.

Similarly, AI Assistant also offers to edit your code directly. We can change the mode to "Edit," and... let's say we want to add one more thing to the crawl method: a check to see if the snake collided with itself. Let's write:

"Write a method that will check if the head of the snake is not in the same place as one of its other body tuples. Use it in the main game loop. If yes, the game should end."

[run it; remember to close the “run” window after each run]

As you can see, AI Assistant first explained what it would do and only then added the code. We can now close the chat window and examine the result. The view is what we have seen already after using the in-editor text field, only this time the changes affect two files. Let's accept them.

Generate git commit messages

Additionally, if we use the git plugin, we can now commit the change and use AI Assistant to produce the commit message. Let's do it now. LibGDX produces many files when it compiles the code. Let's find Models.scala and Main.scala... ok, and let's click this little button here... done. [commit]

To learn more about AI Assistant, you can open the AI Widget and choose "Discover Features." There, you will find detailed descriptions of the features we have just talked about and many more.

Quotas

Before we move from AI Assistant to Junie, there is one last thing I would like to discuss: quotas. Using AI is expensive. Each prompt you send to one of those models triggers a lot of computations, which require significant CPU and GPU power, consuming a lot of electricity. It seems we have more or less hit the ceiling on how big a large language model can get before growing it further stops making sense. Now, more research goes into optimization, that is, making smaller models achieve similar results to larger ones. That might be a way to save you some money, because you can do more with the same amount of tokens. At JetBrains, we are also working on this.

Mellum

Mellum is an open-source Large Language Model developed at JetBrains. It's a focal model, meaning it is designed for a specific purpose – coding – rather than being generic like OpenAI's GPT models. The focus is on creating a smaller, optimized model that significantly reduces latency, offering developers suggestions almost instantly. This makes Mellum more efficient than many generic models currently available on the market. On HuggingFace and Ollama, you can find the base model as well as several fine-tuned versions. These fine-tuned versions power the AI code completions in IntelliJ IDEA, allowing us to offer AI code completions and Next Edit Suggestions for free. They don't use your quota.

What affects your quota is the use of external models. Unfortunately, I can't specify exactly how much. This is something our AI team is working on: they make estimations and try to come up with solutions to minimize token usage, among other things.

Junie

By the way, the game of Snake that I used to show you how we can use AI in IntelliJ IDEA, actually I wrote it in one day with the help of Junie. While my prior experience with LibGDX was important, it’s clear that Junie was super-valuable.

Junie is an AI agent and a plugin available for IntelliJ IDEA and other JetBrains products. You can install it from **Settings | Plugins**, and then you will see a new button on the right panel. After clicking it, a chat interface appears.

While AI Assistant serves as your assistant in writing code—executing precise prompts and helping to understand existing code—the scope of Junie is much broader. With Junie, every request you make is first analyzed, rephrased more precisely in the form of a bullet-point list, and then each item on the list is executed. If possible, the results are tested. For example, if you ask Junie to modify an implementation of a method that is already covered by unit tests, Junie will find them, modify them along with the original method, and run the tests after the modification to ensure that it works as expected. You can even mark the "Think More" checkbox to instruct Junie to be especially careful. The "Brave Mode" allows Junie to execute commands without requiring your approval for each one. While staying vigilant and accepting commands that Junie wants to execute might be a good idea for tasks like changing the folder structure, it really doesn’t make sense to click "OK" every single time Junie wants to modify a file during coding tasks.

Junie sends many requests to the server and waits for answers, so it would be impractical to show you how it works in real-time as that would take too long. Instead, I pre-recorded a video and edited it to make it shorter.

[video]

- 00:00 - GDX-Liftoff
- 00:21 - game template
- 00:34 - "Generate a simple LibGDX game in Scala 3 with an infinite game loop and one which displays an isometric 8x8 board and a Scala icon on one of the tiles. Center the board on the screen. "
- 00:50 - "Move the board to the right in the game window so that the center of the board is in the center of the window."
- 00:59 - "The board is made of tiles. Right now every tile is black and their borders are white. Change the code so that each tile is white and has a thin black border."
- 01:10 - "In the implementation of renderScalaIcon, wrap the Texture into Sprite and set its size to the size of the square."
- 01:26 - "Create a Snake case class which consists of a list of positions (Pos2D) on the board and the direction in which the snake is moving (Dir2D). Create a Board class which consists of the board size and a reference to the snake. Put both Snake and Board in the file Models.scala"
- 01:37 - "Add to Draw.drawBoard the logic for displaying a snake - the snake is displayed simply as green tiles, the head being dark

green, the rest of the tiles standard GREEN. The borders of the tiles should still be visible."

- 01:48 - "Create a subfolder for unit tests for the project. The unit tests will be written in Scala and use mUnit as its test framework - add mUnit to the dependencies. Generate unit tests for the methods in Models.scala. Put them in a file called ModelsSuite.scala. Run them."
- 02:11 - "Create new methods in the Snake class: changeDirection, that takes a SnakeDir and returns a new Snake instance with the updated snakeDir. And crawl, that removes the last position from the snake's body and adds a new one at the head. The new head is the old head updated by the (x, y) coordinates of snakeDir."
- 02:25 - "Rewrite the Main class so that the game is run in an infinite game loop. Inside the game loop, the program calls Draw.drawBoard(board), checks if the user hits any key, waits 1s for that to happen, and if it doesn't, then the program calls board.update, and goes back to the beginning. If the user hit any key, stop the game."
- 02:38 - "Rewrite the `Snake.crawl` method. The new version should use board.size to ensure that the snake will never crawl outside of the board. If the computations make the new head have x or y equal -1 or equal board.size, instead it should wrap over the board: the x or y that is -1 becomes board.size-1, and the x or y that is board.size becomes 0. Add the parameter `board: Board` to the method `Snake.crawl` and update the call to it in Main.scala. Update unit tests and write new ones."

- 03:13 - "Call from here a new method that will check what key was hit. If it was one of the arrow keys: Left, Right, Up, or Down, update the snake on the board, changing its snakeDir respectively to SnakeDir.Left, Right, Up, and Down. Create methods that will allow you to make such an update."
- 03:39 - "Forbid to change direction backwards (Up to Down, Left to Right, and vice versa). Change the result type of the method updateSnakeDirection to Boolean and return true if changing the direction succeeded, false otherwise."
- 03:52 - "Create a new immutable field `hasCoin: Boolean` in the Snake class, and set it by default to false. In the Board.update method, if the snake's head is on the same position as a coin, update the snake in the board and change `setCoin` to true. In the Snake.crawl method, check if `hasCoin` is true. If it is, don't remove the last tuple from the snake's body, but instead change `hasCoin` back to false. Create methods for setting and geeting `hasCoin` as well as updating the snake in the Board instance."
- 04:24 - "Add a method to the Board class to get the list of positions of all empty tiles - ones without the snake or coins. Make a method in the Main class that will take that list and randomly choose one of them. Create a field `newCoinInterval: Float` that will describe how often a new coin should appear on the board. By default it should be 10s. Add to the game loop the logic that updates the board by adding a new coin on one of the empty tiles every

newCoinInterval, but only if `board.coinsNumber` is less than MAX_COINS. Write new unit tests."

How to Use AI Assistance in Coding

I would like to finish this talk with a few words of advice.

AI agents, AI code completions, and large language models in general are just tools, and they are most productive when used as such. They don't act by themselves; they only react to the input. They are not creative; they only respond with output that best fits the input, according to the way they are built and trained. And they make mistakes, also known as hallucinations, simply because if they need to compose a few memorized examples to produce an output, sometimes the middle ground between those good answers becomes a bad answer.

Be careful with prompts. Try to make them as precise and detailed as possible. If they are complicated, split them into a few simpler prompts and give them one by one. Actually, thanks to this, AI Assistant can help you in a way we've known for decades: as a rubber duck that you talk to, and by doing the talking, you yourself find an answer to your problem.

Check the results often. When an AI agent produces or modifies your code, review those results as if you were reviewing a PR from an external contributor. Read and understand the code. If something is unclear, ask the AI agent to explain and/or rewrite it in a simpler way. Again, this interaction may quite often lead to finding a solution better than the one the AI agent initially proposed, and it will surely help you weed out bugs that the AI agent could have introduced if you weren't paying attention.

Specifically, avoid letting AI generate tests for AI-generated code or your own code if you are not 100% sure there are no bugs. If there is a bug and you give that code to an AI agent to generate tests, it will generate tests that pass when the bug is present but fail when someone finally finds and removes it.

This is a good case for Test-Driven Development (TDD). "But Maciek, how can we be sure there are no bugs if we haven't run tests yet?" Good question. Consider writing the tests first. This is actually where an AI agent can help you a lot. Explain to it what you are working on and ask it to generate tests for production code that does not exist yet. This way, you will avoid the bias of writing tests that accommodate bugs, and you may actually have a very valuable conversation with the AI agent – and with yourself – about what it is that you actually want to code. Only then, when you know what you're doing and the tests are ready, can you switch to writing production code – again, with the use of AI. And if for all that you choose IntelliJ IDEA, AI Assistant, and Junie, we will be delighted and happy to help you.

Thank you.

Private notes

* I have "automatic completion on typing" enabled at the beginning -- let's mention that.

* Use Alt+Enter to explain the block of code instead of the chat window

* Say that you wrote that code a long time ago and you need AI Assistant to explain it to you

* FLCC and Mellum are also about latency. We can use it to get a few alternative completions and ask for their values, and choose one. (Paul talks about it in one of the videos).

* Keep Next Edit Suggestions, even talk about it a bit more

* Talk with N.E.S. team how to talk about it.

* Project Guidelines for Junie - read about it.

* ... can I skip MCP?... -- yes, let's skip it

* Read the blog post about Junie quota

* AI code completions and NES don't use quota -- tell that, even repeat a few times. What you can experience is throttling if you use them too much. (Good argument not to have them enabled all the time, but only under a shortcut).

* Unfortunately, rules about the quota may change.

* Change the code style to braces, and named tuples to case classes or dedicated position/direction classes.

* Hide the Run window or minimize it.

* Add a personal note about local code completion ("... What if you're on a plane")

* Add a few more words about what are the differences between completion policies: creative vs focused.

* Can I come up with an example for N.E.S.? Maybe refactoring? Cleaning up? Something that doesn't change the behaviour of the game? -- make the case that NES is good for refactoring.

* Introduce Mellum by asking why is it that code completion don't burn quota. - because we use our own LLM, Mellum.

* Don't say that we don't have a version of Mellum for Scala.

* Make a better transition between AI Assistant to Junie. -- say that the game itself is written with Junie.

* Cut out the part about converting Java LibGDX code to Scala

* In the tips, change "LLM" to "AI tools" or "AI agents", "AI chats". Maybe.

*