

Jdb (Java Debugger) Guide

[Introduction](#)

[Starting jdb](#)

[Breakpoints](#)

[Inspection](#)

[Threads](#)

Introduction

[jdb](#) is a useful tool for debugging Java programs, somewhat similar to gdb. This guide focuses on using it on the command line to solve problems with deadlocks.

Disclaimer: The course staff just started using it during TA camp to fix bugs in past solution code. It takes more familiarity with this tool than we have to know its full capabilities, and a lot of prior experience showing it to others in order to reason about what newcomers might need the most help with. So if you already knew some nice features not mentioned here, or are confused by the guide, please let us know! We will update this based on student feedback.

Starting jdb

To start debugging a program in jdb right off the bat, you may run it as follows to debug your MainClass:

```
jdb MainClass <args>
```

This will start jdb and halt Java before the first instruction of your program is reached. When you have configured your breakpoints (see “Breakpoints” below), type run to start your program.

To start up a Java program and connect to it using another terminal window, run your Java program in a way similar to this:

```
java -agentlib:jdwp=transport=dt_socket,address=<port number>,server=y,suspend=n <other flags> <MainClass> <args>
```

This will start the program and it will run as normal, but allow external connections from another jdb client. From another terminal window, type this:

```
jdb -attach <port number>
```

jdb will then be attached to your running Java code. At this point, you can type `suspend` to suspend all your threads and then use other commands.

Breakpoints

Inserting breakpoints can be done in 4 ways, per the guide linked earlier:

- `stop at MyClass:22` (sets a breakpoint at the first instruction for line 22 of the source file containing `MyClass`)
- `stop in java.lang.String.length` (sets a breakpoint at the beginning of the method `java.lang.String.length`)
- `stop in MyClass.<init>` ("`<init>`" identifies the `MyClass` constructor)
- `stop in MyClass.<clinit>` ("`<clinit>`" identifies the static initialization code for `MyClass`)

Breakpoints work the same way as in `gdb` - when reached, program execution stops and can be resumed.

Inspection

You may use `jdb` to inspect class variables within whatever stack frame you happen to be looking at at the time. Note that **`jdb` does not allow you to inspect local variables by default**. If you compile your Java code with the `-g` flag, however, you will be able to.

The following commands are useful for inspecting program state:

- `print <expression>`: Evaluate and print the result of an expression that does not return `void`. The expression can be arbitrary and may include method invocations.
- `dump <object>`: Dump an object, printing each of its fields individually (but not recursively.) You may supply a primitive as well. The result in that case is the same as if you had printed it.
- `where`: Print a backtrace for your program.
- `up/down <number>`: Switch the current stack frame you are inspecting.

Threads

In multithreaded applications, you need to select a thread to debug.

In order to get a list of threads, type `threads`. You will get a result something like this:

```

Group system:
  (java.lang.ref.Reference$ReferenceHandler)0x153 Reference Handler cond. waiting
  (java.lang.ref.Finalizer$FinalizerThread)0x152 Finalizer cond. waiting
  (java.lang.Thread)0x151 Signal Dispatcher running
Group main:
  (java.lang.Thread)0x1 main cond. waiting
  (java.lang.Thread)0x1af Thread-0 cond. waiting
  (java.lang.Thread)0x1b0 Thread-1 cond. waiting
  (java.lang.Thread)0x1b1 Thread-2 cond. waiting
  (java.lang.Thread)0x1b3 Thread-3 cond. waiting
  (java.lang.Thread)0x1b6 Thread-4 cond. waiting
  (java.lang.Thread)0x1b5 Thread-5 cond. waiting
  (java.lang.Thread)0x1b7 Thread-6 cond. waiting
  (java.lang.Thread)0x1b8 Thread-7 cond. waiting
  (java.lang.Thread)0x1ba Thread-8 cond. waiting
  (java.lang.Thread)0x1b9 Thread-9 cond. waiting

```

The hexadecimal numbers (0x*) are the thread IDs. Switching between threads can be done with the thread <id> command. To select thread 4 in this case, type thread 0x1b6. You may now use the above commands to inspect its state.