

AUTOMATION TESTING LAB

Course Code: UE17MC554

Lecture Hours (IC): 13

Lab Hours: 26

Credit: 1C

Faculty: Ms. Deepthi S Narayan(DSN)

Assistant Professor

Dept of CA

PES University

Course Walk-through

Introduction to Debugging

Debugging Concepts

Debugging C, C++ and Java Programs

Introduction to Testing

Different Types of testing

Why testing is required

Introduction to test cases

Course Walk-through

Differentiating between Manual and Automation
Testing

Testing tools

Introduction to Selenium

Equivalence Class Testing

Boundary-value Testing

Decision-based Testing

Course Walk-through

Data-flow Testing

Path Testing

WebPage Testing using Selenium

Selenium Locators

Selenium with decision and Loops

Debugging



Six Stages of Debugging

1. That can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?

Introduction to Debugging

- A debugger is a program that simulates/runs another program and allows you to:
 - Pause and continue its execution
 - Set “breakpoints” or conditions where the execution pauses so you can look at its state
 - View and “watch” variable values
 - Step through the program line-by-line

GNU Debugger helps you in getting information about the following:

If a core dump happened, then what statement or expression did the program crash on?

If an error occurs while executing a function, what line of the program contains the call to that function, and what are the parameters?

What are the values of program variables at a particular point during execution of the program?

What is the result of a particular expression in a program?

How debugging can make you a better developer

The nature of bugs

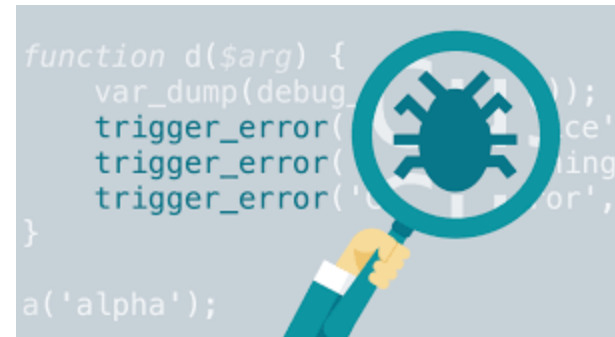
Fixing bugs is how we learn

How do we get better at learning things?

Some experience required

Try things first

Use Google as a last resort

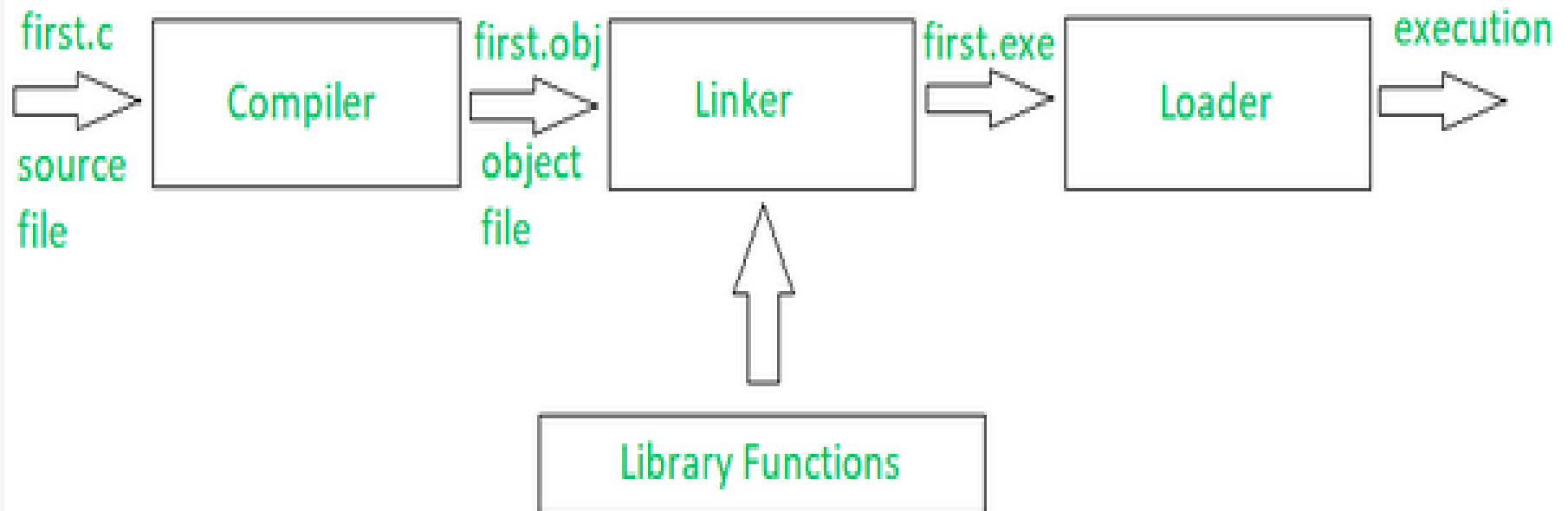


Points to Note

- ❑ Even though GDB can help you in finding out memory leakage related bugs, but it is not a tool to detect memory leakages.
- ❑ GDB cannot be used for programs that compile with errors and it does not help in fixing those errors.



Some Basics of C pgm execution



Pre-requistes

- ❑ Install gcc on Ubuntu16.04 (ANSI-complaint C compiler)
- ❑ Approximately 150MB of free disk space.

GDB installation

Before you go for installation, check if you already have gdb installed on your Unix system by issuing the following command:

```
$gdb -help
```

```
deepthi@deepthi:~$ gdb -help
This is the GNU debugger. Usage:

gdb [options] [executable-file [core-file or process-id]]
gdb [options] --args executable-file [inferior-arguments ...]

Selection of debuggee and its files:

--args           Arguments after executable-file are passed to inferior
--core=COREFILE  Analyze the core dump COREFILE.
--exec=EXECFILE  Use EXECFILE as the executable.
--pid=PID        Attach to running process PID.
--directory=DIR  Search for source files in DIR.
--se=FILE        Use FILE as symbol file and executable file.
--symbols=SYMFILE Read symbols from SYMFILE.
--readnow        Fully read symbol files on first access.
--write          Set writing into executable and core files.

Initial commands and command files:

--command=FILE, -x Execute GDB commands from FILE.
```

GDB installation

- ❑ Use the following command to install gdb on linux machine.
- ❑

```
$ sudo apt-get install libc6-dbg gdb valgrind
```
- ❑ You can install gdb on Debian-based linux distro (e.g. Ubuntu, Mint, etc) by following command.

```
$ sudo apt-get update  
$ sudo apt-get install gdb
```

GDB can be used in two ways:

- 1) To Debug running program having logical error, crashing or hanging.
- 2) To Debug coredump of program generated automatically when program crashes.

Stepping

- Stepping is a debugger feature that lets you execute (step through) your code line by line. This allows you to examine each line of code in isolation to determine whether it is behaving as intended.
- There are actually 3 different stepping commands: step into, step over, and step out.

Breakpoints

- A breakpoint is a special marker that tells the debugger to stop execution of the program at the breakpoint when running in debug mode.

Debug the following programs 1

- 1) `simpledebug.c`
- 2) `test.c`
- 3) `foo.c`
- 4) `Program1.c`
- 5) `Progam2.c`

Potential runtime errors in C

Overflow Numeric calculations which produce a result too large to represent.

Divide by Zero Dividing a numeric value by zero.

Invalid Shift Shifting an integer value by an amount which produces an undefined result according to the C standard.

Memory Errors Accessing an invalid memory region in a way which produces an undefined result, such as accessing an array outside its bounds or accessing heap-allocated memory after the memory has been freed.

Potential runtime errors in C

Uninitialized Data Access Accessing memory before the memory has been initialized, so that the result of the access is undefined under C semantics.

Debugging Tips

Examine the most recent change

Debug it now, not later

Read before typing

Make the bug reproducible

Display output to localize your search

Write a log file

Use tools

Keep records

Getting Started: Starting and Stopping

Compiles myprogram.c with the debugging option (-g)

```
gcc -g myprogram.c -o myprogram
```

Execute with,
./myprogram

Launching gdb,
gdb myprogram

gdb commands

b main - Puts a breakpoint at the beginning of the program

b - Puts a breakpoint at the current line

b N - Puts a breakpoint at line N

b fn - Puts a breakpoint at the beginning of function "fn"

d N - Deletes breakpoint number N

info break - list breakpoints

gdb commands

- **r** - Runs the program until a breakpoint or error
- **c** - Continues running the program until the next breakpoint or error
- **f** - Runs until the current function is finished
- **s** - Runs the next line of the program
- **n** - Like s, but it does not step into functions
- **p var** - Prints the current value of the variable "var"

`gdb` commands

- **`bt`** - backtrack (Prints a stack trace)
- **`u`** - Goes up a level in the stack
- **`d`** - Goes down a level in the stack
- **`l` or `list`** - Lists the source file
- **`l n`** – five lines above and 5 lines below `n` from source file are listed
- **`info locals`** – Gives all local variables information
- **`cntrl + L`** – clears the `gdb` history
- **`q`** - Quits `gdb`

Conditional breakpoints

gdb allows the definition of conditional breakpoints.

There are two ways to specify conditional breakpoints.

These points are defined by first creating a regular breakpoint, and then attaching a condition. The execution is suspended if the breakpoint is reached and the condition satisfied. If the condition is not satisfied, the breakpoint has no effect.

Syntax and example 1

Syntax:

(gdb) break line/function condition

Example -

b 8 if(a > 73)

Syntax and example 2

Syntax:

(gdb) b line/function

(gdb) condition breakpointno. actual_condition

Example

```
(gdb) b 29
Breakpoint 4 at 0x804844f: file gdb_use.c, line 29.
(gdb) condition 4 y == 999
(gdb)
```

Setting the values during debugging

set variablename = expression

Example - (gdb) set l1=10

(gdb) p l1

\$2 = 10

What are coredumps?

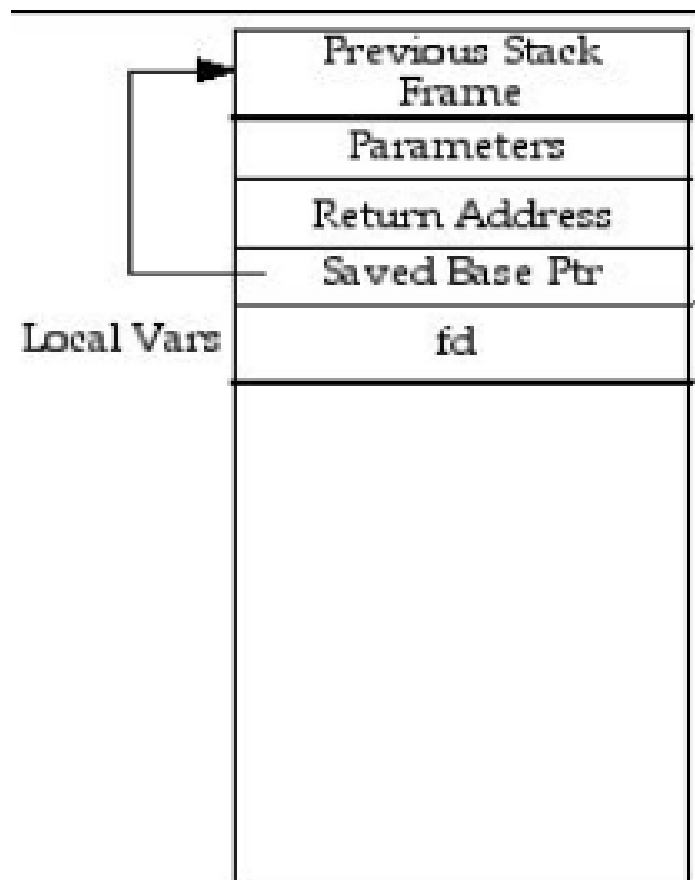
- A core dump occurs when a program is abnormally terminated because it tried to do something illegal.
- When such an interrupt occurs, and an operating system like Linux takes the stack/context of that process, and dumps it into a file.

What is backtracking or back tracing?

- When you have a core file loaded, you can check the status of your application when the core was dumped. This is called a back trace.
- In gdb , you can issue backtrace or bt.
A **backtrace** is a summary of how your program got where it is. It shows one line per frame, for many frames, starting with the currently executing frame (frame zero), followed by its caller (frame one), and on up the stack.

What is a stack frame?

The **stack frame**, also known as activation record is the collection of all data on the stack associated with one subprogram call.



Debug the following programs - 2

- program2.c (from last week)
- factorial.c
- hello.c
- buggy.c
- program3.c – funcs.c (bt)
- gdbtest3.c

Command History (Try it!)

- Inside gdb do this as the first thing -
- set history filename fname
- set history save
- set history save on
- set history save off
- NOTE: Do not set history save to off

Debugging C++ Programs Using "gdb"

- Compile a C++ program with following command -
- `g++ -g myprogram.cpp -o myprogram`
- Normal execution
- `./myprogram`
- Debugging through GNU Debugger
- `gdb myprogram`

Debug the following programs - 3

1. app.cpp (discussed in the class)
2. demo.cpp
3. pplpie.cpp
4. cinget.cpp
5. degree.cpp
6. strlenlength.cpp

IDE's



What is an IDE?

An integrated development environment (IDE) is a software suite that consolidates basic tools required to write and test software.

IDE's that are best suited for code management, highlighting, debugging facilities.

- QtCreator
- Kdevelop
- Eclipse

Common Features of IDE

- An IDE typically contains a code editor, a compiler or interpreter, and a debugger, accessed through a single graphical user interface (GUI).
- The user writes and edits source code in the code editor.
- The compiler translates the source code into a readable language that is executable for a computer.
- And the debugger tests the software to solve any issues or bugs.

Popular IDE's

C-Free
Cloud9
Visual Studio
NetBeans
Eclipse
IntelliJ IDEA



Debugging Java Program in Eclipse

- Eclipse is an integrated development environment (IDE) for Java and other programming languages like C, C++, PHP, and Ruby etc.
- You can download eclipse from <http://www.eclipse.org/downloads/> [OR] <https://www.eclipse.org/downloads/packages/release/kepler/sr1/eclipse-ide-java-developers>
- Java developers typically use Eclipse Classic or Eclipse IDE for developing Java applications.

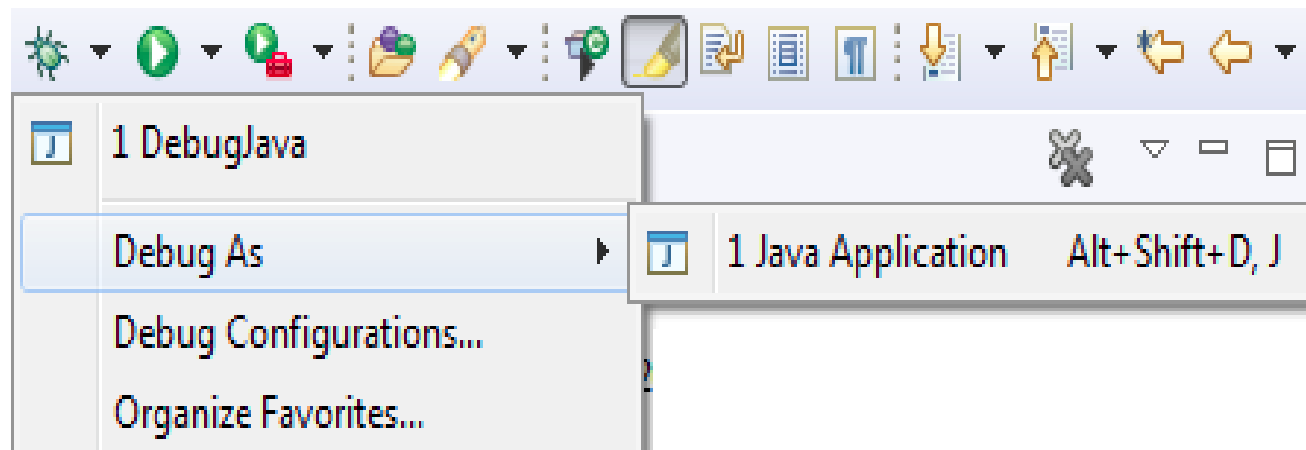
Eclipse – First Steps

- Understanding these primary concepts wrt Eclipse environment is important
- Workspace
- Project Creation
- Perspective
- Views

Debugging Java Program in Eclipse

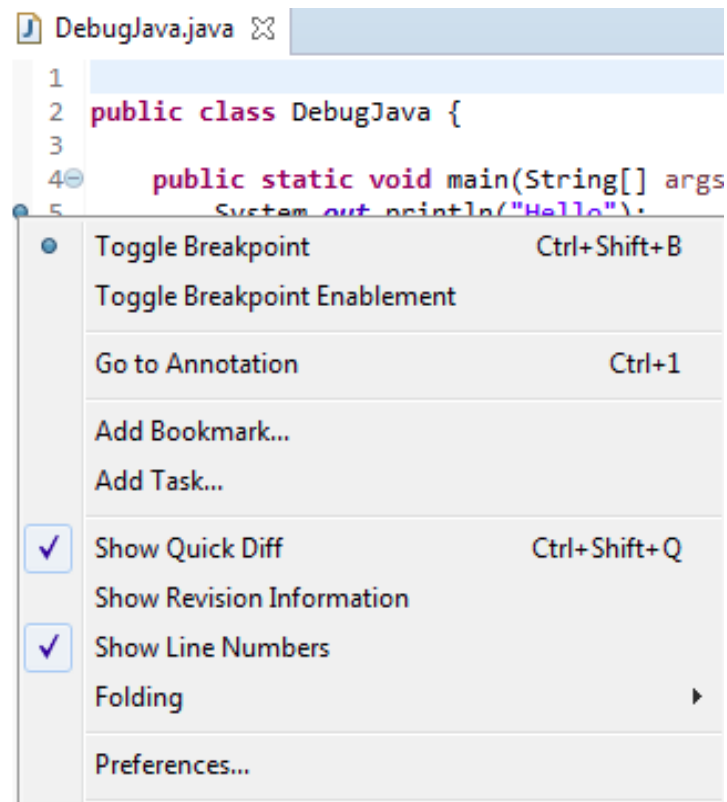
1. Launching and Debugging a Java program

A Java program can be debugged simply by right clicking on the Java editor class file from **Package** explorer. Select **Debug As** → **Java Application** or use the shortcut **Alt + Shift + D, J** instead.



2. Breakpoints

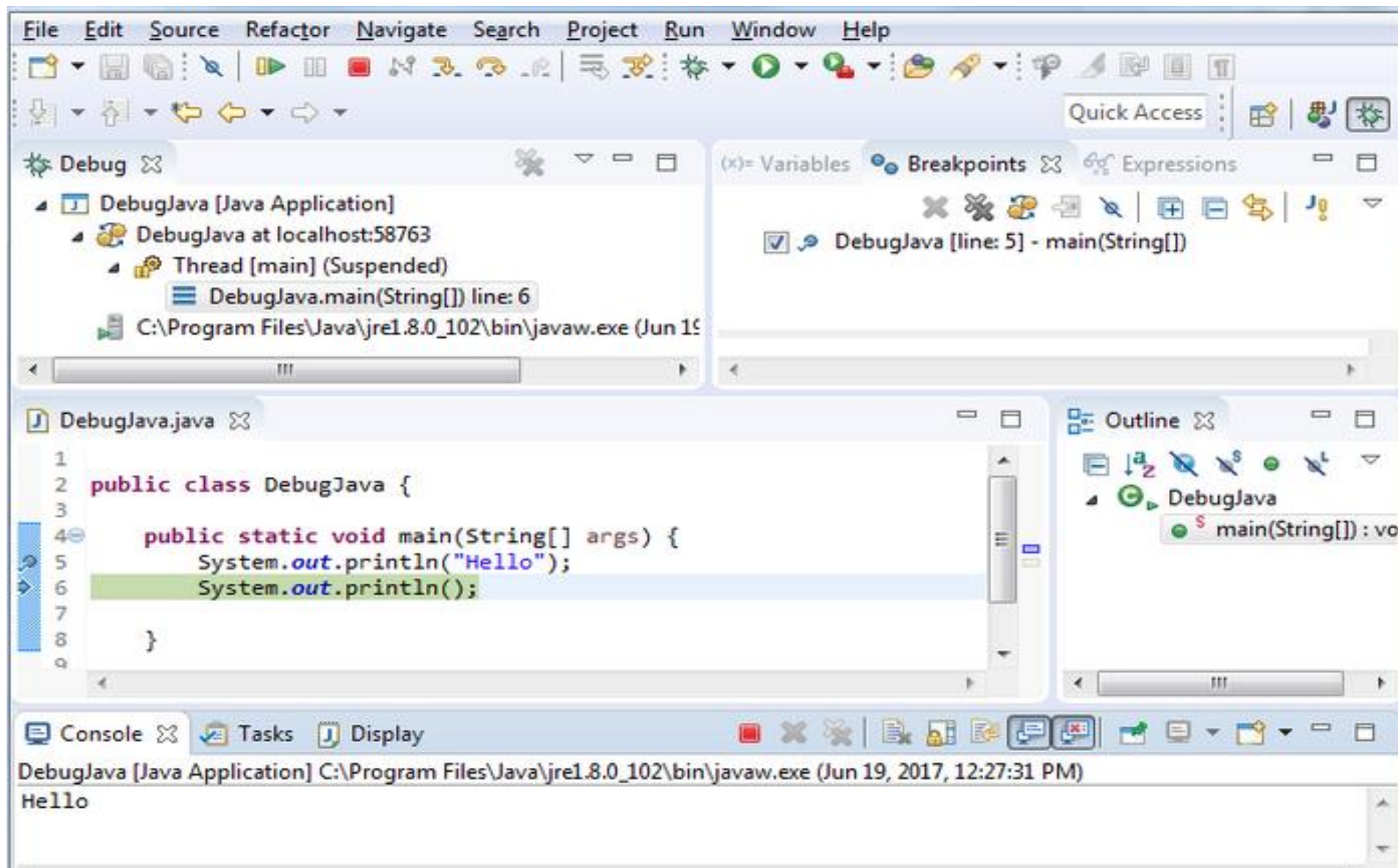
To define a breakpoint in your source code, right-click in the left margin in the Java editor and select Toggle Breakpoint. Alternatively, you can double-click on this position.



3. Debug Perspective

When a Java program is started in the debug mode, users are prompted to switch to the debug perspective.





- Debug view – Visualizes call stack and provides operations on that.
- Breakpoints view – Shows all the breakpoints.
- Variables/Expression view – Shows the declared variables and their values. Press **Ctrl+Shift+d** or **Ctrl+Shift+i** on a selected variable or expression to show its value. You can also add a permanent watch on an expression/variable that will then be shown in the Expressions view when debugging is on.
- Display view – Allows to Inspect the value of a variable, expression or selected text during debugging.
- Console view – Program output is shown here.



4. Stepping commands

The Eclipse Platform helps developers debug by providing buttons in the toolbar and key binding shortcuts to control program execution.



Shortcut	Toolbar	Description
F5 (Step Into)		Steps into the call
F6 (Step Over)		Steps over the call
F7 (Step Return)		Steps out to the caller
F8 (Resume)		Resumes the execution

- Debugging steps and commands can be found in detail @
- <https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Ftasks%2Ftask-stepping.htm>

Common Runtime Errors in Java

- ❖ NullPointerException
- ❖ IndexOutOfBoundsException
- ❖ ClassCastException
- ❖ ArithmeticException
- ❖ NumberFormatException
- ❖ StackOverflowException

Debug the following programs – 4

- 1) Prime.java
- 2) Messageme.java
- 3) AException.java
- 4) AIOB.java
- 5) Temp.java
- 6) If1.java
- 7) If2.java
- 8) NFE.java

Setting the variable values

- ❖ The user can change the value using steps –
- ❖ Right click on the variable add it to 'watch' and then while debugging go to 'Expressions View'. A user can change the value from Expression View directly.

Setting the variable values

- ❖ Click on Window -> Open Perspective -> Debug
- ❖ Click on Tab Variables.
- ❖ Right-click the variable for which you want to change the value and click on Change Value...
- ❖ Set the Value as inBoolean. TRUE the dialogue and click Ok.

Debug the following programs - 5

1. Equals.java – understand the difference
2. Switch2.java – Recognise the problem before debugging
3. Loops_Reverse.java – Debug and fix
4. SimpleExampleSO.java and Car.java - – Debug and fix
5. Area.java – Provide a fix though there is no bug
6. Trial.java – Debug and fix
7. BSE.java – Debug and fix