TASK

Debugging

Visit our website

Introduction

Welcome to the Debugging Task!

Debugging is essential to any software developer! In this task, you will learn about some of the most commonly encountered errors. You will also learn to use your IDE to more effectively debug your code.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

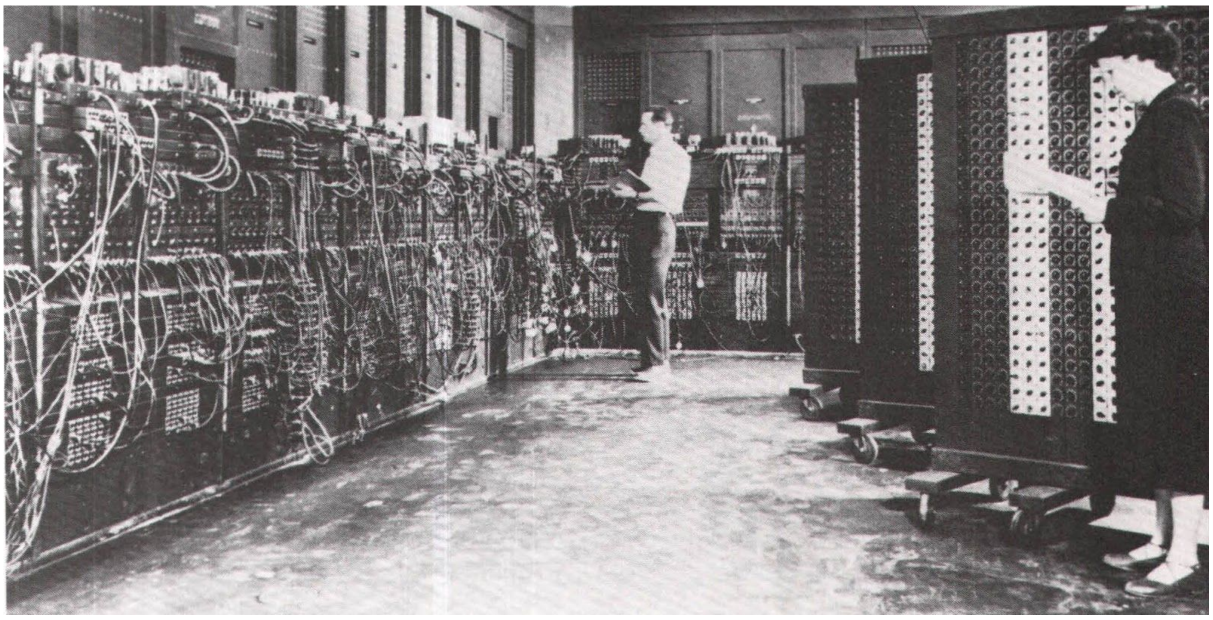
Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

What is Debugging?

Debugging is something that you, as a software engineer, will come to live, breathe and sleep. Debugging is when a programmer looks through their code to try and find a problem that is causing some error.

The word 'debugging' comes from the first computers — that is, when a computer was roughly the size of an entire room. It was called debugging because bugs (as in, living insects) would get into the computer and cause the device to function incorrectly. The programmers would need to go in and remove the insects, hence — debugging.

One such computer is the ENIAC. It was the first true electronic computer and was located in the University of Pennsylvania.



An image of the ENIAC.

Notice how today the computer we know is significantly smaller and also much more powerful than older computers like the ENIAC (despite their enormous size). It goes to show how, in just a few years, we can expand our computational power immensely. We owe that to the ever-changing and growing world of technology. So even though sometimes keeping up with the trends may be tiring, keep in mind that it's for the best!

You can do some more research about the ENIAC if you want on the University of Pennsylvania's website: http:// www.seas.upenn.edu/about-seas/eniac/.

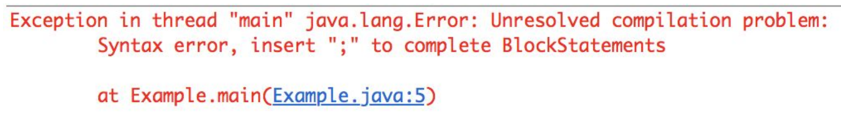
Errors

Errors are unfortunately a very large part of programming, so it’s important that we learn how to handle them and, even more importantly, how to fix them.

There are three categories that errors fit into:

- **Compile Errors**
  These are errors that are picked up by the Java compiler when you try to compile your code. They are related to the way your code is written (for example your spelling, syntax and capitalisation) and are relatively easy to fix.

  An example of what a compilation error looks like:

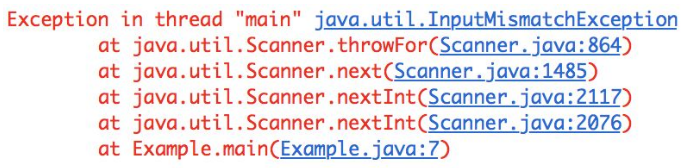
  ```
  Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  	Syntax error, insert ";" to complete BlockStatements

  	at Example.main(Example.java:5)
  ```

  You can now go to the line indicated by the error message and correct the error, then try to recompile your code.

- **Runtime Exceptions**
  These are slightly harder to fix. These errors are caused by something occurring in your program that your code is unable to handle. Because they are only caused by situations that depend on the state of your program, they cannot be picked up beforehand by Java and so it is up to you to avoid them.

  However, no programmer will always be able to avoid them forever, so Java provides some very valuable functionality when it hits an exception. It will return the type of exception, the line the exception happened on, and also show you the flow of the program up to the point that the error occurred.

  An example of what a runtime error looks like:

  ```
  Exception in thread "main" java.util.InputMismatchException
  	at java.util.Scanner.throwFor(Scanner.java:864)
  	at java.util.Scanner.next(Scanner.java:1485)
  	at java.util.Scanner.nextInt(Scanner.java:2117)
  	at java.util.Scanner.nextInt(Scanner.java:2076)
  	at Example.main(Example.java:7)
  ```

- **Logical Errors**
  These are errors in the logic your program is following. These are generally the hardest to solve, as they are not picked up by Java at all. You need to revise your code to try to see where you have gone wrong.

  Generally you can try follow through your code, keeping track of your different variables and other values on each line. However, for larger programs this is totally impractical, and so over time you will need to develop a sense of how to solve logical errors efficiently.

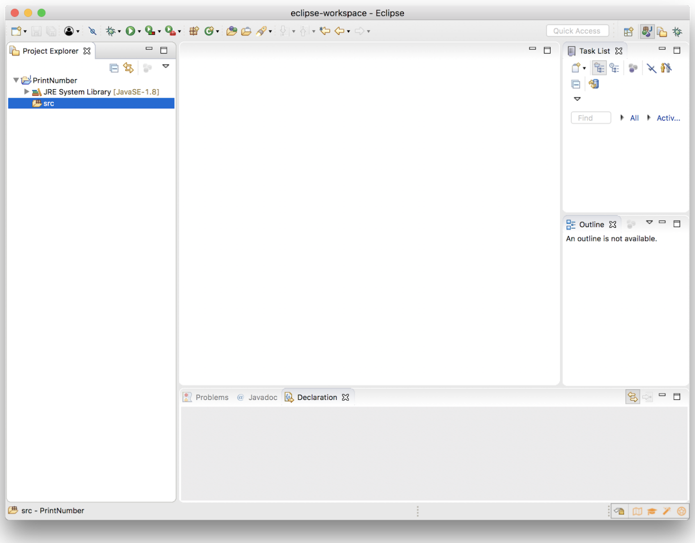
  This should generally happen naturally as your skills in programming grow. Each programmer has their own way of solving problems that best suits their manner of programming and way of thinking.

Debugging with Eclipse

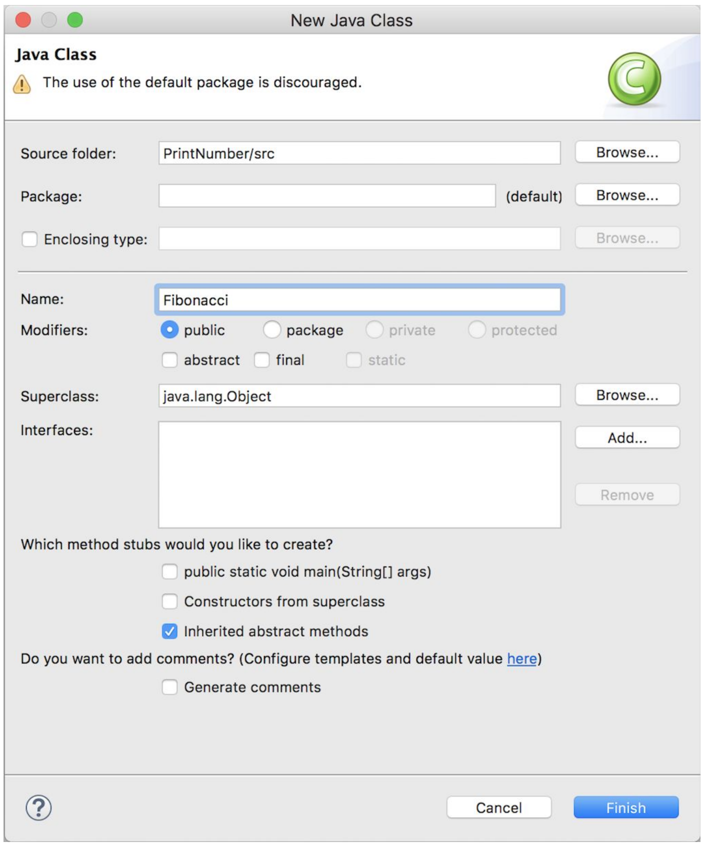
We will explain how to debug your Java projects using Eclipse quickly.

1. Firstly, start by creating a new Java project called *PrintNumber*.

2. Once your project has been created right-click on the *src* folder in the Package Explorer and select *New -> Class* from the menu, to create a new Java class.



3. Name your new class *Fibonacci* and then click the *Finish* button.



New Java Class

Java Class

The use of the default package is discouraged.

Source folder: PrintNumber/src

Package: (default)

Enclosing type:

Name: Fibonacci

Modifiers: [✓ public] package private protected

abstract final static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

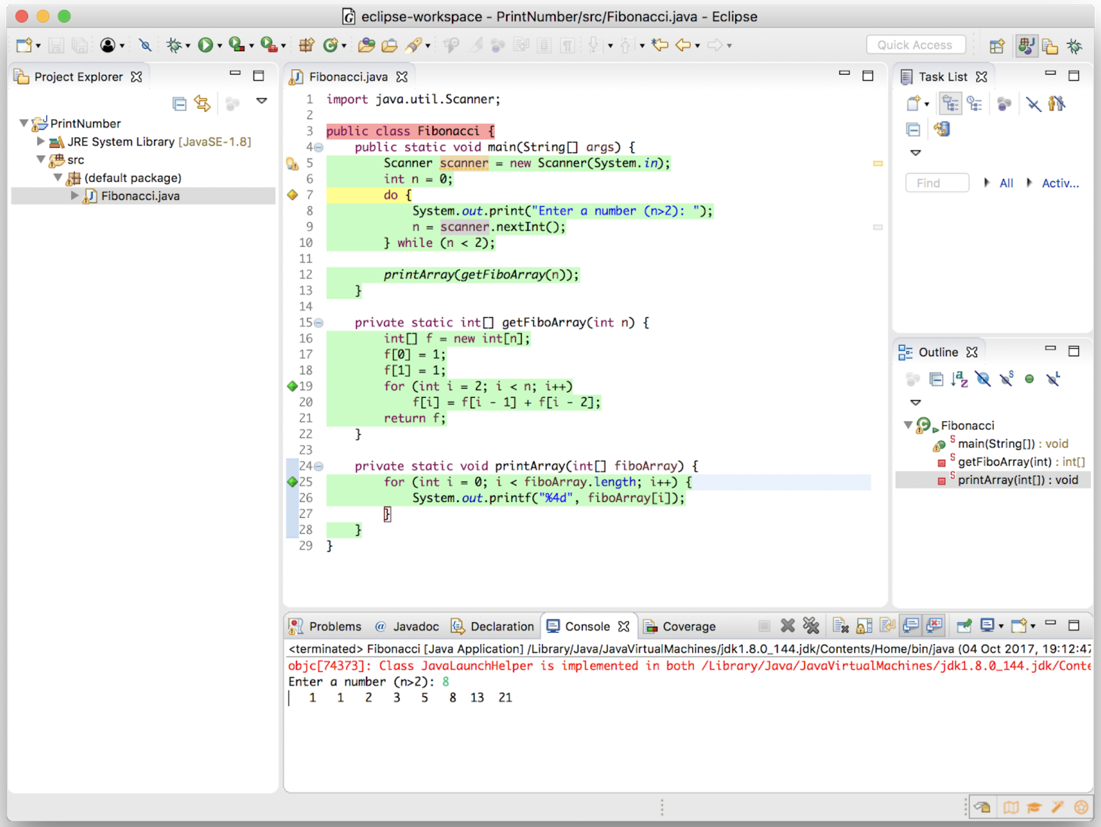
[✓ Inherited abstract methods]

Do you want to add comments? (Configure templates and default value here)

Generate comments

4. Copy the following code into your new *Fibonacci* class. The class should compile and run.

```
import java.util.Scanner;

public class Fibonacci {
      public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            int n = 0;
            do {
                  System.out.print("Enter a number (n>2): ");
                  n = scanner.nextInt();
            } while (n < 2);

            printArray(getFiboArray(n));
      }

      private static int[] getFiboArray(int n) {
            int[] f = new int[n];
            f[0] = 1;
            f[1] = 1;
            for (int i = 2; i < n; i++)
                f[i] = f[i - 1] + f[i - 2];
            return f;
        }

        private static void printArray(int[] fiboArray) {
            for (int i = 0; i < fiboArray.length; i++) {
                System.out.printf("%4d", fiboArray[i]);
            }
        }
}
```

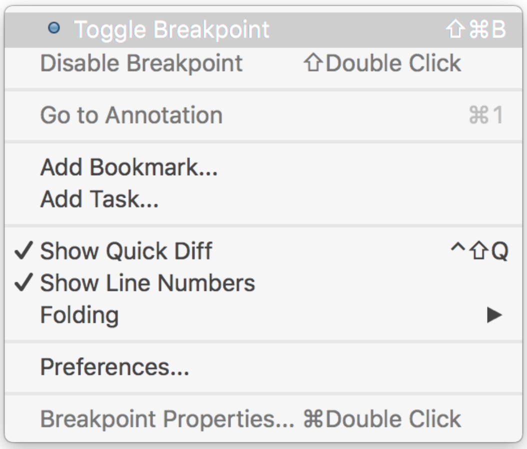


```
import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = 0;
        do {
            System.out.print("Enter a number (n>2): ");
            n = scanner.nextInt();
        } while (n < 2);

        printArray(getFiboArray(n));
    }

    private static int[] getFiboArray(int n) {
        int[] f = new int[n];
        f[0] = 1;
        f[1] = 1;
        for (int i = 2; i < n; i++)
            f[i] = f[i - 1] + f[i - 2];
        return f;
    }

    private static void printArray(int[] fiboArray) {
        for (int i = 0; i < fiboArray.length; i++) {
            System.out.printf("%4d", fiboArray[i]);
        }
    }
}
```

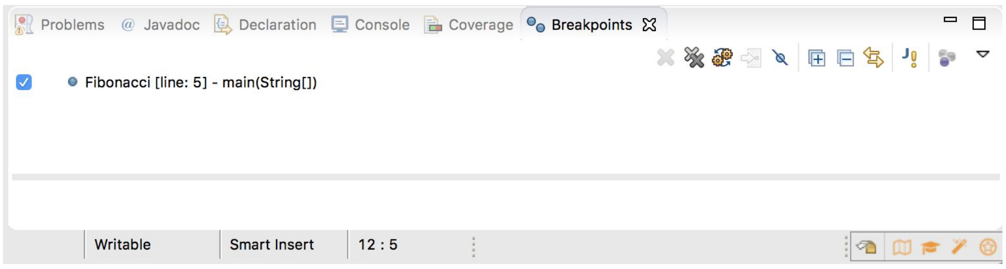
```
<terminated> Fibonacci [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (04 Oct 2017, 19:12:47
objc[74373]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Conte
Enter a number (n>2): 8
   1   1   2   3   5   8  13  21
```

5. We can now add a breakpoint to our main method. A breakpoint tells the debugger to suspend execution at a certain point in the code temporarily. You can do this to view information about a program's state.

   To create a breakpoint, right-click on the left margin in the Java editor and select *Toggle Breakpoint* or double-click in the position you would like to create the breakpoint. Create a breakpoint on the line where *n* is initialised. A blue dot will appear next to the line of code where the execution will halt.



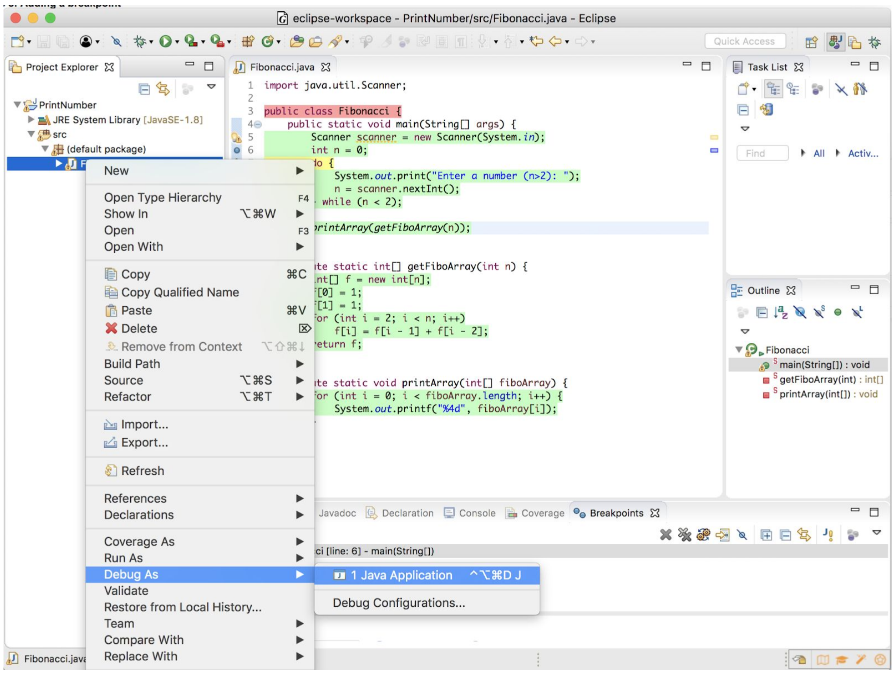
The Breakpoints view at the bottom of your screen allows you to delete and deactivate Breakpoints and modify their properties.



✓ Fibonacci [line: 5] - main(String[])

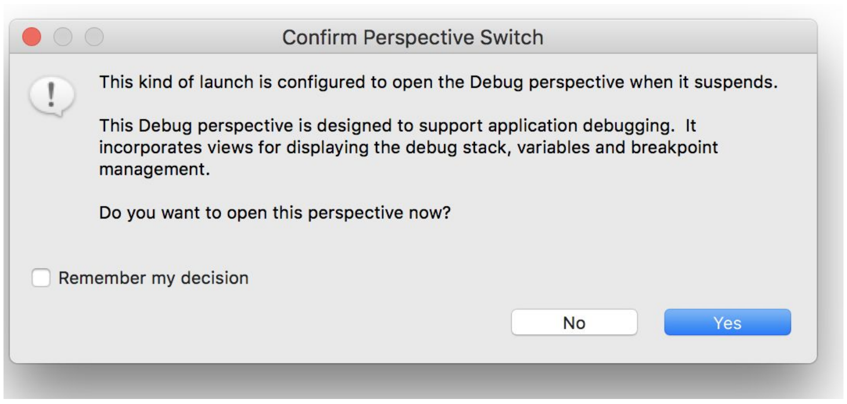
Writable | Smart Insert | 12 : 5

6. We can now run our program through the debugger. To do this, right-click on your *Fibonacci* file in the Package Explorer and then select *Debug As -> Java Application* from the menu.



```
import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = 0;
        do {
            System.out.print("Enter a number (n>2): ");
            n = scanner.nextInt();
        while (n < 2);

        printArray(getFiboArray(n));

    te static int[] getFiboArray(int n) {
     nt[] f = new int[n];
     [0] = 1;
     [1] = 1;
     or (int i = 2; i < n; i++)
            f[i] = f[i - 1] + f[i - 2];
     eturn f;

    te static void printArray(int[] fiboArray) {
     or (int i = 0; i < fiboArray.length; i++) {
            System.out.printf("%4d", fiboArray[i]);
```

If you get a message about switching perspectives, check "Remember my decision" so that you don't see this dialog again.



Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

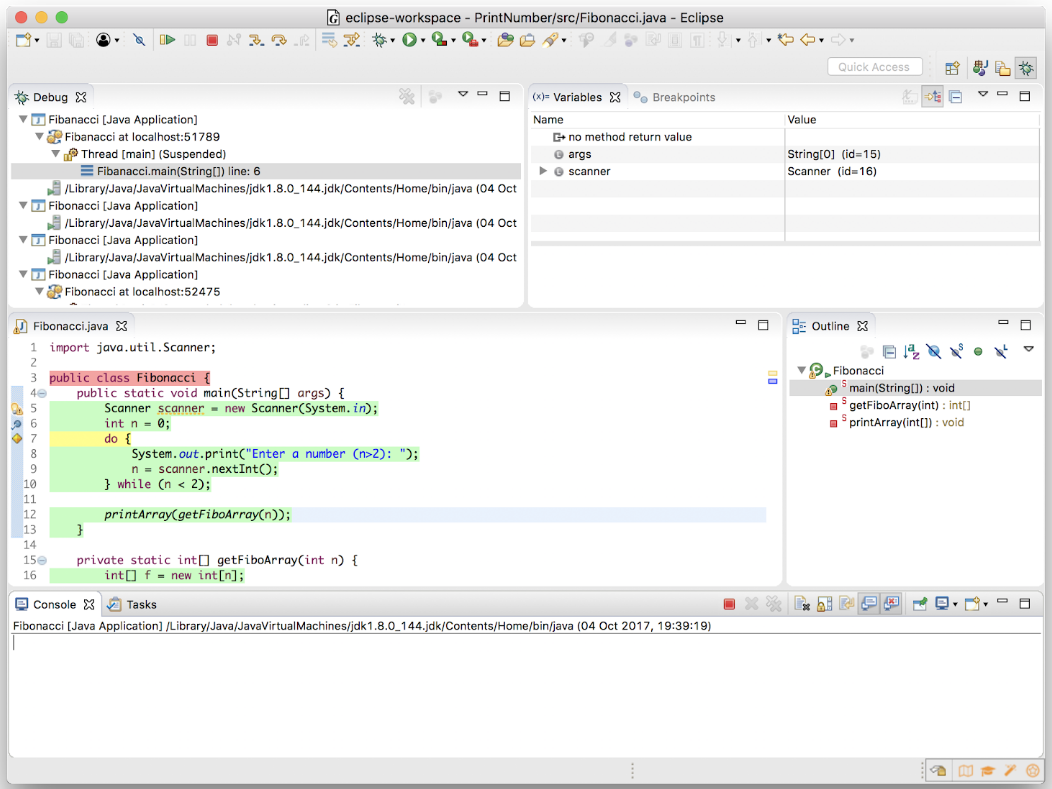
This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

☐ Remember my decision

No Yes

7. You should now see the debug perspective which might look similar to the screenshot below. The debug perspective offers additional views that can be used to troubleshoot an application.



```
Fibanacci [Java Application]
  Fibanacci at localhost:51789
    Thread [main] (Suspended)
      Fibanacci.main(String[]) line: 6
  /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (04 Oct
Fibonacci [Java Application]
  /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (04 Oct
Fibonacci [Java Application]
  /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (04 Oct
Fibonacci [Java Application]
  Fibonacci at localhost:52475
```

Name	Value
no method return value	
args	String[0] (id=15)
scanner	Scanner (id=16)

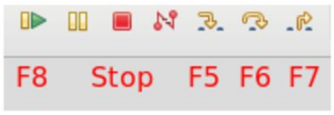
```java
import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = 0;
        do {
            System.out.print("Enter a number (n>2): ");
            n = scanner.nextInt();
        } while (n < 2);

        printArray(getFiboArray(n));
    }

    private static int[] getFiboArray(int n) {
        int[] f = new int[n];
```

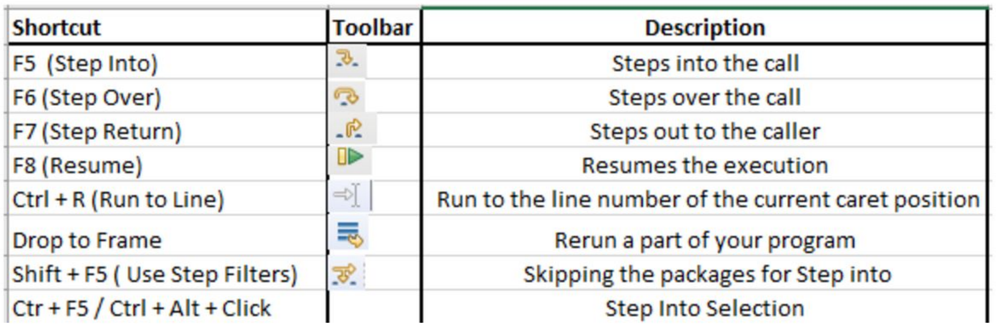
```
Fibonacci [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (04 Oct 2017, 19:39:19)
```

Let’s take a closer look at the views in this perspective (for more information, have a look at this link):

- **Debug:** this view shows the live stack trace of methods. Here we are only in the main method of the Fibonacci program. If other programs were running, we would see them here.
- **Variables:** this view shows all the variables that have been declared and their values. We do not see *n* yet because our breakpoints stop before it was declared.
- **Breakpoints:** this view shows the location of all the breakpoints in your code.
- **Fibonacci.java:** this view shows where we are in the code at the time of the breakpoint
- **Console:** this view shows the output of the program.

8. Eclipse has various buttons in the toolbar and key binding shortcuts to control program execution. These help developers debug their programs.

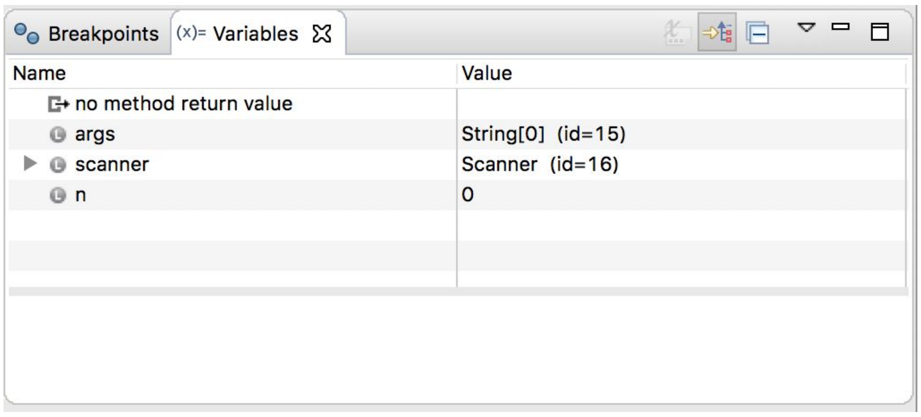




Shortcut	Toolbar	Description
F5 (Step Into)		Steps into the call
F6 (Step Over)		Steps over the call
F7 (Step Return)		Steps out to the caller
F8 (Resume)		Resumes the execution
Ctrl + R (Run to Line)		Run to the line number of the current caret position
Drop to Frame		Rerun a part of your program
Shift + F5 (Use Step Filters)		Skipping the packages for Step into
Ctr + F5 / Ctrl + Alt + Click		Step Into Selection

You can learn more about these stepping commands **here**.

Press the Step Over ( ) button to go to the next line. The new variable *n* should now be declared and initialised to zero. Knowing the values of your variables can eliminate the need for using println statements in your code.



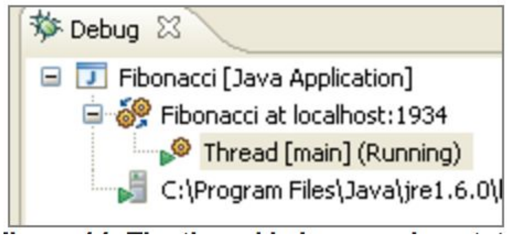
Name	Value
no method return value	
args	String[0] (id=15)
scanner	Scanner (id=16)
n	0

You should now be in the *do-while* loop and suspended on the next line, `System.out.print("Enter a number (n>2): ");`. Step Over will go to the next line in your method and then halt (unless it has some reason to halt before going to the next line).

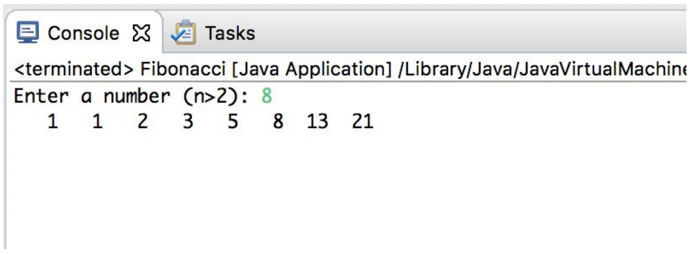
9. Now add a breakpoint in the call to the *printArray* function, `printArray(getFiboArray(n));`.

10. Press the Resume ( ) button to release the current execution and run the program until it comes to another breakpoint.

You might notice that your program has not finished, but is not doing anything. This is because the program is prompting us to enter an integer. We can see that our program (here called a "Thread") is not suspended but running in the Debug view.


Debug
Fibonacci [Java Application]
Fibonacci at localhost:1934
Thread [main] (Running)
C:\Program Files\Java\jre1.6.0\

11. Now In the Console view, type in the number 13 and hit enter.



```
<terminated> Fibonacci [Java Application] /Library/Java/JavaVirtualMachine
Enter a number (n>2): 8
   1    1    2    3    5    8   13   21
```

12. We are now breaking before the *getFiboArray()* and *printArray()* method calls.

Compulsory Task 1

Follow these steps:

- Open the Java file **Errors.java**.
- Attempt to compile the program. You'll notice the compilation will fail due to some errors. Fix all the compilation errors and compile the program again.
- Now run the program. Once again you'll encounter some errors (this time, runtime errors). Fix these errors and run the program once again to see if it works.
- Now run the program and notice the output. You'll notice that there are some logical errors in the code. Fix these errors and run the program once again to see if it now correctly displays what it should.
- Every time you fix an error, add a comment to the line you fixed, and indicate which of the three types of errors it was.

Compulsory Task 2

- Follow the instructions above on how to debug a program using Eclipse.
- Take a screenshot of your debug perspective after following all of the steps and send it to your mentor.

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.