

Chapter 6

How to test, debug, and deploy an application

Objectives

Applied

1. Test and debug your Java applications.
2. Trace the execution of a Java application with `println` statements.
3. Use NetBeans to set breakpoints, step through code, inspect variables, and inspect the stack trace.
4. Use an executable JAR file to deploy a GUI application or a console application.

Objectives (cont.)

Knowledge

1. Distinguish between testing and debugging.
2. Distinguish between syntax, compile-time, runtime, and logic errors.
3. Describe the use of breakpoints and stepping through code when you're using an IDE like NetBeans for debugging.
4. In general terms, describe the advantages and disadvantages for each of the following ways to deploy an application: executable JAR file, Java Web Start, and Installer program.

The Future Value application with a logic error

Welcome to the Future Value Calculator

DATA ENTRY

Enter monthly investment: 100

Enter yearly interest rate: 3

Enter number of years: 3

FORMATTED RESULTS

Monthly investment: \$100.00

Yearly interest rate: 3.0%

Number of years: 3

Future value: \$6,517.42

Continue? (y/n):

The goal of testing

- To find all errors before the application is put into production.

The goal of debugging

- To fix all errors before the application is put into production.

Three test phases

- Check the user interface to make sure that it works correctly.
- Test the application with valid input data to make sure the results are correct.
- Test the application with invalid data or unexpected user actions. Try everything you can think of to make the application fail.

The three types of errors that can occur

- *Syntax errors* violate the rules for how Java statements must be written. These errors, also called *compile-time errors*, are caught by the NetBeans IDE or the Java compiler before you run the application.
- *Runtime errors* don't violate the syntax rules, but they throw *exceptions* that stop the execution of the application.
- *Logic errors* are statements that don't cause syntax or runtime errors, but produce the wrong results. In the Future Value application shown above, the future value isn't correct, which is a logic error.

Code that contains syntax errors

```
public static double getDouble(Scanner sc, String prompt) {  
    d = 0.0; // no data type declared  
    while (true) {  
        System.out.print(prompt) // missing semicolon  
        if (sc.hasNextDouble() { // missing closing paren  
            d = sc.NextDouble(); // improper capitalization  
            sc.nextLine();  
            return d;  
        } else {  
            System.out.println(  
                "Error! Invalid number. Try again.");  
            sc.nextLine();  
        }  
    }  
}
```


Common syntax errors

- Misspelling keywords.
- Forgetting to declare a data type for a variable.
- Forgetting an opening or closing parenthesis, bracket, brace, or comment character.
- Forgetting to code a semicolon at the end of a statement.
- Forgetting an opening or closing quotation mark.

Problems with identifiers

- Misspelling or incorrectly capitalizing an identifier.
- Using a reserved word, global property, or global method as an identifier.

Problems with values

- Not checking that a value is the right data type before processing it. For example, you expect a number to be entered, but the user enters a non-numeric value instead.
- Using one equals sign instead of two when testing numeric and Boolean values for equality.
- Using two equals signs instead of the equals() or equalsIgnoreCase() method to test two strings for equality.

A problem with floating-point numbers

- Using floating-point numbers that can lead to arithmetic errors.
For example:

```
double d = 0.2 + 0.7    // d = 0.8999999999999999
```

- You can prevent errors like this by using BigDecimal numbers or rounding.

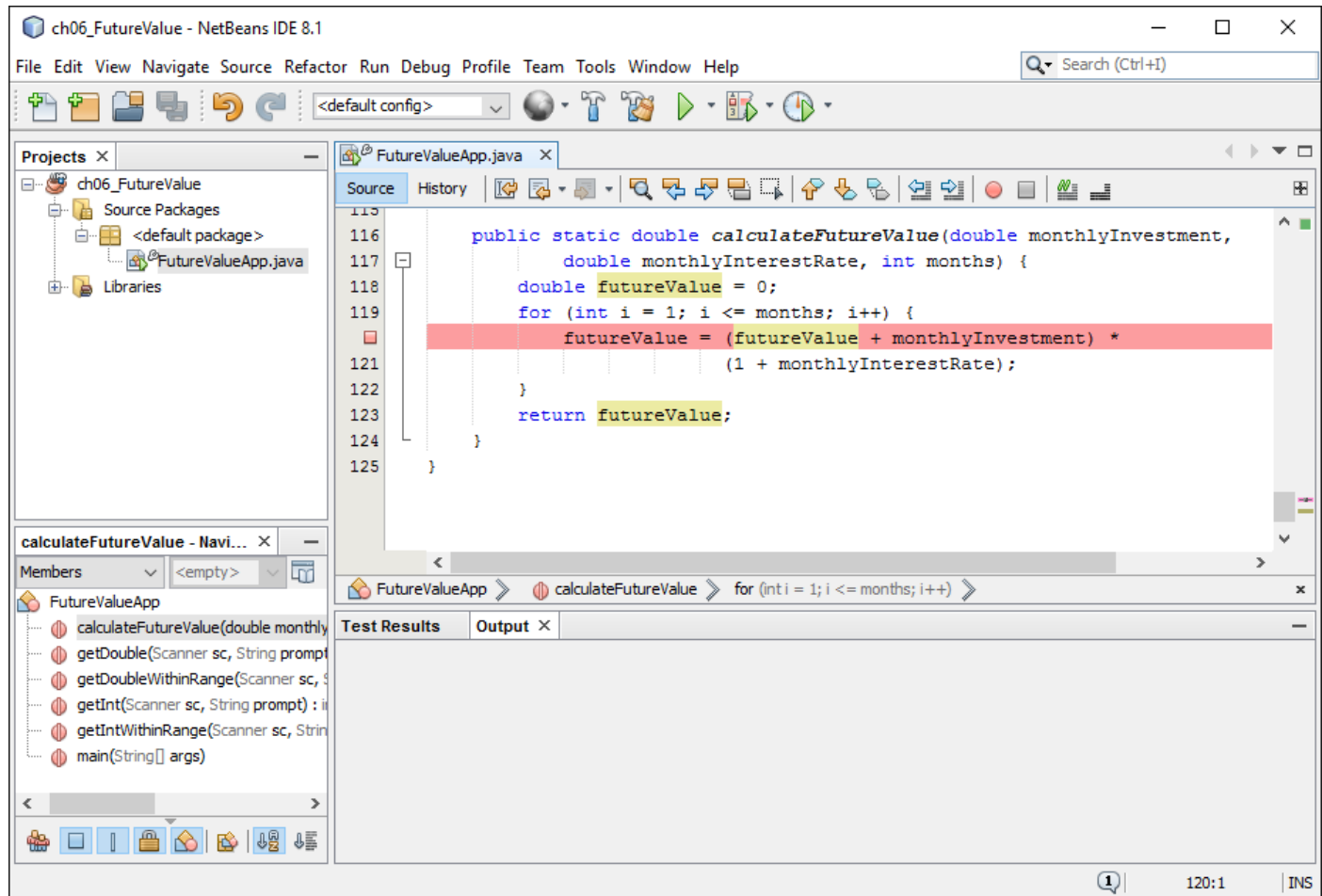
Code that uses println() to trace execution

```
public static double calculateFutureValue(  
    double monthlyInvestment,  
    double monthlyInterestRate, int months) {  
    System.out.println(  
        "starting calculateFutureValue method...");  
    double futureValue = 0;  
    System.out.println("monthlyInvestment: " +  
        monthlyInvestment);  
    System.out.println("monthlyInterestRate: " +  
        monthlyInterestRate);  
    System.out.println("months: " + months);  
    for (int i = 1; i <= months; i++) {  
        futureValue = (futureValue + monthlyInvestment) *  
            (1 + monthlyInterestRate);  
        System.out.println("month " + i +  
            " futureValue: " + futureValue);  
    }  
    return futureValue;  
}
```

The data that's printed to the console

```
starting calculateFutureValue method...  
monthlyInvestment: 100.0  
monthlyInterestRate: 0.03  
months: 36  
month 1 futureValue: 103.0  
month 2 futureValue: 209.09  
month 3 futureValue: 318.3627  
month 4 futureValue: 430.913581  
month 5 futureValue: 546.84098843  
...  
...
```

A code editor window with a breakpoint



A debugging session

ch06_FutureValue - NetBeans IDE 8.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Projects Debugging X

'main' suspended at 'FutureValueApp.main::'

FutureValueApp.cal

FutureValueApp.main::

FutureValueApp.java

```
114 }
115
116 public static double calculateFutureValue(double monthlyInvestment,
117     double monthlyInterestRate, int months) {
118     double futureValue = 0;
119     for (int i = 1; i <= months; i++) {
120         futureValue = (futureValue + monthlyInvestment) *
121             (1 + monthlyInterestRate);
122     }
123     return futureValue;
124 }
125 }
```

calculateFutureValue - N...

Members <empty>

FutureValueApp

- calculateFutureValue(double mon...
- getDouble(Scanner sc, String prom...
- getDoubleWithinRange(Scanner s...
- getInt(Scanner sc, String prompt)
- getIntWithinRange(Scanner sc, S...
- main(String[] args)

Test Results Variables X Breakpoints Output

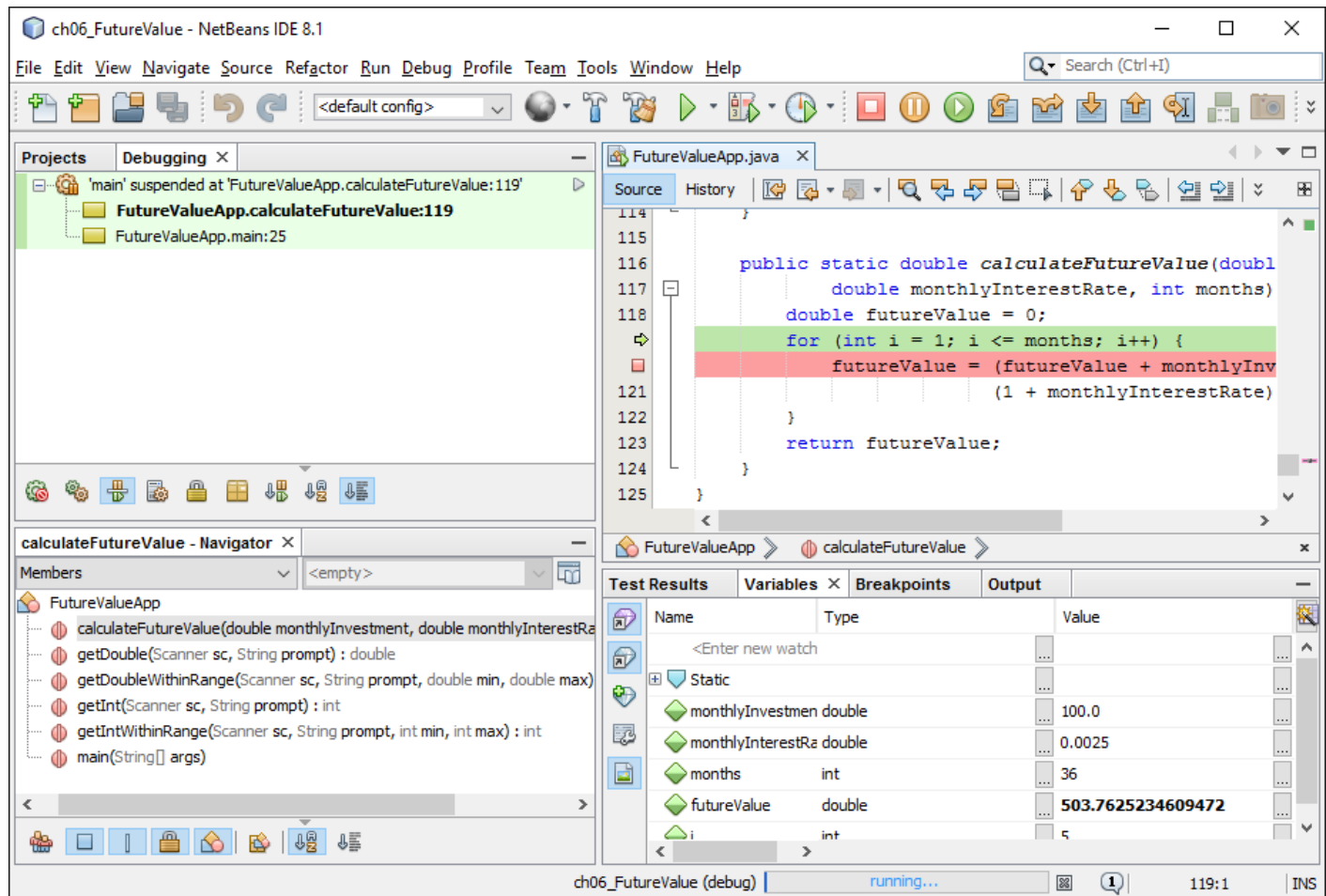
Name	Type	Value
<Enter new watch>		
Static		
monthlyInvestment	double	100.0
monthlyInterestRate	double	0.0025
months	int	36
futureValue	double	503.7625234609472
i	int	5

ch06_FutureValue (debug) running... 119:1 INS

Some of the buttons on the Debug toolbar

Button	Keyboard shortcut
Step Over	F8
Step Into	F7
Step Out	Ctrl+F7
Continue	F5
Finish Debugger Session	

The Call Stack window during debugging



Executable JAR file

An *executable JAR* (Java Archive) *file* stores all the classes and resources needed by your application. You can manually distribute this file to your users and show them how to run it.

Pros

- There are no significant security restrictions.

Cons

- The correct version of the JRE is *not* installed automatically, so you or your users must install it.
- The code is *not* automatically updated, so if the application changes, you must redistribute it.

Web Start

Java Web Start (JWS) allows users to download an application from the web, cache the application locally, and start it.

Pros

- The correct version of Java is automatically installed.
- The code is automatically updated.

Cons

- Security restrictions make it difficult to use.

Installer program

An *installer program* allows you to create an install file for every operating system that you want your application to run on.

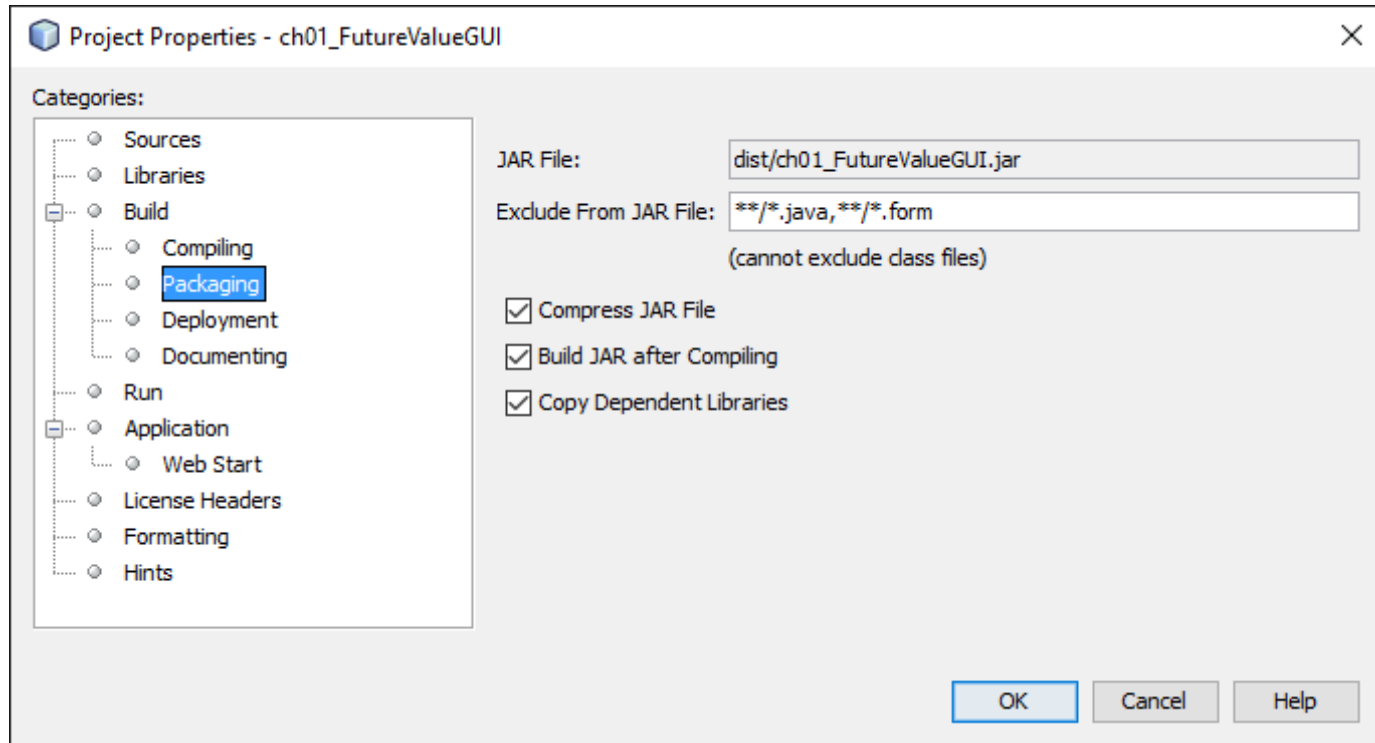
Pros

- The application installs and runs like a professional desktop application.
- The correct version of Java can be installed as part of the installation process.
- There are no significant security restrictions.

Cons

- The code is *not* automatically updated after the program is installed.
- Most installer programs are expensive commercial products.
- This approach requires more work to set up and configure.

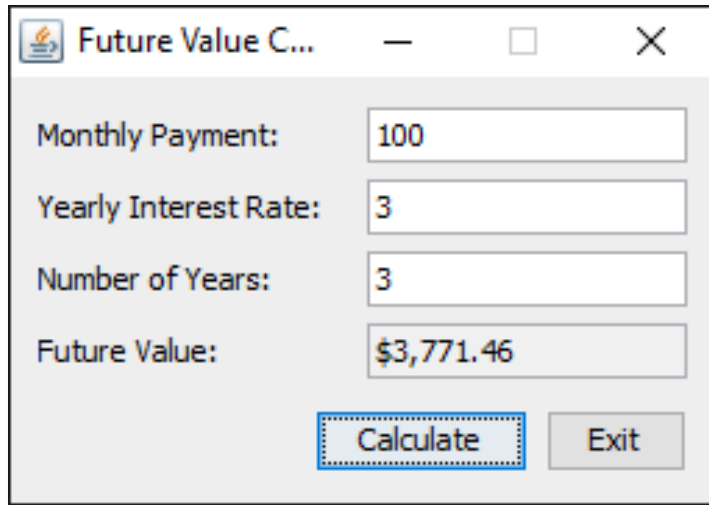
The properties for building a project



How to deploy the files for an application

- Once you've created an executable JAR file for an application, you can deploy it by making the JAR file available to your users.
- The JRE has to be installed on the user's computer for the user to run an executable JAR file.
- To install the JRE on a system other than one running Mac OS X, the user can go to www.java.com.
- The JRE is installed on systems running Mac OS X by default. To install a newer version, the user can use the Software Update feature that's available from the Apple menu.

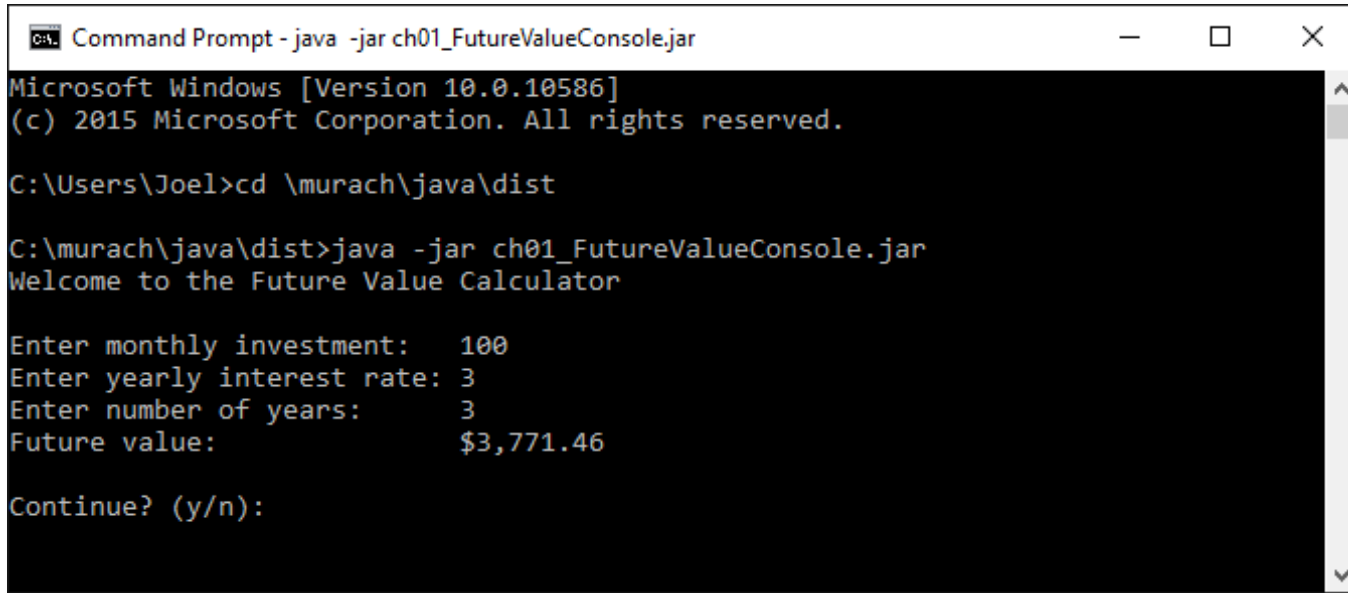
A GUI application running outside of an IDE



How to run an executable JAR file for a GUI application

- Locate the JAR file and then double-click on it.
- For this to work, the operating system must associate the .jar extension with the Java Platform SE binary program that's included as part of the JRE.

A console application running in the Windows command prompt

A screenshot of a Windows Command Prompt window. The title bar reads "C:\ Command Prompt - java -jar ch01_FutureValueConsole.jar". The window has standard Windows window controls (minimize, maximize, close) on the right. The command prompt shows the following text:

```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Joel>cd \murach\java\dist

C:\murach\java\dist>java -jar ch01_FutureValueConsole.jar
Welcome to the Future Value Calculator

Enter monthly investment: 100
Enter yearly interest rate: 3
Enter number of years: 3
Future value: $3,771.46

Continue? (y/n):
```

Syntax to run an executable JAR file

```
java -jar JarName.jar
```

A batch (.bat) file that runs a console application on Windows

```
:: Change to the directory that stores the JAR file  
cd \murach\java\dist
```

```
:: Use the java command to run the JAR file  
java -jar ch01_FutureValueConsole.jar
```

A bash (.sh) file that runs a console application on Mac OS X or Linux

```
#!/bin/bash
```

```
# Change to the directory that stores the JAR file  
cd /murach/java/dist
```

```
#Use the java command to run the JAR file  
java -jar ch01_FutureValueConsole.jar
```