LEARN JAVA

THE COMPLETE BEGINNER'S GUIDE TO LEARN JAVA PROGRAMMING

BRUCE BERKE

Learn Java

THE COMPLETE BEGINNER'S GUIDE TO LEARN JAVA PROGRAMMING

Table of Contents

INTRODUCTION

Java is a programming language used by more than 3 billion devices. Java is used to build desktop, web and mobile applications. It was developed by Sun Computers in 1995. Java runs on multiple platforms i.e. Windows, MAC and multiple versions of UNIX.

Audience

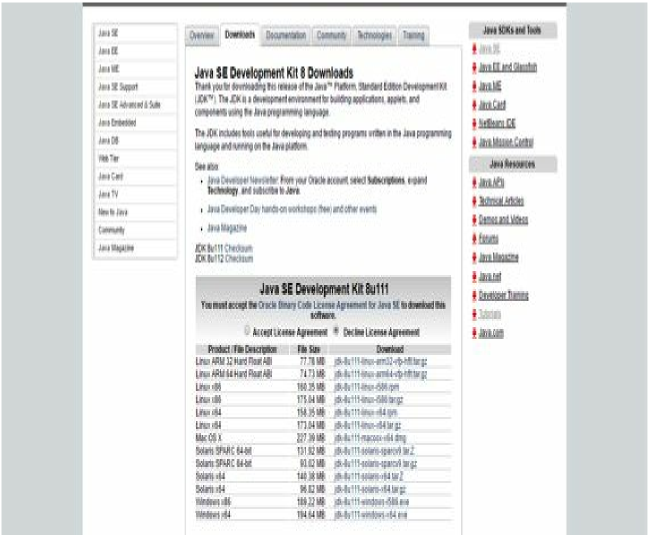
This book is intended for absolute beginners. If you haven't done programming yet, then this is your book.

This book will help you in understanding the basic concepts of not only programming in general but will also help you out in getting familiar with concepts of Java.

Chapter 1: Environmental Setup

If you haven't ever installed Java on your computer, don't worry I will guide you through the process of installing and running Java on your computer. You will need to do the following steps in order to setup your environment for Java Programing

- Go to http://www.oracle.com/technetwork/downloads-2133151.html



Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to Java.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 8u111 Checksum
JDK 8u112 Checksum

Java SE Development Kit 8u111

You must accept the Oracle Binary Code License Agreement for Java SE **to download this software.**

Accept License Agreement [✓ Decline License Agreement]

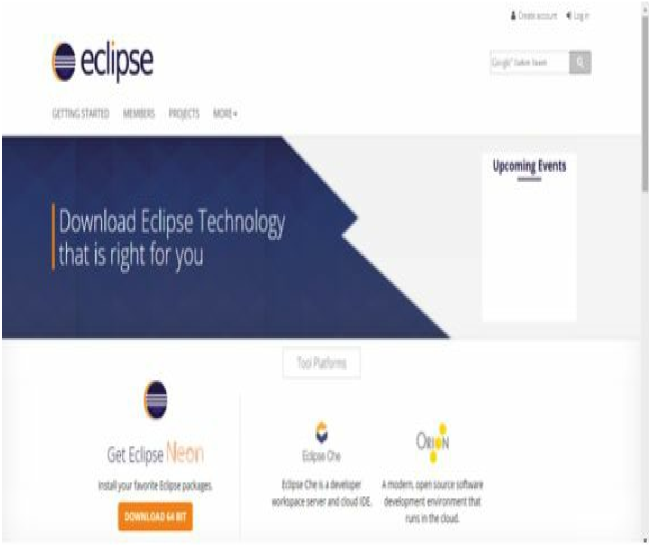
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.78 MB	jdk-8u111-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.73 MB	jdk-8u111-linux-arm64-vfp-hflt.tar.gz
Linux x86	160.35 MB	jdk-8u111-linux-i586.rpm
Linux x86	175.04 MB	jdk-8u111-linux-i586.tar.gz
Linux x64	158.35 MB	jdk-8u111-linux-x64.rpm
Linux x64	173.04 MB	jdk-8u111-linux-x64.tar.gz
Mac OS X	227.39 MB	jdk-8u111-macosx-x64.dmg
Solaris SPARC 64-bit	131.92 MB	jdk-8u111-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	93.02 MB	jdk-8u111-solaris-sparcv9.tar.gz
Solaris x64	140.38 MB	jdk-8u111-solaris-x64.tar.Z
Solaris x64	96.82 MB	jdk-8u111-solaris-x64.tar.gz
Windows x86	189.22 MB	jdk-8u111-windows-i586.exe
Windows x64	194.64 MB	jdk-8u111-windows-x64.exe

Choose an installation setup according to your operating system. X86 means that your operating system must be 32-bit. You can install x86 or x64 versions of Java development Kit if your operating system is 64-bit. The Java development Kit includes the libraries for development. It also has Java runtime environment which is used in order to run java based applications. After you download the setup, install it.

Now that you have installed Java on your system you will need to install an IDE (Integrated Development Environment). An IDE is a tool you will use along with Java to build cool applications.

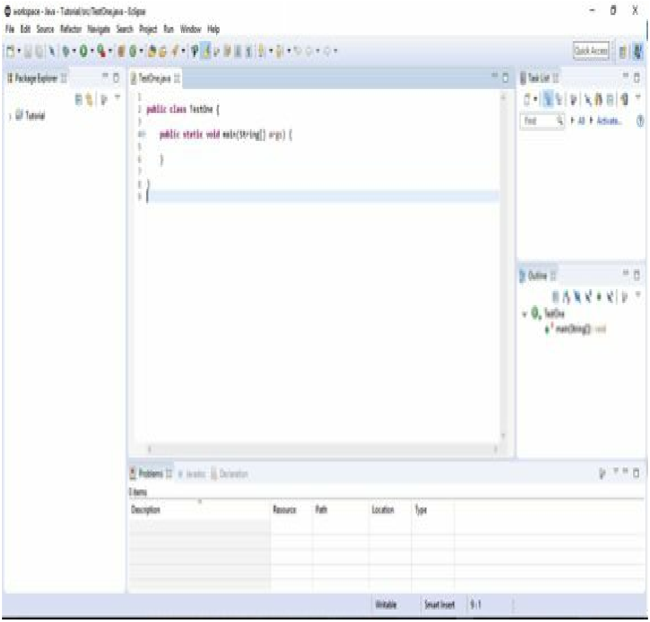
I will use Eclipse as an IDE. Eclipse is a very popular IDE used by millions of java developers. There are other options available such as NETBEANS, IntelliJ etc. I would recommend you guys to install eclipse for now in order to avoid the hassles of installing other tools. Once you start understanding the basics of Java and how it works you can go ahead and install and play with these other tools too.

- Go to https://www.eclipse.org/downloads/


eclipse
GETTING STARTED
MEMBERS
PROJECTS
MORE
Upcoming Events
Download Eclipse Technology that is right for you
Tool Platforms
Get Eclipse Neon
Install your favorite Eclipse packages.
DOWNLOAD 64 BIT
Eclipse Che
Eclipse Che is a developer workspace server and cloud IDE.
ORION
A modern, open source software development environment that runs in the cloud.

Download setup of Eclipse from the above mentioned link. It has a very simple installation. After Installation of eclipse you will see its icon on the desktop. Double Click on it to run

Eclipse. It will ask you for a workspace. A workspace is a directory on your computer where all your java projects will be stored. Select a workspace of your choosing and click OK. Now you’ll see the following screen.


workspace - Java - Tutorial/src/TestOne.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Tutorial
TestOne.java
Task List
Outline
TestOne
main(String[]) : void
Problems Javadoc Declaration
0 items
Description Resource Path Location Type
Writable Smart Insert 9 : 1

```
public class TestOne {

    public static void main(String[] args) {

    }

}
```

If you see a screen similar to the one above that means you have done everything according to my instructions and now you and your computer are ready for programming in java.

Chapter 2: Basic Syntax

Java is a case sensitive language. Programs written in java contain objects, classes, variables and methods.

Object:

Consider an object as a real life entity with a behavior. For example a cat is an object its behavior can be its ability to walk, eat, sleep etc.

Class:

A class is a blueprint and based

on that blue print you create multiple objects.

Variables:

Variable can be attributes of a class like its name, size, and etc. all the variables go into the class and are used once the class is instantiated. By instantiation I mean that you will create an object of a class and then use that object to assign values to its variables. I will explain these concepts in details as we go along this book.

So let's go ahead and open your eclipse IDE, it's time to write some code. When you open eclipse you will see a class names **TestOne** which is created by default when you start eclipse. You can create a new project and then do the same thing too.

Example:

```
public class TestOne {
            public static void main(String []args)
{
              System.out.println("Hello World"); //
prints Hello World
            }
}
```

The code above, uses three basic things

of Java Programming i.e. class, method and variable. The class **TestOne** contains a method. Now this method named as main is the origin of your java program. This is where your java program goes when you run your program. Inside the main method the **println** method prints “Hello World” to your console. The **public** keyword is an access specifier and it allows you the developer of the program to restrict access of a class, method or a variable. Other access specifier are **private**, **protected**, **default**. Go ahead and hit the

Run button it will look like this . After running the program you will see a console window at the bottom of your screen showing you the output of your first java program.

As I told you earlier that java is a case sensitive language. So the word **VOID** and **void** will have different meanings so you will have to be very careful when you use keywords or custom named identifiers for classes, variables, objects and methods. A java program file has a **.java** extension and the code written in this type of file will be known to the

JRE. If you write your java code in a file whom extension is not **.java** then the JRE will not recognize that file or the code written inside it.

Java Keywords:

A java keyword is a reserved word which is used by java itself and java doesn’t allow the programmers to use those words. The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier’s name, meaning you cannot name your attributes of a class (variables), the behavior of a class (methods) with the exact words described below.

continue	abstract	Switch	for
default	assert	synchronized	goto
do	boolean	this	if
double	break	throw	implement
else	byte	throws	import
enum	case	transient	instanceof
extends	catch	try	int
final	char	void	interface
finally	class	volatile	long
float	const	while	native

Comments:

There are multiple types of comments in java. A single line comment and a block comment. The following example shows multiple type of comments in java.

Example:

```
public class TestOne {

	/* This is my first java program.
	 * This will print 'Hello World' as the
output
	 * This is an example of multi-line
comments.
	 */
```

```
        public static void main(String []args)
{
            // This is an example of single line
comment
            /* This is also an example of single
line comment. */
            System.out.println("Hello World"); //
prints Hello World
        }
}
```

CHAPTER 3: OBJECTS AND CLASSES

Java is an object oriented language which means that it will have objects and classes. You may have got a general idea of objects and classes by my previous discussion. Now I will explain these concepts in detail here.

Objects:

Object is something that has some attributes and behaviors. Consider human as a real life entity, now we will map human into our software object. A human has a name, age etc. he walks, talks and eats etc. all of these are its behaviors.

Class:

Class is a blueprint and by using these blueprints we create objects. Following is an example of a class in java. Feel free to tweak it if you want to.

Example:

```
public class Human {
        String name;
        int age;
        String gender;
        void walks() {
```

```
        // What happens when a person walks
        }
        void talks() {
        // What happens when a person talks
        }
        void eats() {
        // What happens when a person eats
        }
}
```

Now we can use the above mentioned blueprint to create as many objects we want. A class can have multiple types of variables.

Local Variables:

Local Variables are the variables that

are declared inside of methods, constructors or blocks.

Instance Variables:

Instance Variables are the variables that are declared inside a class and are only allowed for use once the class is instantiated.

Class Variables:

Class Variables are declared with a keyword static and they can be used by directly via the name of a class.

Constructors:

A constructor is a method inside a class that gets called when we create an object of that class. Constructors are used to set values to attributes or to make calls to a method when we create an object of a class. Following is an example of a constructor:

Example:

```
public class Human {
        String name;
        int age;
        String gender;
        public Human() {
        // This is constructor and this is where
you can assign values to your variables.
```

```
        // or you can call functions.
        }
        void walks() {
        // What happens when a person walks
        }
        void talks() {
        // What happens when a person talks
        }
        void eats() {
        // What happens when a person eats
        }
}
```

Now let us talk about creating object of our own. Create a class named **Human.java** in your IDE and write the content of the below mentioned **Human** class into your new class. Now go to you **TestOne.java** class and review the main method. This is the method we will use to instantiate our **Human** class object. Write the code written in the **TestOne** class below. Run the program and you will see **John** written on your console window.

Example:

Human.java

```
public class Human {


        String name;
        int age;
        String gender;
        void walks() {
        // What happens when a person walks
        }
        void talks() {
        // What happens when a person talks
        }
        void eats() {
        // What happens when a person eats
        }
}
```

TestOne.java

```java
import java.io.Console;

public class TestOne {
        public static void main(String[] args) {
        // TODO Auto-generated method stub
        Human human = new Human();
        human.name = "John";
        System.out.println(human.name);
        }
}
```

Go ahead and assign values to age and gender variable and try printing them out. It will be a good exercise for you.

CHAPTER 4: BASIC DATATYPES

Variables store values in themselves. They are just reserved memory locations inside which you can store multiple types of values. Java has two main types of variables and in this chapter we will cover not only those types but also their sub types. The main variable types in Java are:

- Primitive Data Types
- Reference Object/Data Types

Primitive Data Types:

Primitive data types are those types that are defined in Java. Java has also devised some rules for these types i.e. what kind of a value they will store? What's their range? Etc. Java has eight primitive types. Each primitive type is explained below in detail.

byte:

- This one is an 8-bit signed two's complement integer.
- Its minimum value is -128(-2^7).
- Its maximum value can be 127(2^7-1).

- By default its value is zero.
- Byte is used to save data in big arrays where there is a need to save some memory.
- You can use it instead of an **int** but you will have to keep in mind the range of this type as it is smaller than int.

Example:

- **byte** variableOfType_byte = 10; //this is how you declare a variable of type 'byte' in Java.
- **byte** variable2OfType_byte = -40; //Another example.

short:

- **short** is a 16-bit signed two's complement integer.
- Its minimum value is -32,768 (-2^15).
- Its maximum value is 32,767 (2^15 -1).
- By default its value is zero.
- As short is two times smaller than an integer, it can also be used to save memory.

Example:

- **short** variableOfType_short = 40; //this is how you declare a variable of type 'short' in Java.

- `short variable2ofType_short = -30; //Another example.`

int:

- **int** is a 32-bit signed two’s complement integer.
- Its minimum value is -2,147,483,648 (-2^31).
- Its maximum value is 2,147,483,647 (2^31 -1).
- By default its value is zero.
- This type of variables is used for storing values of integer types.

Example:

- **int** variableOfType_int = 1024; //this is how you declare a variable of type 'int' in Java.
- **int** variable2ofType_int = -1024; //Another example.

long:

- **long** is a 64-bit signed two’s complement integer.
- Its minimum value is -9,223,372,036,854,775,808 (-2^63).
- Its maximum value is 9,223,372,036,854,775,807 (2^63 -1).
- By default its value is zero.

- This type of variables is used for storing values of integer types whom have a wider range.

Example:

- **long** variableOfType_long = 100000L; //this is how you declare a variable of type 'long' in Java.
- **long** variable2ofType_long = -100000L; //Another example.

float:

- **float** is a single-precision 32-bit IEEE 754 floating point.
- By default its value is ‘0.0f’.
- This type of variables is used for

storing values of float types.

Example:

- **float** variableOfType_float = 100000.99f; //this is how you declare a variable of type 'float' in Java.
- **float** variable2ofType_float = 00.975333; //Another example.

double:

- **double** is a double-precision 64-bit IEEE 754 floating point.
- By default its value is ‘0.0d’.
- This type of variables is used for storing values of decimal types.

Example:

- **double** variableOfType_double = 235.99d; //this is how you declare a variable of type 'double' in Java.
- **double** variable2ofType_double = 101.3d; //Another example.

boolean:

- **boolean** stores one-bit information.
- By default its value is false.
- This type of variables is used for storing true/false values.

Example:

- **boolean** variableOfType_boolean = **true**; //this is how you declare a variable of type 'boolean' in Java.

- **boolean** variable2ofType_boolean = **false**; //Another example.

char:

- **char** stores 16-bit Unicode characters.
- Char is used to store characters.
- Its minimum value is ‘\u0000’.
- Its minimum value is ‘\uffff’.

Example:

- **char** variableOfType_char = 'a'; //this is how you declare a variable of type 'char' in Java.
- **char** variable2ofType_char = 'c'; //Another example.

Reference Types:

Reference type variables are class objects. You can also consider arrays of different types an example of reference type variables. The default value of these kind of variables is NULL. See the following Example.

```
import java.sql.Date;

public class Employee {

        String name;
        Date DateOfBirth; //Reference Variable
        int     Age; //Primitive Type Variable
        String Designation;
```

Chapter 5: Variable Types

Variables are used in java to store different type of values. Each variable has a type and value. The type of a variable tells java the type of value it will possess and the range of that value. It also helps us narrow down the type of operations that can be applied on that variable.

A variable must be declared before it's used. Following is a generic template for declaring a variable

---

Datatype Variable’s name = Variable’s Value;

```
int Age = 24;
```

Data type represents the type of data this variable will carry, Variable’s name will show its name and then after the equal sign you will assign it a value based on its type.

There are three kind of variables

- Local Variable.

- Instance Variables.
- Class/Static Variables.

Local Variables:

Local variables are the type of variables that are declared and used inside a block of code. Local variables can be declared inside methods, constructors and other code blocks like If-else, switch etc. these kind of a variables are created when the compiler enters that block of code and are deleted when the compiler exists that block. Local variables cannot use access modifiers like public/private with them as they have a limited scope.

Example:

In the following example the **Employee** Class has a method **calculateAge()** in which we are assigning a value to **Age** variable and are also printing it out on the console window.

```
import java.sql.Date;
public class Employee {
        public void calculateAge(){
        int Age = 0;
        Age = Age + 24;
        System.out.println(Age);
        }
}
```

The employee class is instantiated in the **main** class and the function **calculateAge** gets called.

```
import java.io.Console;
public class TestOne {
        public static void main(String[] args) {
        // TODO Auto-generated method stub
        Employee human = new Employee();
        human.calculateAge();
        }
}
```

Output:

```
24
```

Instance Variables:

Instance variables are declared inside of a class and outside of any code block like if-else, methods etc. These variables are used when an object of a class is created. When an object of a class is created, memory in heap is allocated to that object and part of that heap are allocated to each instance variable. Instance variables life starts when an object is created and ends when that objects ends. You can set the access level of an instance variable. The default values of instance variables depend on its data type. Look at the previous chapter for variables data types and their default values.

Example:

```
import java.sql.Date;
public class Employee {
        String name; //Example of Instance
Variables
        Date DateofBirth; //Example of Instance
Variables
        String Designation; //Example of
Instance Variables
        Public void calculateAge(){
        int Age = 0;
        Age = Age + 24;
        System.out.println(Age);
        }
```

}

Class/Static Variables:

Class variables or static variables are variables that are declared outside of block or methods but inside of class with a **static** keyword. Static variables will have only one value per class and the number of object that class has will not effect this static variable. The static variables are stored in static memory. The static variables are accessed by the following way

---

ClassName.StaticVariableName

```
            Employee.Name;
```

Example:

```
import java.sql.Date;

public class Employee {
        String name; //Example of Instance
Variables
        Date DateofBirth; //Example of Instance
Variables
        String Designation; //Example of
Instance Variables

        private static double salary;
//Example of Class/Static Variables
        private static String Department;
//Example of Class/Static Variables
```

```
        public void calculateAge(){
        int Age = 0;
        Age = Age + 24;
        System.out.println(Age);
        }
}
```

CHAPTER 6: MODIFIER

TYPES

Java has two type of modifiers

- Access Modifiers
- Non-Access Modifiers

Access Modifiers:

Access modifiers are the type of modifiers that decide who can see a certain variable, class, method etc. It has multiple types.

public:

The public access modifier will

allow any class, interface, method or block to access this variable inside of the Java universe.

Example:

```
public class Car{

         public int speed = 0;
         /*see the public keyword? This
keyword helps
        java is figuring out the access level of
this variable*/

        public void calculatespeed(){
        speed = 24;
        System.out.println(speed);
        }
}
```

private:

When the private keyword is attached with a variable it means that, that particular variable will be accessed only from the inside of that class. Private is the most restricted access level in java.

Example:

```
public class Car{

        public int topspeed = 0;
        /*see the public keyword? This keyword
helps
```

```
        Java is figuring out the access level of
this variable*/

        //this variable will be accessed only
inside this class
        private String make = “BMW”;
        public void calculatetopspeed(){
        topspeed = 24;
        System.out.println(topspeed);
        }
}
```

protected:

The protected keyword is used to restrict access of variables and methods.

The protected code of a parent class is only allowed to be accessed by its child class. Apart from the child class nothing else can access protected variables or methods.

Example:

```
class SongPlayer {
          protected boolean PlaySongs(Songs
s) {
            // implementation details
          }
        }
class EnglishSongsPlayer extends SongPlayer
{
  boolean PlaySongs(Songs s) {
```

```
            // implementation details
        }
}
```

In the above example the **Playsongs** method of **AudioPlayerClass** can be accessed only by its child class which in this case is **StreamingAudioPlayer**.

Non-Access Variables:

static:

You can declare static variables or static methods. Use of word static will ensure that the declared variable or method would only be allowed to access by the name of class rather than its object.

final:

The final keyword is used to restrict the use of a variable after its initialized. Meaning you cannot initialize an already initialized variable which has a final keyword attached with it.

Chapter 7: Basic Operators

In Java there are multiple types of operators and by using these operators you can perform mathematical or logical operations.

Let’s suppose, there are two variables named **a** and **b.** a is equal to 10 and b is equal to 20.

Arithmetic operators:

The following table contains all the info you need for arithmetic operators in java.

Operator	Description	Exampl
+ (Addition)	For adding values	a + b = 30
- (Subtraction)	For subtracting values	a - b = -10
* (Multiplication)	For multiplying values	a * b = 200
/ (Division)	For dividing values	b / a =

% (Modulus)	For getting remainders	b % a = 0
++ (Increment)	Add 1 to value.	b++ gives 21
-- (Decrement)	Subtract 1 to value.	b-- give 19

Relational operators:

The following table contains all the info you need for relational operators in java.

Operator	**Description**	**Example**
== (equal	This compares	(a == b is

to)	two values and If values are equal than it returns true.	not true.
!= (not equal to)	This compares two values and if values are not equal then condition becomes true.	(a!= b) is true.
> (greater than)	If the value on the left is greater than value on the	(a > b) is not true.

	right then condition becomes true.	
< (less than)	If value on the left is less than value on the right then condition becomes true.	(a < b) is true.
>= (greater than or equal to)	If value on the left is greater than or equal to the value on the	(a >= b) is not true.

	right than it returns true.	
<= (less than or equal to)	If the value on the left is less than or equal to the value on the right then condition becomes true.	(a <= b) is true.

Logical Operators:

The following table contains all the info you need for logical operators in java.

Operator	Description
&& ('AND')	This operator is known as I AND operator. If values o sides of this operator are no then it returns true otherw result will be false.
|| ('OR')	This operator is known as I

	OR Operator. If value on ar of this operator is non-zero, returns true otherwise it return false.
! (‘NOT’)	This operator is known as I NOT Operator. This op inverses the value of the op it is used with. For example value is true it will make i and if the value is false make it true.

Assignment Operators:

The following table contains all the info you need for assignment operators in java.

Operator	Descriptio
=	This is an assignment ope value from right side and sto side.
+=	This operator not only adds assigns it to the variable on
-=	This operator not only subtr also assigns it to the variabl

*=	This operator not only mult it also assigns it to the va side.
/=	This operator not only divi also assigns it to the variabl
%=	This operator not only tak value but it also assigns it t the left side.
<<=	This operator performs a le and it also assigns it to the v side.

>>=	This operator performs a rig and it also assigns it to the v side.
&=	This operator performs an A a value and it also assigns on the left side.
^=	This operator performs an E operation on a value and it a the variable on the left side.
|=	This operator performs an O value and it also assigns it to the left side.

Bitwise Operator:

The following table contains all the info you need for bitwise operators in java.

Operator	**Description**
& (bitwise AND)	This is a binary ANI operator. It copies a bit t the result if it exists o both side of the operator.
| (bitwise OR)	This is a binary Ol operator. It copies a bit t the result if it exists o

	any side of the operator.
^ (bitwise XOR)	This is a binary XOI operator. It copies a bit t the result if it exists o one side and not on th other side of the operator
~ (bitwise Compliment)	This is binary One complement Operator an it is used to flip bits.
<< (Left Shift)	This is a binary Left Shi operator. It shifts bits c the variable on the left t the left side by the numbe

	of bits specified on th right side of the operator.
>> (Right Shift)	This is a binary Righ Shift operator. It shift bits of the variable on th left to the right side by th number of bits specifie on the right side of th operator.
>>> (zero fill right shift)	This is right zero fil operator. The value on th left side is move right b the number of bit

	specified on the other sid of this operator. Th shifted values ar replaced with zero afte zero fill operation.

CHAPTER 8: LOOPS

In real life, loop is something that goes over and over. Loops work almost the same in java. In java you have a condition and based on that condition you can run a loop. There are three kinds of loops in java:

- While loop
- Do-while loop
- For loop

While Loop:

When a while loop executes, it

keeps on iterating and running the same block of code over and over until the condition defined with the loop is true. When the condition becomes false the loop breaks.

Example:

```
        int i = 0;
        while(i < 10) {
        System.out.println("Programming is Fun
"+ i);
        i++;
        };
```

Output:

```
Programming is Fun 0
Programming is Fun 1
Programming is Fun 2
Programming is Fun 3
Programming is Fun 4
Programming is Fun 5
Programming is Fun 6
Programming is Fun 7
Programming is Fun 8
Programming is Fun 9
```

The above code will print the phrase Programming is fun 10 times.

Do-while loop:

A Do-while loop is similar to a while loop except that it executes at least one time. After its first iteration it checks whether the condition mentioned at the end of loop is true or not. If the condition is true it will keep on executing otherwise the loop will break.

Example:

```
        int i = 0;

        do {
        System.out.println("Programming is Fun
"+ i);
        i++;
        } while (i < 10);
```

Output:

```
Programming is Fun 0
Programming is Fun 1
Programming is Fun 2
Programming is Fun 3
Programming is Fun 4
Programming is Fun 5
Programming is Fun 6
Programming is Fun 7
Programming is Fun 8
Programming is Fun 9
```

The above code will have the same output as to the one in the while loop case.

For Loop:

A “For loop” is another kind of loop that java supports. In for loop you define the condition which must be met in order for the loop to execute. If the condition is false the loop will break. Otherwise for loop will keep on executing. The syntax of for loop is following

Example:

```
        for (int j = 0; j < 9; j++) {
        System.out.println("This is for loop's
iteration number "+ j);
        }
```

Output:

```
This is for loop's iteration number 0
This is for loop's iteration number 1
This is for loop's iteration number 2
This is for loop's iteration number 3
This is for loop's iteration number 4
This is for loop's iteration number 5
This is for loop's iteration number 6
This is for loop's iteration number 7
This is for loop's iteration number 8
```

Break and Continue:

In loops the keywords break or continue can be used to perform certain operations.

Break:

A break statement in any kind of loop will break the loop no matter if the condition of the loop was true or false. As soon as the loop hits the break statement it will stop executing.

Example:

```
        int i = 0;
        do {
        if(i>5) {
        break;
        } else {
        System.out.println("Programming is Fun
"+ i);
        }
        i++;
        } while (i < 10);
```

Output:

```
Programming is Fun 0
Programming is Fun 1
Programming is Fun 2
Programming is Fun 3
Programming is Fun 4
Programming is Fun 5
```

As soon as the loop hits the break statement it will stop iterating.

Continue:

When the code reaches a continue statement inside a loop it skips that iteration and the code below the continue statement and will jump to the next iteration.

Example:

```
        int i = 0;
        do {
        if(i>=5) {
        continue;
        } else {
        System.out.println("Programming is Fun
"+ i);
        }
        i++;
        } while (i < 10);
```

Output:

```
Programming is Fun 0
Programming is Fun 1
Programming is Fun 2
Programming is Fun 3
Programming is Fun 4
```

Chapter 9: Decision Making

In java decision making refers to block of codes which execute only if the condition mentioned before them is true. There are multiple kind of decision making blocks in java.

- If Statement
- If-else statement
- Switch statement

If Statement:

An "If statement" is a block of

code which comprises of a single or multiple conditions and the lines of code that execute if the conditions are met.

Example:

```
        if (true) {
        System.out.println("This is an if
statement example");
        }
```

Output:

```
This is an if statement example
```

If-Else Statement:

An “If-Else statement” is a block of code which comprises of a single or multiple conditions and the lines of code that execute if the conditions are met. If the conditions aren’t met the else block of this statement executes.

Example:

```
        if(false) {
        System.out.println("This is an if-else
statement example");
        }
        else {
        System.out.println("Else block's
output");
        }
```

Output:

```
Else block's output
```

The condition inside the if statement above is false so the compiler will jump to the else block to print the above mentioned output.

Switch Statement:

A switch is very much like an If statement and it works just like an if statement. But what if you have a lot of conditions and you want to do different operations for each condition, you'll have to write multiple if statements to achieve your desired result but with a switch statement you can do all those things inside a single block of code.

Example:

```
String color = "Blue";
    switch(color) {
      case "Red" :
        System.out.println("Red");
        break;
      case "Violet" :
      case "Blue" :
        System.out.println("Blue");
        break;
      case "Green" :
        System.out.println("Green");
      case "Orange" :
        System.out.println("Orange");
        break;
      default :
        System.out.println("White");
    }
    System.out.println("The Color you choose 
is "+ color);
  }
```

Output:

```
Blue
The Color you choose is Blue
```

See the above mentioned switch statement. I have stored the Blue color in the color variable. When I execute this code. It goes to each case statement and compares it with the value I have stored in the color variable. When it finds the match it runs the code inside that case and exits the statement.

CHAPTER 10: NUMBERS

Normally in java when we have to deal with numbers we use integer, float, decimal etc. but you must know that all of these primitive types come from an abstract numbers class. When we create variables of primitive types what really happens behind the scene is that the primitive types gets wrapped around by the Numbers class and as a result an object is created. This process of creating an object from a primitive type is known as boxing. When we convert an object to a primitive type which most of the time what the compilers does to show us numbered values is known as un-boxing.

Number Methods:

The Number class has a lot of built in functions written for you. So you just have to call these methods and use them instead of writing these on your own.

xxxValue()

This method converts the value

of a Number object to the desired data type and returns the result.

compareTo()

This method compares a Number object to an argument.

equals()

This method checks if the number object is equal to the argument.

valueOf()

This method return an object of integer type which has a value specified in a primitive variable.

toString()

This method converts the value into a string.

parseInt()

This method is used to convert a string into an int.

abs()

This method return the absolute value.

ceil()

This method returns a double value greater than or equal to the argument.

floor()

The method returns the largest double value which is lesser than or equal to the argument.

rint()

This method returns a double value closer to the argument.

round()

This method returns a long or int value.

min()

This method return the smaller

value among multiple values.

max()

This method return the biggest value among multiple values.

exp()

The method return the base of log e.

log()

This method performs a log operation on the argument given.

pow()

This method returns the value of the first argument raised to the power of the second argument.

sqrt()

This method performs a square root operation on the argument.

sin()

This method returns the sine of the argument.

cos()

This method returns the cosine of the argument.

tan()

This method returns the tan of the argument.

asin()

This method returns the arcsine of the argument.

acos()

This method returns the arccosine of the argument.

atan()

This method returns the arctan of the argument.

toDegrees()

You can convert an argument into degrees with the help of this method.

toRadians()

You can convert an argument into radians with the help of this method.

random()

This helper method generates a random number. You can give it a range to restrict the outcome.

Chapter 11: Characters

Just like numbers, in Java you can create objects of primitive type **character** by using its wrapper class **Character**. A character class creates an object in which you can store characters and you can also perform different actions on that particular object.

Example

```
        //this is a primitive character type
variable
```

```
        char mychar = 'a';
        // this is how you create a character
object
        Character mycharobject = 'b';
        //this is a primitive character type
variable
        char one = '1';
        // this is how you create a character
object
        Character twoobj = '2';
```

Methods for Character Class:

isLetter()

This method checks if the argument passed to it is a letter or not.

isDigit()

This method checks if the char value in question is a digit or not.

isWhitespace()

This method checks if the char value in question is a whitespace or not.

isUpperCase()

This method checks if the char value in question is uppercased or not.

isLowerCase()

This method checks if the char value in question is lower cased or not.

toUpperCase()

This method converts the char

value in question to uppercase.

toLowerCase()

This method converts the char value in question to lowercase.

toString()

This method returns a string object containing the value of the character passed to it as an argument.

CHAPTER 12: STRINGS

Strings are widely used in java. In java Strings are treated as objects. Java has built-in string class and that class contains many helper methods.

Here’s how you define a string in java.

```
String name = "John";
```

When the compiler hit the above line of code it will create an object of type string whom value is John.

Example:

```
        String name = "John";
        System.out.println("Hello guys my name
is "+ name);
```

Output:

```
Hello guys my name is John
```

In the example above I have created a string object **name** and have assigned a value to it. The second line of code takes a string argument and prints it out on the console window. The string **Hello guys my name is** and the value of the **name** variable are both strings. The plus sign (+) I used above is for joining both strings. The process of joining strings together is known as concatenation. Here’s another example.

Example:

```
        String greet1 = "Hello";
        String greet2 = "World";
        System.out.println(greet1 + " " +
greet2);
```

Output:

```
Hello World
```

In the example above I joined two string variables named **greet1** and **greet2**. I also concatenated a space in between them to improve readability.

You can also format a string in java. Meaning you can use placeholders for values and can pass values to the print statement on the go. Have a look at the following example.

Example:

```
        int number1 = 1;
        float number2 = 2.0f;
        System.out.printf("The value of
number1 is %d
                    and the value of number2 is
            %f", number1, number2);
```

Output:

```
The value of number1 is 1 and the value of
number2 is 2.000000
```

In the above example I passed two variables **number1** and **number2** to the **printf** statement. The **"%d"** is a place holder for integers and the **"%f"** is a place holder for float point values.

String Methods:

The string class has multiple pre-defined helper methods.

char charAt(int index)

This method returns a character on a specific index of a string.

int compareTo(Object o)

This method compares two string objects.

String concat(String str)

This method concatenates the specified string to the end of another string.

boolean contentEquals(StringBuffer sb)

This method return true if the sequence of characters in a string is same to the string buffer sequence.

boolean endsWith(String suffix)

This method checks if the string ends with the specified suffix.

boolean equals(Object anObject)

This method takes two string objects and it returns true if they are a match.

boolean equalsIgnoreCase(String anotherString)

This method compares two strings but it does not take the case into consideration.

byte getBytes()

This method converts a string into an array of bytes.

void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

This method copies a string into a character array.

int hashCode()

This method generates a hash code for a given string.

int indexOf(String str)

This method returns the index where a given substring starts.

int lastIndexOf(int ch)

This method returns the index where the specified character was last found.

int lastIndexOf(String str, int fromIndex)

This method returns the index of

a substring where it last occurred.

int length()

This method returns the length of a specified string.

boolean matches(String regex)

This method returns true if the string matches a specified regular expression.

String replace(char oldChar, char newChar)

This method replaces an existing character present in a string with a new character.

String[] split(String regex)

This method splits the string when it comes across a regular expression.

boolean startsWith(String prefix)

This method tests if the string starts with a given prefix.

String substring (int beginIndex)

This method returns a string that is a substring of the given string.

Char [] toCharArray()

This method converts a string into a character array.

String toLowerCase()

This method converts the string to lowercase.

String toUpperCase()

This method converts the string to uppercase.

Chapter 13: Arrays

Arrays are a data structure. Java allows the use of this data structure. An array is basically a fixed size sequential data structure. Array is used as a collection of data of a similar type. You can store a set of integers in a single array instead of storing them in different variables.

In this chapter we will look at how you can declare an array. How you can assign values in it and then how can you process the arrays.

Creating an Array:

Just like a variable you have to declare an array before you can use it. You will tell your compiler the type of the array and its name. The generic syntax for declaring an array is following:

---

datatype[] arrayname = {“value1”,”value2”,”value3”}

datatype arrayname[] = {“value1”,”value2”,”value3”}

You can use any one way mentioned

above to declare an array.

Now Open your Compiler and create a new class called **Candies**. Add the Following code into it.

```
public class Candies {
        String[] candies =
{"m&Ms","Yummybears","Mayfairs"};
        }
```

In the above example I have created a class named **Candies** and I have declared an array of type String in it. Then I assigned three values to that array.

You can also declare the above

mentioned array like this too.

```
public class Candies {
        String candies[] =
{"m&Ms","Yummybears","Mayfairs"};
        }
```

The type of value stored in an array depends on the type of that array for example in the following example I have declared an array of interger type. This array will only contains integer type data and nothing else.

```
public class Candies {
        int[] myIntArray =  {1,2,3};
        }
```

Using Arrays:

Now that we have seen how to declare different types of arrays and how to assign values to them let us go ahead and see how to process these array data structures. We can use the loops we read earlier in this book to traverse each index of an array and read , print or do anything else with the values inside of an array. For processing the array mostly we use for loop but you can do the same thing with other types of loops which we talked about.

Example:

```
// Your Candies Class
public class Candies {
        String[] candies =
{"m&Ms","Yummybears","Mayfairs"};

         int[] myIntArray =  {1,2,3      };

        public void ShowMyCandies(){

        for (int i = 0; i < candies.length; i++) {
        System.out.println("The name of my
Candy is " + candies[i]);
        }
          }
}
// Main Class where you instantiate your
Candies class
import java.io.Console;

public class MainClass {
	public static void main(String[] args) {
	// TODO Auto-generated method stub
	Candies myCandies = new Candies();
	myCandies.ShowMyCandies();
	}
}
```

Output:

```
The name of my Candy is m&Ms
The name of my Candy is Yummybears
The name of my Candy is Mayfairs
```

Example #2:

```
// Your Candies Class
public class Candies {
        String[] candies =
{"m&Ms","Yummybears","Mayfairs"};

         int[] myIntArray =  {1,2,3      };

        public void ShowMyCandies(){

        for (int i = 0; i < candies.length; i++) {
        System.out.println("The name of my
Candy is " + candies[i]);
        }
          }
        public void ShowMyNumbers(){

        for (int i = 0; i < myIntArray.length;
i++) {
        System.out.println("My Number Array
Contains "+ myIntArray[i]);
        }
          }
}
// Main Class where you instantiate your
Candies class
import java.io.Console;

public class MainClass {
        public static void main(String[] args) {
        // TODO Auto-generated method stub
        Candies myCandies = new Candies();
        //myCandies.ShowMyCandies();
        myCandies.ShowMyNumbers();
        }
}
```

Output:

```
My Number Array Contains 1
My Number Array Contains 2
My Number Array Contains 3
```

Helper Methods:

The Arrays also have some helper methods that are built in java to make our life easier.

public static int binarySearch(Object[] a, Object key)

This method searches the array passed to it for a value using the Binary search algorithm. If this method finds the specific value it was looking for it returns the index where it is located inside the array.

public static boolean equals(long[] a, long[] a2)

This methodstells us whether the two arrays passed to it are equal. Arrays are only equal if there type is same, they have same number of elements and the elements are same in each array. This method can be used for any other primitive type of arrays.

public static void fill(int[] a, int val)

This method sets the value of each index of an array to the value passed to this method along with the array to be effected. This method can be used for any other primitive type of array too.

public static void sort(Object[] a)

This method sorts an array of any primitive type into ascending order.

Chapter 14: Date and Time

In the **java.util** package there exists a **Date** class along with other classes written with built in features. You can use date class for the date related operation like getting the current date, Performing different functions on that date string etc.

In this chapter we will talk about dates and the operations you can perform on dates by using java. Open up your compiler and create a class named **mydateclass**.

```
public class myDateClass {

}
```

Current Date and Time:

Now let us use the class we created to get our current date and time. Go ahead and add the following method to the class.

```
public void getDateTime() {
        Date date = new Date();
        System.out.println("My Current Date
and time is: "+ date.toString());
}
```

The Final code of **myDateClass** will look like the following.

```
import java.util.Date;

public class myDateClass {

	public void getDateTime() {
	Date date = new Date();
	System.out.println("My Current Date and time is: "+ date.toString());
	}
}
```

After completing the new **myDateClass** go to the **main** class and create an object of your **myDateClass** and call the **getDateTime()** method.

```
import java.io.Console;

public class MainClass {
        public static void main(String[] args) {
        // TODO Auto-generated method stub
        myDateClass mydate = new
myDateClass();
        mydate.getDateTime();
        }
}
```

When you run the above code you will get your current date. The output of this program will be shown in the console window and will look like the following.

Output:

```
My Current Date and time is: Sat Dec 24
16:58:13 PKT 2016
```

Formatting Dates:

You can format your dates using the built in date formatting mechanisms.

Example:

```
import java.text.SimpleDateFormat;
import java.util.Date;
public class myDateClass {
        public void getDateTime() {
        Date date = new Date();
        SimpleDateFormat newFormat = new
SimpleDateFormat ("E yyyy-MM-dd 'at'
hh:mm:ss a zzz");
        System.out.println("Current Date is: " +
newFormat.format(date));
        }
}
import java.io.Console;
public class MainClass {
        public static void main(String[] args) {
        // TODO Auto-generated method stub
        myDateClass mydate = new
myDateClass();
        mydate.getDateTime();
        }
}
```

Output:

Current Date is: Sat 2016-12-24 at 05:18:02 PM PKT

You can also format the date using the **printf** statement in java.

Example:

```
import java.text.SimpleDateFormat;
import java.util.Date;
public class myDateClass {
        public void getDateTime() {
        Date date = new Date();
        String mystring =
String.format("Current Date and Time is :
%tc",date);
          System.out.printf(mystring);
        }
}
```

```
import java.io.Console;
public class MainClass {
          public static void main(String[] args) {
          // TODO Auto-generated method stub
          myDateClass mydate = new
myDateClass();
          mydate.getDateTime();
          }
}
```

Output:

```
Current Date and Time is : Sat Dec 24
17:24:26 PKT 2016
```

CHAPTER 15: METHODS

There comes a time when you need some lines of code over and over again. You can do that by just copying those lines again and again but that's not a good way as it is tedious and inefficient.

In java we have methods which can perform or execute a set of programmatic lines as many times as you ask them. You can call a method over and over again anywhere inside your code and can get the same result. Let us talk about creating a method, calling it and other fun stuff related to methods in Java.

```
public static int
methodName(int a, int b) {
// body }
```

Here,

publicstatic – these are the modifiers, we talked about them in earlier chapters.

int – This is the type of value this method will return.

methodName – This is the name of the method or function.

a, b – These literals represent the parameters of a method they could be any letter or variable.

Following is the generic template of methods for your future reference.

---

modifier return Type nameOfMethod (Parameter List) {

// method body

}

Example:

We have created a lot of methods

so far in this book. I am going to use a familiar code so that you can understand the methods better. Let us take a look at the **candies** class we created earlier

```
// Your Candies Class
public class Candies {
        String[] candies =
{"m&Ms","Yummybears","Mayfairs"};
         int[] myIntArray =  {1,2,3      };
         public void ShowMyCandies(){
                for (int i = 0; i < candies.length;
i++) {
        System.out.println("The name of my
Candy is " + candies[i]);
        }
          }
}
// Main Class where you instantiate your
Candies class
import java.io.Console;

public class MainClass {
        public static void main(String[] args) {
        // TODO Auto-generated method stub
        Candies myCandies = new Candies();
        myCandies.ShowMyCandies();
        }
}
```

Output:

```
The name of my Candy is m&Ms
The name of my Candy is Yummybears
The name of my Candy is Mayfairs
```

In the above example we have a class named **Candies**. Inside that class we declared a method called **ShowMyCandies** this method loops through the candy array and prints all the name of candies on the console screen. We call this function after making an object of the class in which this method resides. This is an **instance** method because it can only be used once you instantiate the class. We can also do this by a static method. Here's how

Example:

```
// Your Candies Class
```

```
public class Candies {
        String[] candies =
{"m&Ms","Yummybears","Mayfairs"};
        int[] myIntArray =  {1,2,3       };
        public static void ShowMyCandies(){
            for (int i = 0; i < candies.length;
i++) {
        System.out.println("The name of my
Candy is "+ candies[i]);
        }
          }
}
// Main Class where you instantiate your
Candies class
import java.io.Console;
public class MainClass {
        public static void main(String[] args) {
        // TODO Auto-generated method stub
        Candies.ShowMyCandies();
        }
}
```

Output:

```
The name of my Candy is m&Ms
The name of my Candy is Yummybears
The name of my Candy is Mayfairs
```

The highlighted part in the above example are the changes I made to the original Candies class. Just by adding a keyword "**static**" to my method definition I have changed the way it's called. Now you can call the

ShowMyCandies method directly by using the class name.

The **Void** keyword used in the above methods means that the method above doesn’t return anything.

Chapter 16: Exceptions

Exception are errors in our code. When an exception occurs it means that the flow of the program you were running has been disturbed. The exceptions cause the programs to behave abnormally so we should take some precautions in order to deal with these exceptions.

So far the code we dealt with was error free. But now in order to understand exceptions and how to deal with them, I will generate some errors in a code and then I will teach you guys how to deal with those errors so have a look at the following code

Example:

```
import java.io.Console;

public class ErrorClass {
	public static void main(String[] args) {
	String[] candies =
{"m&Ms","Yummybears","Mayfairs"};
	System.out.println(candies[4]);
	}
}
```

Output:

```
Exception in thread "main" 
java.lang.ArrayIndexOutOfBoundsException: 4
	at ErrorClass.main(ErrorClass.java:16)
```

The above error occurred because the array candies have three values stored at zero, first and second index of the array respectively. But in the second line where I try to print this array I pass it the index four which does not exist as an array is a fixed size data structure. I can correct this error just by updating the index of the array. Here’s how:

Example:

```
import java.io.Console;

public class ErrorClass {
        public static void main(String[] args) {
        String[] candies =
{"m&Ms","Yummybears","Mayfairs"};
        System.out.println(candies[1]);
        }
}
```

Output:

```
Yummybears
```

Now How to avoid this? How do we

make sure that we the programmers handle the exceptions rather than displaying the end user a lot of technical details which he doesn't understand? The answer to that is also present in Java. In Java there is exception handling. Which is done by using the try-catch blocks available in java.

```
try {
  // Protected code
}catch(ExceptionName e1) {
  // Catch block
}
```

The try block is the block where you write your code. And the catch block is the block where you handle your exceptions. I re-created the error inside a try-catch block.

Example:

```
import java.io.Console;

public class ErrorClass {
        public static void main(String[] args) {
        try{
        String[] candies =
{"m&Ms","Yummybears","Mayfairs"};
```

```
        System.out.println(candies[4]);
        }
        catch (Exception e) {
        System.out.println("Something went
wrong");
        }
  }
}
```

Output:

```
Something went wrong
```

You see? Now the code doesn’t show us the real exception but just the alternative text we added in our code. We can also see technical detail if we want to by doing this.

Example:

```
import java.io.Console;

public class ErrorClass {
          public static void main(String[] args) {
          try{
          String[] candies =
{"m&Ms","Yummybears","Mayfairs"};
          System.out.println(candies[4]);
          }
          catch (Exception e) {
          System.out.println("Something went
wrong: "+ e.getMessage());
          }
   }
}
```

Output:

```
Something went wrong: 4
```

As you can see now the exception is showing us the value that it thinks has caused the error. If you want to see more detail about the exception you can do this too

Example:

```
import java.io.Console;

public class ErrorClass {
        public static void main(String[] args) {
        try{
        String[] candies =
{"m&Ms","Yummybears","Mayfairs"};
        System.out.println(candies[4]);
        }
        catch (Exception e) {
        throw e;
        }
  }
}
```

Output:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 4
        at ErrorClass.main(ErrorClass.java:16)
```

The cause of exception can be anything. It can be a logical mistake programmer made while writing the code, or it can be because of the system you are working on or the memory stack getting full. Sometimes Java Virtual Machine (JVM) also end up being the cause of unwanted behaviors.

You can use multiple catch blocks for a single try block. The reason you would want to do that is because you expect more than one type of exception occurring in your code. So you declare multiple catch blocks and attend each error accordingly. A general template for multiple catch blocks is given below.

```
try {

  // Protected code

}catch(ExceptionType1 e1) {

  // Catch block

}catch(ExceptionType2 e2) {

  // Catch block

}catch(ExceptionType3 e3) {

  // Catch block

}
```

Chapter 17: Inner Classes

In java we can create nested classes just like nested if statements or nested functions. Inner Classes are kind of a security mechanism provided by java. Following is the general syntax for declaring an inner class

---

```
class OuterClass {

    class InnerClass {
  }

}
```

Here’s a real code example.

Example:

```
import java.io.Console;
public class Candies {
        private class chocoCandies{
        public void Hello(){
        System.out.println("Hello from the
inner Choco Candies Class");
        }
        }
        public void displayChocoClass(){
        chocoCandies mychoco = new
chocoCandies();
        mychoco.Hello();
        }
}
import java.io.Console;

public class MainClass {
        public static void main(String[] args) {
        Candies myCandies = new Candies();
        myCandies.displayChocoClass();
        }
}
```

Output:

```
Hello from the inner Choco Candies
Class
```

Conclusion

Now that we have come to an end of this small book. I hope that you have learned something. You should not stop here because there is more to java than this book. Read as much as you can and most importantly code as much as you can. Only then you will be able to master java. Happy Coding!