KARNATAKA STATE OPEN UNIVERSITY

MUKTHAGANGOTRI, MYSORE- 570 006

DEPARTMENT OF STUDIES IN INFORMATION TECHNOLOGY

M.Sc IN INFORMATION TECHNOLOGY

III SEMESTER

OOPS WITH JAVA

MSIT- 114

MSIT-114: OOPS with JAVA

Course Writers

Dr. Vinay K	**Mr. Chandrajith M**
Assistant Professor	**Assistant Professor**
JSS College of Arts, Commerce and Science	**Department of MCA**
Ooty Road,	**Maharaja Institute of Technology**
Mysore	**Mysore**

Publisher

Registrar
Karnataka State Open University
Mukthagangotri, Mysore – 570 006

Developed by Academic Section, KSOU, Mysore
Karnataka State Open University, 2014

Karnataka State Open University

Mukthagangothri, Mysore – 570 006

Third Semester M.Sc in Information Technology

OOPS with JAVA – MSIT 114

PREFACE

JAVA is one of the powerful programming language in the present field of information technology. Since from its inception into the computer world Java had carried out its own specialties. Thus by satisfying today's need for early instruction in an object-oriented language, while avoiding the complexities of C++.

In this curriculum we have a course on OOPS with Java. This study materials have been prepared by refereeing one or more sources and compiled as per the requirement. The material consists of total four modules. Each module is dived into 4 units. Therefore there will be total of sixteen units.

The very first modules covers the overview of Java programming language by covering the programming constructs. In second modules object oriented programming aspects have been introduced. Module 3 covers advanced programming technology in java i.e., multithreading concepts, applets, event handling and swings. In last module, the application of Java in web is explained. This module covers a knowledge of apache web servers, common gateway interface. MySQL and PHP related server side includes.

A rich set of examples have been given for the better understanding. The readers are instructed to implement the programs and concepts given in the material for hands on experience.

Wish you happy reading.

Module-1

UNIT 1: INTRODUCTION AND OVERVIEW OF JAVA

Structure:

1.0 OBJECTIVES

At the end of this unit you will be able to:

- Know about Java programming language
- Understand OOPS concept
- Elucidate the stryucture Java program.

1.1 Introduction

Java is a high-level, third generation programming language, like C, FORTRAN, Smalltalk, Perl, and many others. You can use Java to write computer applications that play games, store data or do any of the thousands of other things computer software can do. Compared to other programming languages, Java is most similar to C. However although Java shares much of C's syntax, it is not C. Knowing how to program in C or, better yet, C++, will certainly help you to learn Java more quickly, but you don't need to know C to learn Java. A Java compiler won't compile C code, and most large C programs need to be changed substantially before they can become Java programs. What's most special about Java in relation to other programming languages is that it lets you write special programs called applets that can be downloaded from the Internet and played safely within a web browser. Java language is called as an Object-Oriented Programming language and before beginning for Java, we have to learn the concept of OOPs (Object-Oriented Programming).

1.2 Creation of JAVA

Java is related to C++, which is a direct descendent of C. Much of the character of Java is inherited from these two languages. From C, java derives its syntax. Many of java's object-oriented features were influenced by C++. In fact, several of Java's defining characteristics come from of are responses to its predecessors. Moreover, the creation of java was deeply rooted in the process of refinement and adaptation that has been occurring in computer programming languages for the past three decades. For these reasons, this section reviews the sequence of events and forces that led up to java. As you will see, each innovation in languages design was driven by the need to solve a fundamental problem that the preceding languages could not solve. Java is no exception.

James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan conceived Java at Sun Microsystems, Inc. in 1991. It took 18 months to develop the first working version. This language was initially called "Oak" but was renamed "Java" in 1995. Between the initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995, many more people contributed to the design and evolution of the language. Bill Joy, Arthur van Hoff, Jonathan Payane, Frank Yellin, and Tim Lindholm were Key contributors to the maturing of the original prototype. Somewhat surprisingly, the original impetus for java was not the Internet! Instead, the primary motivation was the need for a platform-independent (that is, architecture-neutral) language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls. As you can probably guess, many different types of CPUs are used as controllers.

The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target. Although it is possible to compile a C++ program for just about any type of CPU, to do so requires a full C++ compiler targeted for that CPU. The problem is that compilers are expensive and time- consuming to create. An easier - and more cost – efficient – solution was needed. In an attempt of find such a solution, Gosling and others began work on a portable, platform-independent language that could be used to produce code that would run on a variety of CPUs under differing environments. This effort ultimately led to the creation of Java.

Most programmers learn early in their careers that portable programs are as elusive as they are desirable. While the quest for a way to create efficient, portable (plat form - independent) programs is nearly as old as the discipline of programming itself, it had taken a back seat to other, more pressing problems. Further, because much of the computer world had divided itself into the three competing camps of Intel, Macintosh, and UNIX, most programmers stayed within their fortified boundaries, and the urgent need for portable code was reduced. However, with the advent of the Internet and the Web, the old problem of portability returned with a vengeance. After all, the Internet consists of a diverse, distributed universe populated with many types of computers, operating systems, and CPUs. Even though many types of platforms are attached to the Internet, users would like them all to be able to run the same program. What was once an irritation but low-priority had become a high-profile necessity.

As mentioned earlier, Java derives much of its character from C and C++. This is by intent. The java designer knew that using the familiar syntax of C and echoing the object-oriented features of C++ would make their language appealing to the legions of experienced C/C++ programmers. In addition to the surface similarities, Java shares some of the other attributes that helped make C and C++ Successful. First, Java was designed, tested, and refined by real, working programmers. It is a language grounded in the needs and experiences of the people who devised it. Thus, Java is also a programmer's language. Second. Java is cohesive and logically consistent. Third, except for those constraints imposed by the Internet environment, java gives you, the programmer, and full control. If you program well, your programs reflect it. If you program poorly, your programs reflect that, too. Put differently, java is not a language with training wheels. It is a language for professional programmers.

1.3 Why JAVA

- Simple grammar - Java has a very simple grammar familiar to anyone with experience in C and C++, which must be 99.9% of programmers. The BNF for Java has about 50 rules;

that for C++, about 140. And C++ also has templates and a preprocessor in addition to the grammar. Java just got quite a bit more complex in 1.5 (excuse me, Java 5). They haven't even released a new version of the language spec yet.

- Portability - These days Java really does run well on all the popular platforms (Linux was a little behind, until Sun realized they needed them... now it's just FreeBsd, OpenBsd, and NetBsd lagging) (Too bad that Ruby, Python, Perl, Squeak don't work well on most platforms... Oh wait, they do!).
- Speed - The latest JIT compilers for Suns JVM approach the speed of C/C++ code, and in some memory allocation intensive circumstances, exceed it. (Too bad Ruby, Python, Perl, and Squeak don't even come close).
- Standard APIs - You can happily write your code knowing that the standard java.* libraries will be waiting on the client for it, assuming a recent enough version of Java is installed.
- Garbage Collection - the programmer doesn't have to worry about memory (most of the time)
- VM - see Why Are Virtual Machines Great
- Interface vs. class
- Checked Exceptions (some people hate this, but its optional) (some ppl love it)
- Single class inheritance
- Singly rooted class hierarchy (the reason that lack of templates isn't a killer)
- No Operator Overloading
- Reflection
- Inherent support for dynamic linking and loading.
- Guarantees of binary compatibility w.r.t. changes to linked code.
- Fast edit/compile/run cycle faster than what?
    - I can only attest that this makes Eclipse Ide great. It is wonderful to be able to make small changes to a class and have the recompiled class linked into the running application for immediate testing. I don't know whether this is possible in other Java environments.
- Broad industry support

- Safe semantics -- no Undefined Behavior in pure Java code
- Security model for restricted execution
- It's relatively easy to make programs that parse or produce classfiles (but not as good as lisp)
- Mr Bunny's Big Cup O' Java (By high-performance we mean adequate. By adequate we mean slow.)
    - But, hey, at least it ain't BANCStar [BancStarLanguage]
- Code is fairly transparent: except for a bit of built-in magic to do with String, code never invokes methods implicitly. (By contrast it's in general impossible in C++ to work out in isolation what a statement will do.)
- No Fragile Binary InterfaceProblem
- JavaDocsForLibraries

1.4 Byte Code

Bytecode is nothing but the intermediate representation of Java source code which is produced by the Java compiler by compiling that source code. This byte code is an machine independent code. It is not a completely a compiled code but it is an intermediate code somewhere in the middle which is later interpreted and executed by JVM. Bytecode is a machine code for JVM. But the machine code is platform specific whereas bytecode is platform independent that is the main difference between them. It is stored in .class file which is created after compiling the source code. Most developers although have never seen byte code (nor have ever wanted to see it!) One way to view the byte code is to compile your class and then open the .class file in a hex editor and translate the bytecodes by referring to the virtual machine specification. A much easier way is to utilize the command-line utility javap. The Java SDK from Sun includes the javap disassembler that will convert the byte codes into human-readable mnemonics.

The bytecode format

Bytecodes are the machine language of the Java virtual machine. When a JVM loads a class file, it gets one stream of bytecodes for each method in the class. The bytecodes streams are stored in the method area of the JVM. The bytecodes for a method are executed when that method is invoked during the course of running the program. They can be executed by interpretation, just-in-time compiling, or any other technique that was chosen by the designer of a particular JVM.

A method's bytecode stream is a sequence of instructions for the Java virtual machine. Each instruction consists of a one-byte opcode followed by zero or more operands. The opcode indicates the action to take. If more information is required before the JVM can take the action, that information is encoded into one or more operands that immediately follow the opcode.

Each type of opcode has a mnemonic. In the typical assembly language style, streams of Java bytecodes can be represented by their mnemonics followed by any operand values. For example, the following stream of bytecodes can be disassembled into mnemonics:

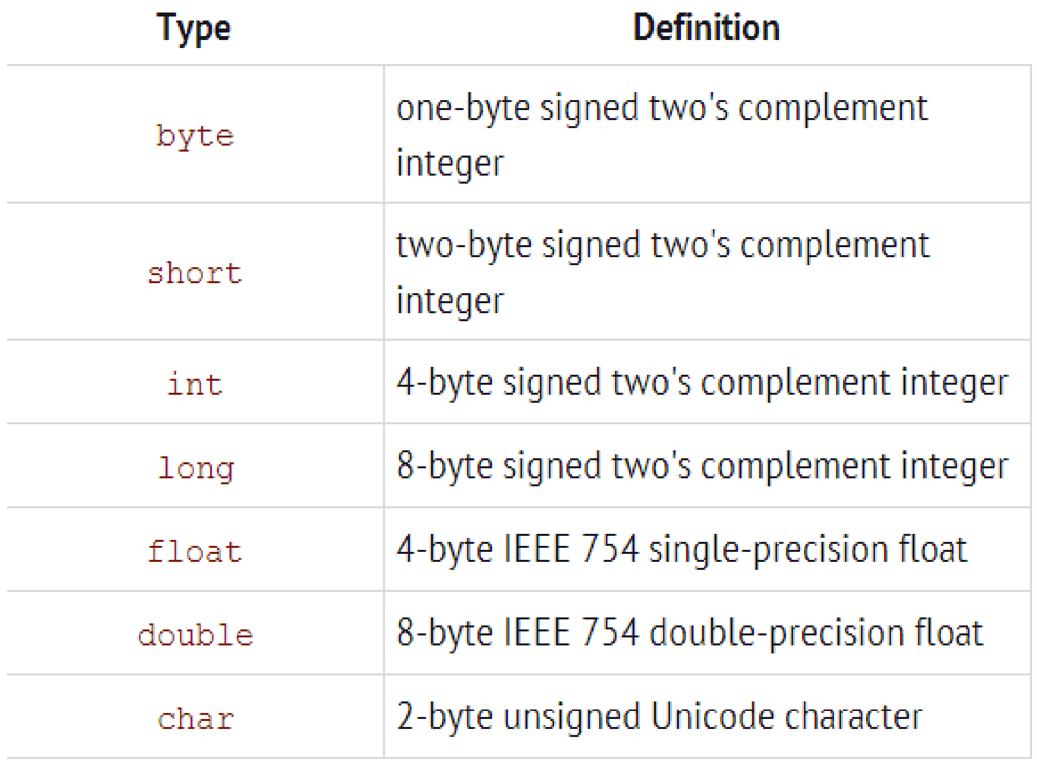
```
//Bytecode stream: 03 3b 84 00 01 1a 05 68 3b a7 ff f9

//Disassembly

iconst_0      //03

istore_0      //3b

iinc 0,1      //84 00 01

iload_0       //1a

iconst_2      //05

imul          //68

istore_0      //3b

goto -7       //a7 ff f9
```

The bytecode instruction set was designed to be compact. All instructions, except two that deal with table jumping, are aligned on byte boundaries. The total number of opcodes is small enough so that opcodes occupy only one byte. This helps minimize the size of class files that may be traveling across networks before being loaded by a JVM. It also helps keep the size of the JVM implementation small.

All computation in the JVM centers on the stack. Because the JVM has no registers for storing abitrary values, everything must be pushed onto the stack before it can be used in a calculation. Bytecode instructions therefore operate primarily on the stack. For example, in the above bytecode sequence a local variable is multiplied by two by first pushing the local variable onto the stack with the iload_0instruction, then pushing two onto the stack with iconst_2. After both integers have been pushed onto the stack, the imul instruction effectively pops the two integers off the stack, multiplies them, and pushes the result back onto the stack. The result is popped off the top of the stack and stored back to the local variable by theistore_0 instruction. The JVM was designed as a stack-based machine rather than a register-based machine to facilitate efficient implementation on register-poor architectures such as the Intel 486.

Primitive types

The JVM supports seven primitive data types. Java programmers can declare and use variables of these data types, and Java bytecodes operate upon these data types. The seven primitive types are listed in the following table:



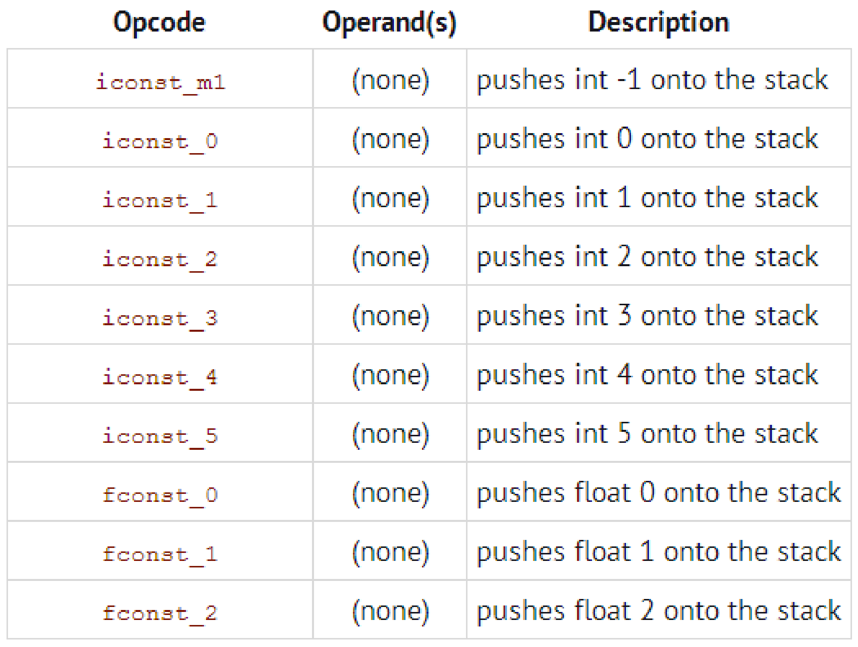
Type	Definition
`byte`	one-byte signed two's complement integer
`short`	two-byte signed two's complement integer
`int`	4-byte signed two's complement integer
`long`	8-byte signed two's complement integer
`float`	4-byte IEEE 754 single-precision float
`double`	8-byte IEEE 754 double-precision float
`char`	2-byte unsigned Unicode character

The primitive types appear as operands in bytecode streams. All primitive types that occupy more than 1 byte are stored in big-endian order in the bytecode stream, which means higher-order bytes precede lower-order bytes. For example, to push the constant value 256 (hex 0100) onto the stack, you would use the sipush opcode followed by a short operand. The short appears in the bytecode stream, shown below, as "01 00" because the JVM is big-endian. If the JVM were little-endian, the short would appear as "00 01".

Pushing constants onto the stack

Many opcodes push constants onto the stack. Opcodes indicate the constant value to push in three different ways. The constant value is either implicit in the opcode itself, follows the opcode in the bytecode stream as an operand, or is taken from the constant pool.

Some opcodes by themselves indicate a type and constant value to push. For example, the iconst_1 opcode tells the JVM to push integer value one. Such bytecodes are defined for some commonly pushed numbers of various types. These instructions occupy only 1 byte in the bytecode stream. They increase the efficiency of bytecode execution and reduce the size of bytecode streams. The opcodes that push ints and floats are shown in the following table:



Opcode	Operand(s)	Description
iconst_m1	(none)	pushes int -1 onto the stack
iconst_0	(none)	pushes int 0 onto the stack
iconst_1	(none)	pushes int 1 onto the stack
iconst_2	(none)	pushes int 2 onto the stack
iconst_3	(none)	pushes int 3 onto the stack
iconst_4	(none)	pushes int 4 onto the stack
iconst_5	(none)	pushes int 5 onto the stack
fconst_0	(none)	pushes float 0 onto the stack
fconst_1	(none)	pushes float 1 onto the stack
fconst_2	(none)	pushes float 2 onto the stack

The opcodes shown in the previous table push ints and floats, which are 32-bit values. Each slot on the Java stack is 32 bits wide. Therefore each time an int or float is pushed onto the stack, it occupies one slot.

Pushing local variables onto the stack

Local variables are stored in a special section of the stack frame. The stack frame is the portion of the stack being used by the currently executing method. Each stack frame consists of three sections -- the local variables, the execution environment, and the operand stack. Pushing a local variable onto the stack actually involves moving a value from the local variables section of the stack frame to the operand section. The operand section of the currently executing method is always the top of the stack, so pushing a value onto the operand section of the current stack frame is the same as pushing a value onto the top of the stack.

The Java stack is a last-in, first-out stack of 32-bit slots. Because each slot in the stack occupies 32 bits, all local variables occupy at least 32 bits. Local variables of type long and double, which are 64-bit quantities, occupy two slots on the stack. Local variables of type byte or short are stored as local variables of type int, but with a value that is valid for the smaller type. For example, an int local variable which represents a byte type will always contain a value valid for a byte (-128 <= value <= 127).

Each local variable of a method has a unique index. The local variable section of a method's stack frame can be thought of as an array of 32-bit slots, each one addressable by the array index. Local variables of type long or double, which occupy two slots are referred to by the lower of the two slot indexes. For example, a double that occupies slots two and three would be referred to by an index of two.

Several opcodes exist that push int and float local variables onto the operand stack. Some opcodes are defined that implicitly refer to a commonly used local variable position. For example, iload_0 loads the int local variable at position zero. Other local variables are pushed onto the stack by an opcode that takes the local variable index from the first byte following the opcode. The iload instruction is an example of this type of opcode. The first byte following iload is interpreted as an unsigned 8-bit index that refers to a local variable.

Unsigned 8-bit local variable indexes, such as the one that follows the iload instruction, limit the number of local variables in a method to 256. A separate instruction, called wide, can extend an 8-bit index by another 8 bits. This raises the local variable limit to 64 kilobytes. The wide opcode is followed by an 8-bit operand. The wide opcode and its operand can precede an instruction, such as iload that takes an 8-bit unsigned local variable index. The JVM combines the 8-bit operand of the wide instruction with the 8-bit operand of the iload instruction to yield a 16-bit unsigned local variable index.

Popping to local variables

For each opcode that pushes a local variable onto the stack there exists a corresponding opcode that pops the top of the stack back into the local variable. The names of these opcodes can be formed by replacing "load" in the names of the push opcodes with "store". Each of these opcodes moves one 32-bit value from the top of the stack to a local variable.

Type conversions

The Java virtual machine has many opcodes that convert from one primitive type to another. No operands follow the conversion opcodes in the bytecode stream. The value to convert is taken from the top of the stack. The JVM pops the value at the top of the stack, converts it, and pushes the result back onto the stack.

IMPORTANCE

The most knowledgeable C, C++ and JAVA programmers know the assembler instruction set of the processor for which they are compiling. This knowledge is crucial when debugging and doing performance and memory usage tuning. Knowing the assembler instructions that are generated by the compiler for the source code you write, helps you know how you might code differently to achieve memory or performance goals. In addition, when tracking down a problem, it is often useful to use a debugger to disassemble the source code and step through the assembler code that is executing.

1.5 Java Buzzwords

Platform Independent

Java is most known for its Platform Independent nature, one can write a program in any environment and that can be run on a variety of Hardware and Platforms. Java is compiled and run within JDK (Java Development Kit). JDK includes JRE (Java Runtime Environment) in it.

JRE is what makes java program run, this JRE can be different for different platforms but java code will run on each JRE.

Object Oriented

Java is fully object oriented or not, it is still a debate among programmers and communities. In java everything is an object except that of Primitive Types (int, char…) so this can be assumed that java is not a pure object oriented language, On the other hand Java supports Wrapper classes (Integer, Float …), so all primitive types can be represented as objects using wrapper classes, this is also true that java is an pure object oriented language.

Secure

Java is well known for its security features, java implements security API's that makes is fully secured programming language. Java security API covers Platform security, Cryptography, Public key infrastructure, Authentication and Access Control that provides the programmers a secured framework to develop applications.

Simple

Java was intended to build a simple programming language with much similar that C/C++ syntax and functionality. Java is an easy to learn and implement programming language with a clear understanding of its useful features. One can be a master in java with a quick understanding of its basic syntax and data flow across the applications.

Portable

Portability is again a big concern in Java popularity and wide usage. Being platform independent and architecture neutral makes java an efficient and idle language in portability aspects.

Robust

JVM is the root of Java, JVM is said to be the best virtual machine made till data. JVM enables a strong debugging and compilation capabilities that makes Java an error free and easy to detect errors mechanism implementation. Java is very strong in its robust part by providing an efficient mechanism against error detection at compile time and run time.

Architectural Neutral

Java compilation is a two step process, all java codes are converted to byte codes first. This makes java to run everywhere and architecturally neutral programming language.

Multithreaded

Java supports a fully integrated multithreaded environment, this feature makes the developers to utilize bandwidth and reduce idle CPU time. Multithreading makes it easy to divide the task in multiple parts to make the application more productive and safe to develop.

Distributed

Java support distribution of its code across the internet, its platform independent nature and byte code make it easy to distribute easily.

Dynamic

Java is being called more dynamic than c/c++, the reason behind its strong dynamic support is the capabilities of run time implementation and easy to adopt dynamic features.

1.6 Object-Oriented Programming

Creating a program that can use and support objects. Object-oriented programming (OOP) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs

Definitions of some of the key concepts in Object Oriented Programming (OOP)

Term	Definition
Abstract Data Type	A user-defined data type, including both attributes (its state) and methods (its behaviour). An object oriented language will include means to define new types (see class) and create instances of those classes (see object). It will also provide a number of primitive types.

Aggregation	Objects that are made up of other objects are known as aggregations. The relationship is generally of one of two types: • Composition – the object is composed of other objects. This form of aggregation is a form of code reuse. *E.g. A Car is composed of Wheels, a Chassis and an Engine* • Collection – the object contains other objects. *E.g. a List contains several Items; A Set several Members*.
Attribute	A characteristic of an object. Collectively the attributes of an object describe its state. *E.g. a Car may have attributes of Speed, Direction, Registration Number and Driver.*
Class	The definition of objects of the same abstract data type. In Java `class` is the keyword used to define new types.
Dynamic (Late) Binding	The identification at run time of which version of a method is being called (see polymorphism). When the class of an object cannot be identified at compile time, it is impossible to use static binding to identify the correct object method, so dynamic binding must be used.
Encapsulation	The combining together of attributes (data) and methods (behaviour/processes) into a single abstract data type with a public interface and a private implementation. This allows the implementation to be altered without affecting the interface.
Inheritance	The derivation of one class from another so that the attributes and methods of one class are part of the definition of another class. The first class is often referred to the base or parent class. The child is often referred to as a derived or sub-class. Derived classes are always 'a kind of' their base classes. Derived classes generally add to the attributes and/or behaviour of the base class. Inheritance is one form of object-oriented code reuse.

	E.g. Both Motorbikes and Cars are kinds of Motor Vehicles and therefore share some common attributes and behaviour but may add their own that are unique to that particular type.
Interface	The behaviour that a class exposes to the outside world; its public face. Also called its ‘contract’. In Java `interface` is also a keyword similar to class. However a Java interface contains no implementation: it simply describes the behaviour expected of a particular type of object, it doesn’t so how that behaviour should be implemented.
Member Variable	See attribute
Method	The implementation of some behaviour of an object.
Message	The invoking of a method of an object. In an object-oriented application objects send each other messages (i.e. execute each others methods) to achieve the desired behaviour.
Object	An instance of a class. Objects have state, identity and behaviour.
Overloading	Allowing the same method name to be used for more than one implementation. The different versions of the method vary according to their parameter lists. If this can be determined at compile time then static binding is used, otherwise dynamic binding is used to select the correct method as runtime.
Polymorphism	Generally, the ability of different classes of object to respond to the same message in different, class-specific ways. Polymorphic methods are used which have one name but different implementations for different classes. *E.g. Both the Plane and Car types might be able to respond to a turn Left message. While the behaviour is the same, the means of achieving it are specific to each type.*
Primitive Type	The basic types which are provided with a given object-oriented programming language. *E.g. int, float, double, char, boolean*
Static(Early)	The identification at compile time of which version of a polymorphic method

Binding	is being called. In order to do this the compiler must identify the class of an object.

1.7 Simple Java Program

- As all other programming languages, Java also has a structure.
- The first line of the C/C++ program contains include statement. For example, <stdio.h> is the header file that contains functions, like printf (), scanf () etc. So if we want to use any of these functions, we should include this header file in C/ C++ program.
- Similarly in Java first we need to import the required packages. By default java.lang.* is imported. Java has several such packages in its library. A package is a kind of directory that contains a group of related classes and interfaces. A class or interface contains methods.
- Since Java is purely an Object Oriented Programming language, we cannot write a Java program without having at least one class or object. So, it is mandatory to write a class in Java program. We should use class keyword for this purpose and then write class name.
- In C/C++, program starts executing from main method similarly in Java, program starts executing from main method. The return type of main method is void because program starts executing from main method and it returns nothing.

Sample Program:

```
//A Simple Java Program
        import java.lang.System;
        import java.lang.String;
                class Sample
                {
                        public static void main(String args[])
                        {
                                System.out.print ("Hello world");
                        }
                }
```

- Since Java is purely an Object Oriented Programming language, without creating an object to a class it is not possible to access methods and members of a class. But main method is also a method inside a class, since program execution starts from main method we need to call main method without creating an object.
- Static methods are the methods, which can be called and executed without creating objects. Since we want to call main () method without using an object, we should declare main () method as static. JVM calls main () method using its Classname.main () at the time of running the program.
- JVM is a program written by Java Soft people (Java development team) and main () is the method written by us. Since, main () method should be available to the JVM, it should be declared as public. If we don't declare main () method as public, then it doesn't make itself available to JVM and JVM cannot execute it.
- JVM always looks for main () method with String type array as parameter otherwise JVM cannot recognize the main () method, so we must provide String type array as parameter to main () method.
- A class code starts with a {and ends with a}. A class or an object contains variables and methods (functions). We can create any number of variables and methods inside the class. This is our first program, so we had written only one method called main ().
- Our aim of writing this program is just to display a string "Hello world". In Java, print () method is used to display something on the monitor.
- A method should be called by using objectname.methodname (). So, to call print () method, create an object to PrintStream class then call objectname.print () method.
- An alternative is given to cre ate an object to PrintStream Class i.e. System.out. Here, System is the class name and out is a static variable in System class. out is called a field in System class. When we call this field a PrintStream class object will be created internally. So, we can call print() method as: System.out.print ("Hello world");
- Println () is also a method belonging to PrintStream class. It throws the cursor to the next line after displaying the result.

In the above Sample program System and String are the classes present in java.lang package.

1.8 Summary

In this unit, you have learned about creation and importance of java. How oriented programming object is blend with java for reusing the code and a simple java program to begin with.

1.9 Key words

1. Byte Code: Bytecode is nothing but the intermediate representation of Java source code which is produced by the Java compiler by compiling that source code.
2. Object-oriented programming: creating a program that can use and support objects.

1.10 Unit end exercise and answers

1. Expalin about java creation?
2. Why do you need java?
3. What is byte code? Explain in detail.
4. Explain all the java Buzz words?
5. What is object oriented programming? Explain some its key concept.
6. Write a simple java program?

Answers :see 1. 1.2
2. 1.3
3. 1.4
4. 1.5
5. 1.6
6. 1.7

1.11 Reference

Teach yourself Java - Joseph O'Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP	James Lee, Brent Ware
Java Hand Book, TMH, 2002.	Patrick Naughton
Programming with Java, 2nd Edition.	Balaguruswamy
CGI Programming with Perl, 2nd Edition.	Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 2: DATA TYPES VARIABLES AND ARRAYS

Structure:

2.0 OBJECTIVES

At the end of this unit you will be able to know:

- Primitive Data Types
- Integers
- Floating Point
- Characters
- Booleans
- Literals
- Variables

- Type conversion and casting
- Automatic Type Promotion in Expressions
- Arrays

2.1 INTRODUCTION

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. A Java applet is a small application written in Java and delivered to users in the form of byte code. The user launches the Java applet from a web page and it is then executed within a Java Virtual Machine (JVM) in a process separate from the web browser itself. A Java applet can appear in a frame of the web page, a new application window, Sun's Applet Viewer or a stand-alone tool for testing applets. Java applets were introduced in the first version of the Java language in 1995.

Java applets run at very fast speeds comparable to, but generally slower than, other compiled languages such as C++. Java applets had been many times faster than JavaScript. Unlike JavaScript, Java applets have access to 3D hardware acceleration, making them well suited for non-trivial, computation intensive visualizations. As browsers have gained support for hardware accelerated graphics thanks to the canvas, as well as just in time compiled JavaScript, the speed difference has become less noticeable.

2.2 Primitive Data types

A data type is a scheme for representing values. An example is int which is the Integer, a data type. Values are not just numbers, but any manner of data that a computer can process. The data type defines the kind of data that is represented by a variable. As with the keyword class, Java data types are case sensitive.

There are two types of data types

- Primitive data type
- Non-primitive data type

In primitive data types, there are two categories

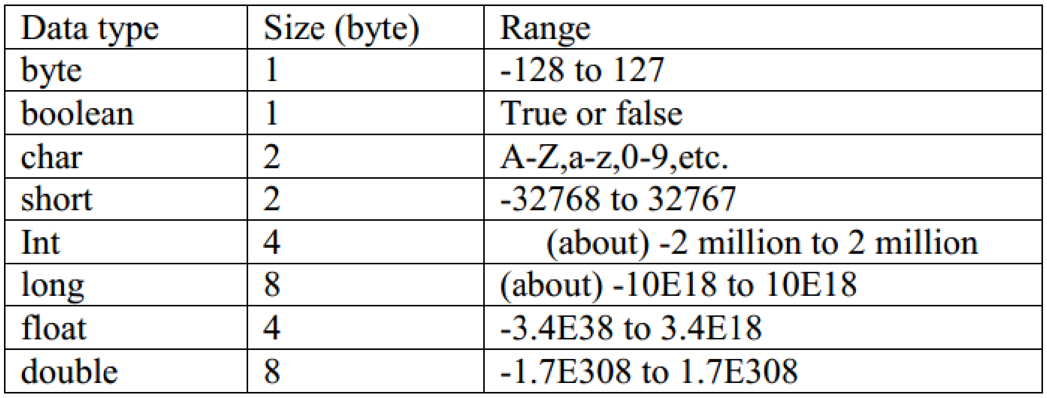
- Numeric means Integer, Floating points

- Non-numeric means Character and Boolean

In non-primitive types, there are three categories

- Classes
- Arrays
- Interface

Following table shows the data types with their size and ranges.



Data type	Size (byte)	Range
byte	1	-128 to 127
boolean	1	True or false
char	2	A-Z,a-z,0-9,etc.
short	2	-32768 to 32767
Int	4	(about) -2 million to 2 million
long	8	(about) -10E18 to 10E18
float	4	-3.4E38 to 3.4E18
double	8	-1.7E308 to 1.7E308

Fig: Data types with size and range

2.3 Integer data type:

Integer data type can hold the numbers (the number can be positive number or negative number). In Java, there are four types of integer as follows:

- byte
- short
- int
- long

byte

- Byte data type is an 8-bit signed two's complement integer.
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0

- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example: byte a = 100, byte b = -50

short

- Short data type is a 16-bit signed two's complement integer.
- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767 (inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int
- Default value is 0.
- Example: short s = 10000, short r = -20000

int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648.(-2^31)
- Maximum value is 2,147,483,647(inclusive).(2^31 -1)
- Int is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0.
- Example: int a = 100000, int b = -200000

long

- Long data type is a 64-bit signed two's complement integer.
- Minimum value is -9,223,372,036,854,775,808.(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive). (2^63 -1)
- This type is used when a wider rang e than int is needed.
- Default value is 0L.
- Example: long a = 100000L, int b = -200000L

2.4 Floating point data type

It is also called as Real number and when we require accuracy then we can use it. There are two types of floating point data type.

- float
- double

float

- Float data type is a sing le-precision 32-bit IEEE 754 floating point.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example: float f1 = 234.5f

Double

- Double data type is a double-precision 64-bit IEEE 754 floating point.
- This data type is generally used as the default data type for decimal values, generally the default choice.
- Double data type should never be used for precise values such as currency.
- Default value is 0.0d.
- Example: double d1 = 123.4

It is represent single and double precision numbers. The float type is used for single precision and it uses 4 bytes for storage space. It is very useful when we require accuracy with small degree of precision. But in double type, it is used for double precision and uses 8 bytes of storage space. It is useful for large degree of precision.

2.5 Character data type

It is used to store single character in memory. It uses 2 bytes storage space.

char

- char data type is a sing le 16-bit Unicode character.

- Minimum value is '\u0000' (or 0).
- Maximum value is '\uffff' (or 65,535 inclusive).
- Char data type is used to store any character.
- Example: char letterA ='A'

2.6 Boolean data type

It is used when we want to test a particular condition during the execution of the program. There are only two values that a boolean type can hold: true and false. Boolean type is denoted by the keyword boolean and uses only one bit of storage.

boolean

- boolean data type represents one bit of information.
- There are only two possible values: true and false.
- This data type is used for simple flag s that track true/false conditions.
- Default value is false.
- Example: boolean one = true

Following program shows the use of data types.

Program:

```
import java.io.DataInputStream;
class cc2
        {
                public static void main(String args[]) throws Exception
                        {
                                s1=new DataInputStream(System.in);
                                byte rollno;
                                int marks1,marks2,marks3;
                                float avg;
                                System.out.println("Enter roll number:");
                                rollno=Byte.parseByte(s1.readLine());
                                System.out.println("Enter marks m1, m2,m3:");
                                marks1=Integer.parseInt(s1.readLine());
```

```
                    marks2=Integer.parseInt(s1.readLine());
                    marks3=Integer.parseInt(s1.readLine());
                    avg = (marks1+marks2+marks3)/3;
                    System.out.println("Roll number is="+rollno);
                    System.out.println("Average is="+avg);
                }
        }
```

Output:

```
C:\cc>java cc2
Enter roll number:
07
Enter marks m1, m2,m3:
66
77
88
Roll number is=7
Average is=77.0
```

2.7 Java Literals

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation. Literals can be assigned to any primitive type variable. For example:

```
        byte a = 68;
        char a = 'A'
```

byte, int, long , and short can be expressed in decimal(base 10), hexadecimal(base 16) or octal(base 8) number systems as well.

Prefix 0 is used to indicate octal and prefix 0x indicates hexadecimal when using these number systems for literals. For example:

```
        int decimal = 100;
        int octal = 0144;
```

```
int hexa = 0x64;
```

String literals in Java are specified like they are in most other languages by enclosing a sequence of characters between a pair of double quotes. Examples of string literals are:

```
"Hello World"
"two\nlines"
"\"This is in quotes\""
```

String and char types of literals can contain any Unicode characters. For example:

```
char a = '\u0001';
String a = "\u0001";
```

Java language supports few special escape sequences for String and char literals as well. They are:

Notation	**Character represented**
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\"	Double quote
\'	Sing le quote
\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

2.8 Variables

In Java, all variables must be declared before they can be used. The basic form of a variable declaration is shown here:

```
type identifier [ = value][, identifier [= value] ...] ;
```

The *type* is one of Java's data types. The *identifier* is the name of the variable. To declare more than one variable of the specified type, use a comma-separated list.

Here are several examples of variable declarations of various types. Note that some include an initialization.

```
int a, b, c;                // declares three ints, a, b, and c.
int d = 3, e, f = 5;        // declares three more ints, initializing
                            // d and f.
byte z = 22;                // initializes z.
double pi = 3.14159;        // declares an approximation of pi.
char x = 'x';               // the variable x has the value 'x'.
```

This chapter will explain various variable types available in Java Language. There are three kinds of variables in Java:

- Local variables
- Instance variables
- Class/static variables

Local variables:

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

Example:

Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to this method only.

```
public class Test
```

```
        {
                public void pupAge()
                {
                        int age = 0;
                        age = age + 7;
                        System.out.println("Puppy age is : " + age);
                }
                public static void main(String args[])
                {
                        Test test = new Test();
                        test.pupAge();
                }
        }
```

This would produce the following result:

```
Puppy age is: 7
```

Example:

Following example uses *age* without initializing it, so it would give an error at the time of compilation.

```
public class Test
        {
                public void pupAge()
                {
                        int age;
                        age = age + 7;
                        System.out.println("Puppy age is : " + age);
                }
                public static void main(String args[])
                {
                        Test test = new Test();
```

```
                              test.pupAge();
                    }
          }
```

This would produce the following error while compiling it:

```
Test.java:4:variable number might not have been initialized
age = age + 7;
      ^
1 error
```

Instance variables:

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object references it is null. Values can be assigned during the declaration or within the constructor.

- Instance variables can be accessed directly by calling the variable name inside the class. However within static methods and different class (when instance variables are given accessibility) should be called using the fully qualified name. *ObjectReference.VariableName*.

Example:

```
import java.io.*;
public class Employee
{
// this instance variable is visible for any child class.
public String name;
// salary  variable is visible in Employee class only.
private double salary;
// The name variable is assigned in the constructor.
public Employee (String empName)
{
name = empName;
}
// The salary variable is assigned a value.
public void setSalary(double empSal)
{
salary = empSal;
}
// This method prints the employee details.
public void printEmp()
{
System.out.println("name  : " + name );
System.out.println("salary :" + salary);
}
public static void main(String args[])
{
Employee empOne = new Employee("Ransika");
empOne.setSalary(1000);
empOne.printEmp();
}
}
```

This would produce the following result:

```
name  : Ransika
salary :1000.0
```

Class/static variables

- Class variables also known as static variables are declared with the *static* keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final and static. Constant variables never change from their initial value.
- Static variables are stored in static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name.ClassName.VariableName.

- When declaring class variables as public static final, then variables names (constants) are all in upper case. If the static variables are not public and final the naming syntax is the same as instance and local variables.

Example:

```
import java.io.*;
public class Employee{
                    // salary  variable is a private static variable
                    private static double salary;
                    // DEPARTMENT is a constant
                    public static final String DEPARTMENT = "Development ";
                    public static void main(String args[]){
                    salary = 1000;
                    System.out.println(DEPARTMENT+"average salary:"+salary);
                                                            }
                        }
```

This would produce the following result:

```
Development average salary: 1000
```

2.9 Type conversion and casting

Java supports two types of castings – primitive data type casting and reference type casting. Reference type casting is nothing but assigning one Java object to another object. It comes with very strict rules and is explained clearly in Object Casting. Now let us go for data type casting. Java data type casting comes with 3 types.

1. Implicit casting
2. Explicit casting
3. Boolean casting.

Implicit casting (widening conversion)

A data type of lower size (occupying less memory) is assigned to a data type of higher size. This is done implicitly by the JVM. The lower size is widened to higher size. This is also named as automatic type conversion.

Examples:

```
int x = 10;                  // occupies 4 bytes
double y = x;                // occupies 8 bytes
System.out.println(y);       // prints 10.0
```

In the above code 4 bytes integer value is assigned to 8 bytes double value.

Explicit casting (narrowing conversion)

A data type of higher size (occupying more memory) cannot be assigned to a data type of lower size. This is not done implicitly by the JVM and requires explicit casting; a casting operation to be performed by the programmer. The higher size is narrowed to lower size.

```
double x = 10.5;         // 8 bytes
int y = x;               // 4 bytes ;  raises compilation error
```

In the above code, 8 bytes double value is narrowed to 4 bytes int value. It raises error. Let us explicitly type cast it.

```
double x = 10.5;
int y = (int) x;
```

The double x is explicitly converted to int y. The thumb rule is, on both sides, the same data type should exist.

Boolean casting

A boolean value cannot be assigned to any other data type. Except boolean, all the remaining 7 data types can be assigned to one another either implicitly or explicitly; but boolean cannot. We say, boolean is incompatible for conversion. Maximum we can assign a boolean value to another boolean.

Following raises error.

```
boolean x = true;
int y = x;               // error
```

```
boolean x = true;
int y = (int) x;          // error
```

byte –> short –> int –> long –> float –> double

In the above statement, left to right can be assigned implicitly and right to left requires explicit casting. That is, byte can be assigned to short implicitly but short to byte requires explicit casting.

2.10 Automatic Type Promotion in Expressions

In addition to assignments, there is another place where certain type conversions may occur: in expressions. To see why, consider the following. In an expression, the precision required of an intermediate value will sometimes exceed the range of either operand.

For example, examine the following expression:

```
byte a = 40;
byte b = 50;
byte c = 100;
int d = a * b / c;
```

The result of the intermediate term a*b easily exceeds the range of either of its byte operands. To handle this kind of problem, Java automatically promotes each byte or short operand to int when evaluating an expression. This means that the sub expression a*b is performed using integers-not bytes. Thus, 2000, the result of the intermediate expression, 50*40, is legal even though a and b are both specified as type byte. As useful as the automatic promotions are, they can cause confusing compile-time errors.

For example, this seemingly correct code causes a problem:

```
byte b=50;
b=b*2;          //Error! Cannot assign an int to a byte
```

The code is attempting to store 50*2, a perfectly valid byte value, back into a byte variable. However, because the operands were automatically promoted to int when the expression was evaluated, the result has also been promoted joint. Thus, the result of the expression is now of type int, which cannot be assigned to a byte without the use of a cast. This is true even if, as in this particular case, the value being assigned would still fit in the target type.

In cases where you understand the consequences of overflow, you should use an explicit cast, such as

```
        byte b=50;
        b=byte (b*2);
```

which yields the correct value of 100.

The Type Promotion Rules

In addition to the elevation of bytes and shorts to int, Java defines several type promotion rules that apply to expressions. They are as follows. First, all byte and short values are promoted to int, as just described. Then, if one operand is a long, the whole expression is promoted to long. If one operand is a float, the entire expression is promoted to float. If any of the operands is double, the result is double.

The following program demonstrates how each value in the expression gets promoted to match the second argument to each binary operator:

```
class Promote
{
        public static void main(String args[])
        {
                byte b = 42;
                char c = 'a';
                short s = 1024;
                int i = 50000;
                float f = 5.67f;
                double d = .1234;
                double result = (f * b) + (i / c) - (d * s);
                System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
                System.out.println("result = " + result);
        }
}
```

Let's look closely at the type promotions that occur in this line from the program:

double result = (f * b) + (i / c) - (d * s);

In the first sub expression, f * b, b is promoted to a float and the result of the sub expression is float. Next, in the sub expression i / c, c is promoted to int, and the result is of type int. Then, in d * s, the value of s is promoted to double, and the type of the sub expression is double. Finally, these three intermediate values, float, int, and double, are considered. The outcome of float plus an int is a float. Then the resultant float minus the last double is promoted to double, which is the type for the final result of the expression.

2.11 Arrays

Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
dataType[] arrayRefVar;   // preferred way.
or
dataType arrayRefVar[];  //  works but not preferred way.
```

Example:

The following code snippets are examples of this syntax:

```
double[] myList;         // preferred way.
or
```

```
double myList[];         //  works but not preferred way.
```

Creating Arrays

You can create an array by using the new operator with the following syntax:

```
arrayRefVar = new dataType[arraySize];
```

The above statement does two things:

- It creates an array using new dataType[arraySize];
- It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

```
dataType[] arrayRefVar = new dataType[arraySize];
```

Alternatively you can create arrays as follows:

```
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

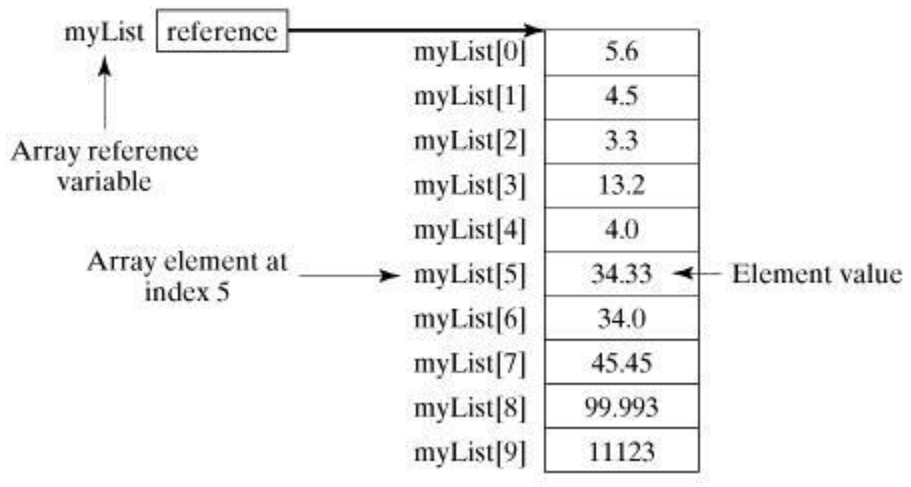
The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayRefVar.length-1.

Example:

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:

```
double[] myList = new double[10];
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Processing Arrays

When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

Example:

Here is a complete example of showing how to create, initialize and process arrays:

```
public class TestArray {

  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};

    // Print all the array elements
    for (int i = 0; i < myList.length; i++) {
      System.out.println(myList[i] + " ");
    }
    // Summing all elements
    double total = 0;
       for (int i = 0; i < myList.length; i++) {
      total += myList[i];
    }
    System.out.println("Total is " + total);
    // Finding the largest element
    double max = myList[0];
    for (int i = 1; i < myList.length; i++) {
      if (myList[i] > max) max = myList[i];
    }
    System.out.println("Max is " + max);
  }
}
```

This would produce the following result:

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

The foreach Loops

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

Example:

The following code displays all the elements in the array myList:

```
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (double element: myList) {
         System.out.println(element);
      }
   }
}
```

This would produce the following result:

```
1.9
2.9
3.4
3.5
```

Passing Arrays to Methods

Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an int array:

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

You can invoke it by passing an array. For example, the following statement invokes the printArray method to display 3, 1, 2, 6, 4, and 2:

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Returning an Array from a Method

A method may also return an array. For example, the method shown below returns an array that is the reversal of another array:

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
    result[j] = list[i];
  }
  return result;
}
```

2.12 Summary

In this unit, you have learned the importance of data types and array. There are different type of data types in primitive data type.Integer type, float type character type, Boolean type are some of the data type discussed in this unit. Literals, Variables, Type conversion and casting, Automatic Type Promotion in Expressions are some of the other topic discussed.Arrays are important topic in programming which has discussed in this unit.

2.13 Key words

1. Primitive Types: A class at the top of a hierarchy of other classes can define only the behavior and attributes common to all the classes.
2. Explicit casting (narrowing conversion): A data type of higher size (occupying more memory) cannot be assigned to a data type of lower size.
3. Implicit casting (widening conversion): A data type of lower size (occupying less memory) is assigned to a data type of higher size.

2.14 Unit end exercise and answers

1. What is a Primitive type? Explain in detail.
2. Explain each primitive type in detail.
3. What are Literals? Explain.
4. What are variables? Explain.
5. Explain about local variable and instance variable.
6. What is Type conversion and casting? Explain about different type conversion and casting discussed in this unit.
7. What is Automatic Type Promotion in Expressions? Explain about Automatic promotion rules.
8. What is Array? Explain in detail.
9. Explain how to creat an Array, Declare an Array and processing an Array.
10. Explain Passing Arrays to Methods and returning an Array from method.

Answers :see 1. 2.2
2. 2.3, 2.4, 2.5, 2.6
3. 2.7
4. 2.8
5. 2.8
6. 2.9
7. 2.10
8. 2.11
9. 2.11
10. 2.11

2.15 Reference

Teach yourself Java - Joseph O'Neil
Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 2002. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 3: Operators.

Structure:

3.0 OBJECTIVES

At the end of this unit you will be able to know:

- Arithmetic operators
- Bitwise operators
- Relational operators
- logical operators
- Assignment operators
- ? operators
- Operator Precedence
- Using parenthesis

3.1 INTRODUCTION

Java carries a broad range of operators. An operator is symbols that specify operation to be performed may be certain mathematical and logical operation. Operators are used in programs to operate data and variables. They frequently form a part of mathematical or logical expressions. Categories of operators are as follows:

1) Arithmetic operators
2) Bit wise operators
3) Logical operators
4) Relational operators
5) Assignment operators
6) Conditional operators

3.2 Arithmetic operators

Arithmetic operators are used to make mathematical expressions and the working out as same in algebra. Java provides the fundamental arithmetic operators. These can operate on built in data type of Java. Following table shows the details of operators.

Operator	**Importance/ significance**
+	Addition
-	Subtraction
/	Division
*	Multiplication
%	Modulo division or remainder

Now the following programs show the use of arithmetic operators.

"+" operator in Java:

In this program, we have to add two integer numbers and display the result.

```
class AdditionInt
        {
                public static void main (String args[])
                {
                        int a = 6;
                        int b = 3;
                        System.out.println("a = " + a);
                        System.out.println("b =" + b);
                        int c = a + b;
                        System.out.println("Addition = " + c);
                }
        }
```

Output:

```
a= 6
b= 3
Addition=9
```

"-" operator in Java:

```
class SubstractionInt
        {
                public static void main (String args[])
                {
                        int a = 6;
                        int b = 3;
                        System.out.println("a = " + a);
                        System.out.println("b =" + b);
                        int c = a - b;
                        System.out.println("Subtraction= " + c);
                }
        }
```

Output:

```
a=6
```

```
b=3
Subtraction=3
```

“*” operator in Java:

```
class MultiplicationInt
        {
                public static void main (String args[])
                {
                        int a = 6;
                        int b = 3;
                        System.out.println("a = " + a);
                        System.out.println("b =" + b);
                        int c = a * b;
                        System.out.println("Multiplication= " + c);
                }
        }
```

Output:

```
a=6
b=3
Multiplication=18
```

“/” operator in Java:

```
class DivisionInt
        {
                public static void main (String args[ ])
                {
                        int a = 6;
                        int b = 3;
                        System.out.println("a = " + a);
                        System.out.println("b =" + b);
                        c = a / b;
                        System.out.println("division=" + c);
                }
        }
```

Output:

```
a=6
b=3
Division=3
```

Remainder or modulus operator (%) in Java

```
class Remainderoptr
        {
                public static void main (String args[ ])
                {
                        int a = 6;
                        int b = 3;
                        System.out.println("a = " + a);
                        System.out.println("b =" + b);
                        c = a % b;
                        System.out.println("remainder=" + c);
                }
        }
```

Output:

```
a=6
b=3
Remainder=0
```

- When both operands in the expression are integers then the expression is called Integer expression and the operation is called Integer arithmetic.
- When both operands in the expression are real then the expression is called Real expression and the operation is called Real arithmetic.
- When one operand in the expression is integer and other is float then the expression is called Mixed Mode Arithmetic expression and the operation is called Mixed Mode Arithmetic operation.

As we learn the Arithmetic operation on integer data and store data in integer variable. But the following program shows the use of operators with integer data and store data in float variable.

Program: write a program to calculate average of three numbers.

```
class Avg1
        {
                public static void main(String args[])
                {
                        int a=3;
                        int b=3;
                        int c=4;
                        int avg;
                        avg=a+b+c;
                        avg=avg/3;
                        System.out.println("Avg of three numbers=" +avg);
                }
        }
```

Output:

```
Avg of three numbers=3
```

3.3 Bit Wise Operators

Bit wise operator execute single bit of their operands. Following table shows bit wise operator:

Operator	**Importance/ significance**
|	Bitwise OR
&	Bitwise AND
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^	Bitwise Exclusive OR
<<	Left shift

>>	Right shift
~	One‘s complement

Now the following program shows the use of operators.

```
class Boptr1
        {
                public static void main (String args[])
                {
                        int a = 4;
                        int b = a<<3;
                        System.out.println("a = " +a);
                        System.out.println("b = " +b);
                }
        }
```

Output:

```
a =4
b =16
```

3.4 Relational Operators

When evaluation of two numbers is performed depending upon their relation, assured decisions are made. The value of relational expression is either true or false.

If A=7 and A < 10 is true while 10 < A is false. Following table shows the details of operators.

Operator	Importance/ significance
>	Greater than
<	Less than
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

Now, following examples show the actual use of operators.

1) If 10 > 30 then result is false

2) If 40 > 17 then result is true
3) If 10 >= 300 then result is false
4) If 10 <= 10 then result is true

Now the following program shows the use of operators.

(1) Program 1:

```
class Reloptr1
        {
                public static void main (String args[])
                {
                        int a = 10;
                        int b = 30;
                        System.out.println("a>b = " +(a>b));
                        System.out.println("a<b = "+(a<b));
                        System.out.println("a<=b = "+(a<=b));
                }
        }
```

Output:

```
a>b = false
a<b = true
a<=b = true
```

(2) Program 3

```
class Reloptr3
        {
                public static void main (String args[])
                {
                        int a = 10;
                        int b = 30;
                        int c = 30;
                        System.out.println("a>b = " +(a>b));
                        System.out.println("a<b = "+(a<b));
                        System.out.println("a<=c = "+(a<=c));
                        System.out.println("c>b = " +(c>b));
                        System.out.println("a<c = "+(a<c));
                        System.out.println("b<=c = "+(b<=c));
                }
        }
```

Output:

```
a>b = false
a<b = true
a<=c = true
c>b = true
a<c = true
b<=c = true
```

3.5 Logical operators

When we want to form compound conditions by combining two or more relations, then we can use logical operators. Following table shows the details of operators.

Operators	**Importance/ significance**
||	Logical – OR
&&	Logical –AND
!	Logical –NOT

The logical expression defers a value of true or false. Following table shows the truth table of Logical-OR and Logical-AND. Truth table for Logical-OR operator:

Operand1	Operand3	Operand1 || Operand3
T	T	T
T	F	T
F	T	T
F	F	F

T - True

F - False

Truth table for Logical-AND operator:

Operand1	Operand3	Operand1 && Operand3
T	T	T
T	F	F
F	T	F
F	F	F

Now the following program shows the use of Logical operators.

```
class LogicalOptr
        {
                public static void main (String args[])
                {
                        boolean a = true;
                        boolean b = false;
                        System.out.println("a||b = " +(a||b));
                        System.out.println("a&&b = "+(a&&b));
                        System.out.println("a! = "+(!a));
                }
        }
```

Output:

```
a||b = true
a&&b = false
a! = false
```

3.6 Assignment Operators

Assignment Operators is used to assign the value of an expression to a variable and is also called as Shorthand operators.

Variable_name binary_operator = expression

Following table show the use of assignment operators.

Simple Assignment Operator	Statement with shorthand Operators
A=A+1	A+=1
A=A-1	A-=1
A=A/(B+1)	A/=(B+1)
A=A*(B+1)	A*=(B+1)
A=A/C	A/=C
A=A%C	A%=C

These operators avoid repetition, easier to read and write. Now the following program shows the use of operators.

```
class Assoptr
        {
                public static void main (String args[])
                {
                        int a = 10;
                        int b = 30;
                        int c = 30;
                        a+=1;
                        b-=3;
                        c*=7;
                        System.out.println("a = " +a);
                        System.out.println("b = "+b);
                        System.out.println("c = "+c);
                }
        }
```

Output:

```
a = 11
b = 18
c = 310
```

3.7 ? Operators

The character pair ?: is a ternary operator of Java, which is used to construct conditional expressions of the following form:

Expression1 ? Expression3 : Expression3

The operator ? : works as follows:

Expression1 is evaluated if it is true then Expression3 is evaluated and becomes the value of the conditional expression. If Expression1 is false then Expression3 is evaluated and its value becomes the conditional expression.

For example:

```
A=3;
B=4;
C= (A<B)? A:B;
C= (3<4)? 3:4;
C=4
```

Now the following program shows the use of operators.

```
class Coptr
{
        public static void main (String args[])
        {
                int a = 10;
                int b = 30;
                int c;
                c=(a>b)?a:b;
                System.out.println("c = " +c);
                c=(a<b)?a:b;
                System.out.println("c = " +c);
        }
}
```

Output:

```
c = 30
```

```
c = 10
```

program3: Write a program to check whether number is positive or negative.

```
class PosNeg
        {
                public static void main(String args[])
                {
                        int a=10;
                        int flag=(a<0)?0:1;
                        if(flag==1)
                        System.out.println("Number is positive");
                        else
                        System.out.println("Number is negative");
                }
        }
```

Output:

```
Number is positive
```

3.8 Operator precedence

Operator precedence defines the order in which various operators are evaluated. (In fact, you may remember "order of operations" from secondary school algebra).

As an example, let's say we have the following line of Java code:

```
int x = 4 + 3 * 5;
```

The variable x gets the value of evaluating the expression 4 + 3 * 5. There are a couple of ways to evaluate that expression, though: We can either perform the addition first or perform the multiplication first.

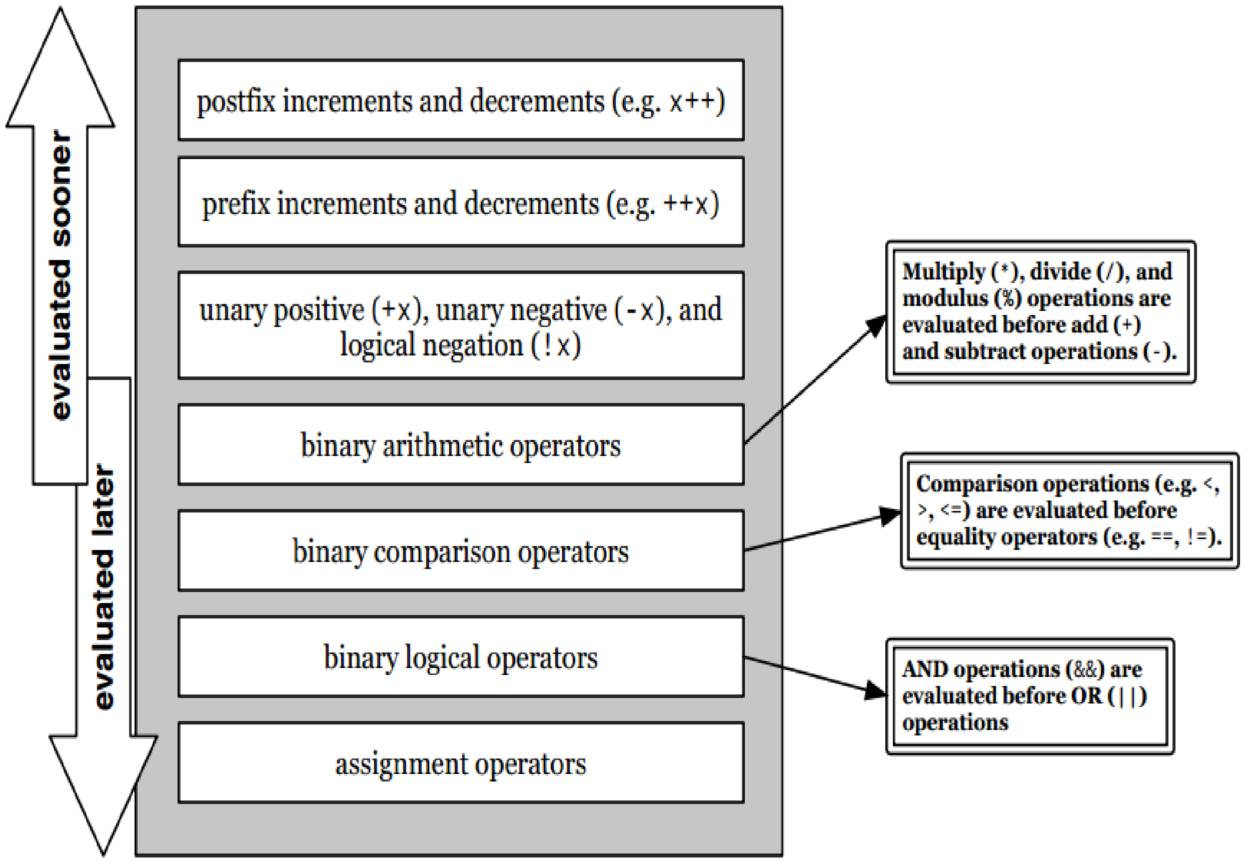
By choosing which operation to perform first, we are actually choosing between two different expressions:

```
(4 + 3) * 5 == 35
4 + (3 * 5) == 19
```

In the absence of parentheses, which choice is appropriate? Programming languages answer this question by defining precedence levels for each operator, indicating which is to be performed first. In the case of Java, multiplication takes precedence over addition; therefore, x will get the value 19.For arithmetic expressions, multiplication and division are evaluated before addition and subtraction, just like in mathematics. Of course, just as you might in a math class, you can always parenthesize Java expressions to indicate which are to be evaluated first.

Sensible use of parentheses will make your programs easier to read even if your expressions all use the standard evaluation order.

This sheet shows the operator precedence's for the Java operators you'll be using most frequently in CS 302. On the reverse of this sheet is a chart of the precedence levels for every operator in the Java language, provided in case,


evaluated sooner
evaluated later
postfix increments and decrements (e.g. x++)
prefix increments and decrements (e.g. ++x)
unary positive (+x), unary negative (-x), and logical negation (!x)
binary arithmetic operators
binary comparison operators
binary logical operators
assignment operators
Multiply (*), divide (/), and modulus (%) operations are evaluated before add (+) and subtract operations (-).
Comparison operations (e.g. <, >, <=) are evaluated before equality operators (e.g. ==, !=).
AND operations (&&) are evaluated before OR (||) operations

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left

Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> > < <<	Left to right
Relational	> >= < < =	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	A	Left to right
Bitwise OR	1	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= A = | =	Right to left
Comma	,	Left to right

3.9 Using Parentheses

Sometimes the default order of evaluation isn't what you want. For instance, the formula to change a Fahrenheit temperature to a Celsius temperature is C = (5/9) (F - 32) where C is degrees Celsius and F is degrees Fahrenheit. You must subtract 32 from the Fahrenheit temperature before you multiply by 5/9, not after. You can use parentheses to adjust the order much as they are used in the above formula. The next program prints a table showing the conversions from Fahrenheit and Celsius between zero and three hundred degrees Fahrenheit every twenty degrees.

```
// Print a Fahrenheit to Celsius table
class FahrToCelsius
        {
        public static void main (String args[])
                {
                        // lower limit of temperature table
                        double lower = 0.0;
                        // upper limit of temperature table
                        double upper = 300.0;
                        // step size
                        double step  = 20.0;
                        double fahr = lower;
                        while (fahr <= upper)
                              {
                                    double celsius = (5.0 / 9.0) * (fahr-32.0);
                                    System.out.println(fahr + " " + celsius);
                                    fahr = fahr + step;
                              }
                  }
            }
```

3.10 Summary

In this unit, you have learned different type of operator and also how important operators are for programming. Arithmetic, Bitwise, Relational, logical, Assignment, ? Operators are some of the operator discussed in this unit. Operator precedence and how to priorities operators while using parenthesis are some of the other topic discussed in this unit.

3.11 Key words

Operator precedence : defines the order in which various operators are evaluated.

3.12 Unit end exercise and answers

1. What is Arithmetic operator? Explain
2. What is Bitwise operator? Explain
3. What is Relational operator? Explain
4. What logical operator? Explain
5. What is Assignment operator? Explain

6. What is? Operator? Explain
7. How Operator Precedenceis done? Explain
8. Using parenthesis how to slove a expression? Explain

Answers :see 1. 3.2

2. 3.3

3. 3.4

4. 3.5

5. 3.6

6. 3.7

7. 3.8

8. 3.9

3.13 Reference

Teach yourself Java - Joseph O'Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 2002. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 4: Control Statements

Structure:

4.0 OBJECTIVES

At the end of this unit you will be able to know:

- Control structure
- Selection
- Iteration
- Jump Statements.

4.0 INTODUCTION

In Java, program is a set of statements and which are executed sequentially in order in which they appear. In that statements, some calculation have need of executing with some conditions and for that we have to provide control to that statements. In other words, Control statements are used to provide the flow of execution with condition.

4.1 Control Structure

In java program, control structure is can divide in three parts:

- Selection statement
- Iteration statement
- Jumps in statement

4.2 Selection Statement

Selection statement is also called as Decision making statements because it provides the decision making capabilities to the statements. In selection statement, there are two types:

- if statement
- switch statement

These two statements are allows you to control the flow of a program with their conditions.

if Statement

The "if statement" is also called as conditional branch statement. It is used to program execution through two paths. The syntax of "if statement" is as follows:

Syntax:

```
if (condition)
{
        Statement 1;
        Statement 2;
        ...
}
else
{
        Statement 3;
        Statement 4;
        ...
}
```

The “if statement” is a commanding decision making statement and is used to manage the flow of execution of statements. The “if statement” is the simplest one in decision statements. Above syntax is shows two ways decision statement and is used in combination with statements.

Simple if statement:

Syntax:

```
If (condition)
{
Statement block;
}
Statement-a;
```

In statement block, there may be single statement or multiple statements. If the condition is true then statement block will be executed. If the condition is false then statement block will omit and statement-a will be executed.

The if…else statement:

Syntax:

```
If (condition)
        {
                True - Statement block;
        }
else
        {
                False - Statement block;
        }
Statement-a;
```

If the condition is true then true - statement block will be executed. If the condition is false then false - statement block will be executed. In both cases the statement-a will always executed. Following program shows the use of if statement.

Program: write a program to check whether the number is positive or negative.

```
import java.io.*;
```

```
class NumTest
        {
                public static void main (String[] args) throws IOException
                {
                        int Result=11;
                        System.out.println("Number is"+Result);
                        if ( Result < 0 )
                        {
                                System.out.println("The number "+ Result +" is negative");
                        }
                        else
                        {
                                System.out.println("The number "+ Result +" is positive");
                        }
                }
        }
```

Output:

```
C:\MCA>java NumTest
Number is 11
The number 11 is positive
```

Nesting of if-else statement

Syntax:

```
if (condition1)

        {

                If(condition2)

                {
```

```
                Statement block1;
            }
            else
            {
                Statement block2;
            }
    }
else
    {
            Statement block3;
    }
Statement 4;
```

If the condition1 is true then it will be goes for condition2. If the condition2 is true then statement block1 will be executed otherwise statement2 will be executed. If the condition1 is false then statement block3 will be executed. In both cases the statement4 will always executed.

F**or example:** Write a program to find out greatest number from three numbers.

```
class greatest
    {
        public static void main(String args[])
        {
            int a=10;
            int b=20;
            int c=3;
            if(a>b)
```

```
                        {
                                if(a>c)
                                {
                                        System.out.println("a is greater number");
                                }
                                else
                                {
                                        System.out.println("c is greater number");
                        }
                }
                else
                {
                        if(c>b)
                        {
                                System.out.println("c is greater number");
                        }
                        else
                        {
                                System.out.println("b is greater number");
                        }
                }
        }
}
```

Output:

```
C:\MCA>java greatest
b is greater number
```

Switch statement:

In Java, switch statement check the value of given variable or statement against a list of case values and when the match is found a statement-block of that case is executed. Switch statement is also called as multiway decision statement.

Syntax:

```
switch(condition)// condition means case value
{
        case value-1:statement block1;break;
        case value-2:statement block2;break;
        case value-3:statement block3;break;
        …
        default:statement block-default;break;
}
statement a;
```

The condition is byte, short, character or an integer. value-1,value-2,value-3,…are constant and is called as labels. Each of these values be matchless or unique with the statement. Statement block1, Statement block2, Statement block3,..are list of statements which contain one statement or more than one statements. Case label is always end with “:” (colon).

Program: write a program for bank account to perform following operations.

-Check balance

For example:

```
import java.io.*;
class bankac
{
    public static void main(String args[]) throws Exception
    {
        int bal=20000;
        int ch=Integer.parseInt(args[0]);
        System.out.println("Menu");
        System.out.println("1:check balance");
```

```
		System.out.println("2:withdraw amount ");
		System.out.println("3:deposit amount ");
		System.out.println("4:exit");
		switch(ch)
		{
			case 1:System.out.println("Balance is:"+bal);
			break;
			case 2:int w=Integer.parseInt(args[1]);
			if(w>bal)
			{
				System.out.println("Not sufficient balance");
			}
			bal=bal-w;
			System.out.println("Balance is"+bal);
			break;
			case 3:int d=Integer.parseInt(args[1]);
			bal=bal+d;
			System.out.println("Balance is"+bal);
			break;
			default:break;
		}
	}
}
```

Output:

```
C:\MCA>javac bankac.java
C:\MCA>java bankac 1
Menu
1: check balance
2: withdraw amount
3: deposit amount
4: exit
```

```
Balance is: 20000
```

4.3 Iteration Statement

The process of repeatedly executing a statements and is called as looping. The statements may be executed multiple times (from zero to infinite number). If a loop executing continuous then it is called as Infinite loop. Looping is also called as iterations. In Iteration statement, there are three types of operation:

- for loop
- while loop
- do-while loop

for loop

The for loop is entry controlled loop. It means that it provide a more conscious loop control structure.

Syntax:

```
for(initialization;condition;iteration)
{
        Statement block;
}
```

When the loop is starts, first part(i.e. initialization) is execute. It is just like a counter and provides the initial value of loop. But the thing is, Initialization is executed only once. The next part(i.e. condition) is executed after the initialization. The important thing is, this part provide the condition for looping. If the condition will satisfying then loop will execute otherwise it will terminate. Third part (i.e. iteration) is executed after the condition. The statements that incremented or decremented the loop control variables.

For example:

```
import java.io.*;
class number
        {
                public static void main(String args[]) throws Exception
        {
            int i;
            System.out.println("list of 1 to 10 numbers");
            for(i=1;i<=10;i++)
            {
                System.out.println(i);
            }
        }
    }
```

Here we declare i=1 and then it check the condition that if i<10 then only loop will be executed. After first iteration the value of i will print and it will incremented by 1. Now the value of i=2 and again we have to check the condition and value of i will print and then increment I by 1 and so on.

while loop

The while loop is entry controlled loop statement. The condition is evaluated, if the condition is true then the block of statements or statement block is executed otherwise the block of statement is not executed.

Syntax:

```
while(condition)
{
    Statement block;
}
```

For example: Write a program to display 1 to 10 numbers using while loop.

```
import java.io.*;
class number
    {
        public static void main(String args[]) throws Exception
        {
            int i=1;
            System.out.println("list of 1 to 10 numbers");
```

```
                  while(i<=10)
                  {
                        System.out.println(i);
                        i++;
                  }
            }
      }
```

do-while loop

In do-while loop, first attempt of loop should be execute then it check the condition. The benefit of do-while loop/statement is that we get entry in loop and then condition will check for very first time. In while loop, condition will check first and if condition will not satisfied then the loop will not execute.

Syntax:

```
do
{
      Statement block;
}
      while(condition);
```

In program, when we use the do-while loop, then in very first attempt, it allows us to get enter in loop and execute that loop and then check the condition. Following program show the use of do-while loop.

For example: Write a program to display 1 to 10 numbers using do-while loop.

```
import java.io.*;
class number
{
      public static void main(String args[]) throws Exception
      {
            int i=1;
            System.out.println("list of 1 to 10 numbers");
            do
```

```
          {
                    System.out.println(i);
                    i++;
          }
     while(i<=10);
     }
}
```

4.4 Jumps in statement

Statements or loops perform a set of operations continually until the control variable will not satisfy the condition. but if we want to break the loop when condition will satisfy then Java give a permission to jump from one statement to end of loop or beginning of loop as well as jump out of a loop. “**break**” keyword use for exiting from loop and “continue” keyword use for continuing the loop. Following statements shows the exiting from loop by using “break” statement.

do-while loop

```
do
     {
          ………………
          ………………
          if(condition)
               {
                    break;//exit from if loop and do-while loop
               }
          ……………..
          ……………..
     }
while(condition);
     ………..
     ………..
```

For loop

```
for(…………)
        {
                ……………
                …………..
                if(…………..)
                break; ;//exit from if loop and for loop
                ……………
                ……………
        }
……………
…………..
```

While loop

```
while(…………)
        {
                ……………
                …………..
                if(…………..)
                break; ;//exit from if loop and while loop
                ……………
                ……………
        }
```

Following statements shows the continuing the loop by using "**continue**" statement.

do-while loop

```
do
        {
                ………………
                ………………
                if(condition)
                {
                        continue;//continue the do-while loop
                }
                ……………..
                ……………..
        }
while(condition);
………..
………..
```

For loop

```
for(…………)
        {
                ……………
                …………..
                if(…………..)
                continue ;// continue the for loop
                ……………
                ……………
        }
……………
…………..
```

While loop

```
while(…………)
        {
                ……………
                …………..
                if(…………..)
                continue ;// continue the while loop
                ……………
                ……………
        }
…………….
…………….
```

Labeled loop

We can give label to a block of statements with any valid name. Following example shows the use of label, break and continue. For example:

```
import java.io.*;
class Demo
        {
                public static void main(String args[]) throws Exception
                {
                        int j,i;
                        LOOP1: for(i=1;i<100;i++)
                        {
                                System.out.println(““);
                                if(i>=10)
                                {
                                        break;
                                }
                                for(j=1;j<100;j++)
                                {
                                        System.out.println(“* ”);
                                        if(i==j)
                                        {
                                                continue LOOP1;
                                        }
                                }
                        }
                System.out.println(“ End of program “);
                }
        }
```

Output:

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
End of program
```

4.5 Summary

In this unit, you have learned about the control structures used in java programming. Selection statement, Iteration statement, Jump Statements are the control structure. These control structures are used widely in programming and in this unit all three control structure are explained in detail.

4.6 Key words

- **if statement** : is also called as conditional branch statement
- **Labeled loop:** loop give label to a block of statements with any valid name

4.7 Unit end exercise and answers

1. Which are the control structures used in java programming?
2. Explain in detail selection statement?
3. Explain in detail Iteration statement?
4. Explain in detail Jump statement?

Answers :see 1. 4.1
2. 4.2
3. 4.3
4. 4.4

4.8 Reference

Teach yourself Java - Joseph O'Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 2002. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

MODULE 2

UNIT 5: Introducing Classes

Structure:

5.0 OBJECTIVES

At the end of this unit you will be able to know:

- Class Fundamentals
- Declaring Objects
- Assigning Object Reference variables
- Introducing Methods
- Constructors
- this keyword
- Garbage collection

- finalize() method
- Stack class.

5.1 INTRODUCTION

Java is a true object oriented language and therefore the underlying structure of all Java programs is classes.Anything we wish to represent in Java must be encapsulated in a class that defines the “state” and “behaviour” of the basic program components known as objects.
Classes create objects and objects use methods to communicate between them. They provide a convenient method for packaging a group of logically related data items and functions that work on them.

5.2 Class fundamentals

Classes have been used since the beginning of java. However, until now, only the most rudimentary form of a class has been used. The classes created in the preceding chapters primarily exist simply to encapsulate the main () method, which has been used to demonstrate the basics of the java’s syntax. Perhaps the most important thing to understand about a class is that it defines a new data type. Once defined, this new type can be used to create objects of that type. Thus, a class is a template for an object, and an object is an instance of a class. Because an object is an instance of a class, you will often see the two words object and instance used interchangeably.

The general form of class:

```
class classname
        {
                //………
        }
```

When you define a class, you declare its exact form and nature. You do this by specifying the data that it contains and the code that operates on that data. While very simple classes may contain only code or only data, most real-world classes contain both. As you will see, a class' code defines the interface to its data.

A class is declared by use of the class keyword. The classes that have been used up to this point are actually very limited examples of its complete form. Classes can (and usually do) get much more complex. The general form of a class definition is shown here:

```
class classname
        {
                type instance-variable1;
                type instance-variable2;
                // ..
                type instance-variableN;
                type methodname1(parameter-list)
                        {
                                //body of method
                        }
                type methodname2(parameter-list)
                        {
                                //body of method
                        }
                //..
                type methodnameN(parameter-list)
                        {
                                // body of method
                        }
        }
```

The data, or variables, defined with a class are called instance variables. The code is contained within methods. Collectively, the methods and variables defined within a class are called members of the class. In most classes, the instance variables are acted upon and accessed by the methods defined for that class. Thus, it is the methods that determine how a class' data can be used.

Variables defined within a class are called instance variables because each instance of the class (that is, each object is separate and unique from the data for another. We will come back to this point shortly, but it is an important concept to learn early.

All methods have the same general form as main (), which we have been using thus far. However, most methods will not be specified as static or public. Notice that the general form of a class does not specify a main () method. Java classes do not need to have a main () method. You only specify one if that class is the starting point for your program. Further, applets don’t require a main () method at all.

Creating Objects

In Java, you create an object by creating an *instance of a class* or, in other words, *instantiating a class*. You will learn how to create a class later in Creating Classes. Until then, the examples contained herein create objects from classes that already exist in the Java environment.

Often, you will see a Java object created with a statement like this one:

```
Date today = new Date();
```

This statement creates a new Date object (Date is a class in the java.util package). This single statement actually performs three actions: declaration, instantiation, and initialization. Date today is a variable declaration which simply declares to the compiler that the name today will be used to refer to an object whose type is Date, the new operator instantiates the Date class (thereby creating a new Date object), and Date initializes the object.

5.3 Declaring an Object

While the declaration of an object is not a necessary part of object creation, object declarations often appear on the same line as the creation of an object. Like other variable declarations, object declarations can appear alone like this:

```
Date today;
```

Either way, declaring a variable to hold an object is just like declaring a variable to hold a value of primitive type:

type name

where *type* is the data type of the object and *name* is the name to be used for the object. In Java, classes and interfaces are just like a data type. So *type* can be the name of a class such as the Date class or the name of an interface. Variables and Data Types discusses variable declarations in detail.

Declarations notify the compiler that you will be using *name* to refer to a variable whose type is *type*. **Declarations do not create new objects.** Date today does not create a new Date object, just a variable named today to hold a Date object. To instantiate the Date class, or any other class, use the new operator.

Instantiating an Object

The new operator instantiates a class by allocating memory for a new object of that type. new requires a single argument: a call to a constructor method. Constructor methods are special methods provided by each Java class that are responsible for initializing new objects of that type. The new operator creates the object, the constructor initializes it.

Here's an example of using the new operator to create a Rectangle object (Rectangle is a class in the java.awt package):

```
new Rectangle(0, 0, 100, 200);
```

In the example, Rectangle(0, 0, 100, 200) is a call to a constructor for the Rectangle class. The new operator returns a reference to the newly created object. This reference can be assigned to a variable of the appropriate type.

```
Rectangle rect = new Rectangle(0, 0, 100, 200);
```

(Recall from Variables and Data Types that a class essentially defines a new reference data type. So, Rectangle can be used as a data type in your Java programs. The value of any variable whose data type is a reference type, such as rect, is a reference (a pointer in other terminology) to the actual value or set of values represented by the variable.

Initializing an Object

Classes provide constructor methods to initialize a new object of that type. A class may provide multiple constructors to perform different kinds of initialization on new objects. When looking at the implementation for a class, you can recognize the constructors because they have

the same name as the class and have no return type. Recall the creation of the `Date` object used at the beginning of this section. The `Date` constructor used there doesn't take any arguments:
Date()

A constructor that takes no arguments, such as the one shown, is known as the *default constructor*. Like `Date`, most classes have at least one constructor, the default constructor. If a class has multiple constructors, they all have the same name but a different number or type of arguments. Each constructor initializes the new object in a different way. Besides the default constructor used to initialize a new `Date` object earlier, the `Date` class provides another constructor that initializes the new `Date` with a year, month, and day:
Date MyBirthday = new Date(1996, 8, 30);
The compiler can differentiate the constructors through the type and number of the arguments.

5.4 Assigning Object Reference Variables

Object reference variables act differently than you might expect when an assignment takes place. It has been explained in this article. You can take an example of following code. Here an object t1 is created for Test class and another object T2 is assigned for the t1 then what do you think the following fragment does?

```
Test t1 = new test ();
Test t2 = t2;
```

You might think that t2 is being assigned a reference to a copy of the object referred to by t5. That is, you might think that t1 and t2 separate and distinct objects. However, this would be wrong. Instead, after this fragment executes, t1 and t2 will both refer to the same object. The assignment of t1 to t2 did not allocate any memory or copy and part of the original object. It simply makes t2 refer to the same object, as does t5. Thus, any changes made to the object through t2 will affect the object to which t1 is referring, since they are the same object.

Although t1 and t2 both refer to the same object, they are not linked in any other way. For example, a subsequent assignment to t1 will simply unhook t1 from the original object without affecting the object or affecting t6.

For example,

```
Test t1 = new test ();

Test t2 = t1;

//…

t1 = null
```

Here, t1 has been set to null, but still points to the original object.

5.5 Introducing Methods

Classes usually consist of two things: - Instance variable and Methods. The topic of methods is a large one because java gives them so much power and flexibility. However, there are some fundamentals that you need to learn now so that you can begin to add methods to your classes.

This is the general form of a method:

```
Type name( parameter-list)
        {
                //body of method
        }
```

Here, type specifies the type of data returned by the method. This can be any valid type, including class types that you create. If the method does not return a value, its return type must be void. The name of the method is specified by name. This can be any legal identifier other than those already used by other items within the current scope. The parameter-list is a sequence of type and identifier pairs separated by commas. Parameters are essentially variables that receive the value of the arguments passed to the method when it is called. If the method has no parameters, then the parameter list will be empty.

Methods that have a return type other than void return a value to the calling routine using the following form of the return statement:

return value;

Here, value is the value returned.

Adding a method to a class

Although it is perfectly fine to create a class that contains only data, it rarely happens. Most of the time you will use methods to access the instance variables defined by the class. In fact, methods define the interface to most classes. This allows the class implementor to hide the specific layout of internal data structures behind cleaner method abstractions. In addition to defining methods that provide access to data, you can also define methods that are used internally by the class itself.

It may have occurred to you while looking at the preceding programs that the computation of a summation was something that was best handled by the test class rather than the testdemo class. After all, since the volume of a box is dependent upon the values, it makes sense to have the test class compute it. To do this, you must add a method to test class, as shown here:

```
class Test
        {
                double item1;
                double item2;
                double item3;
                void value()
                        {
                                System.out.println("value is");
                                System.out.println( item1 + item2 + item3);
                        }
                }
class Testdemo()
        {
                public static void main (String args[])
                        {
                                Test test1 = new test();
                                Test test2 = new test();
                                test5.item1 = 10;
                                test5.item2 = 20;
```

```
                    test5.item3 = 15;
                    test6.item1 = 3;
                    test6.item2 = 6;
                    test6.item3 = 9;
                    test5.value();
                    test6.value();
          }
}
```

This program generates the following output,

```
value is 45.0
value is 18.0
```

Look closely at the following two lines of code:

```
test5.value();
test6.value();
```

The first line here invokes the value () method on test5. That is, it calls value () relative to the test1 object, using the object's name followed by the dot operator. Thus, the call to test5.value() displays the value of the members defined by test1 and the call to test6.value() displays the value of the members defined by test6. Each time value () is invoked, it displays the volume for the specified data member.

5.6 Constructors

A **constructor** has the same name as the name of the class to which it belongs. Constructor's syntax does not include a return type, since constructors never return a value. Constructors may include parameters of various types. When the constructor is invoked using the new operator, the types must match those that are specified in the constructor definition. Java provides a default constructor which takes no arguments and performs no special actions or initializations, when no explicit constructors are provided. The only action taken by the implicit default constructor is to call the superclass constructor using the super() call. Constructor arguments provide you with a way to provide parameters for the initialization of an object. Below is an example of a cube class containing 2 constructors.

```
public class Cube1 {
   int length ;
   int breadth ;
   int height ;
   public int getVolume( ) {
     return ( length * breadth * height );
   }
   Cube1(){
               length = 10;
               breadth = 10;
               height = 10;
   }
   Cube1(int l, int b, int h){
               length = l;
               breadth = b;
               height = h;
   }
   public static void main(String[] args) {
                Cube1 cubeObj1, cubeObj2;
                        cubeObj1 = new Cube1();
                        cubeObj2 = new Cube1(10, 20, 30);
                        System.out.println("Volume of Cube1 is : "+cubeObj5.getVolume());
                        System.out.println("Volume of Cube2 is : "+cubeObj6.getVolume());
            }
}
```

Java Overloaded Constructors

Like methods, constructors can also be overloaded. Since the constructors in a class all have the same name as the class, />their signatures are differentiated by their parameter lists. The above example shows that the Cube1 constructor is overloaded one being the default constructor and the other being a parameterized constructor.

It is possible to use this() construct, to implement local chaining of constructors in a class. The this() call in a constructor invokes the another constructor with the corresponding parameter list within the same class. Calling the default constructor to create a Cube object results in the second and third parameterized constructors being called as well. Java requires that any this() call must occur as the first statement in a constructor.

Below is an example of a cube class containing 3 constructors which demonstrates the this() method in Constructors context.

```
public class Cube2 {
  int length ;
  int breadth ;
  int height ;
  public int getVolume( ) {
    return ( length * breadth * height );
  }
  Cube2(){
            this(10, 10);
      System.out.println("Finished with Default Constuctor");
  }
  Cube2(int l, int b){
            this(l, b, 10);
      System.out.println("Finished with Parametrized Constuctor having 2 params");
  }
  Cube2(int l, int b, int h){
            length = l;
            breadth = b;
            height = h;
            System.out.println("Finished with Parametrized Constuctor having 3 params");
  }
  public static void main(String[] args) {
             Cube2 cubeObj1, cubeObj2;
                    cubeObj1 = new Cube2();
		cubeObj2 = new Cube2(10, 20, 30);
		System.out.println("Volume of Cube1 is : "+cubeObj5.getVolume());
		System.out.println("Volume of Cube2 is : "+cubeObj6.getVolume());
	}
}
```

Output:

```
Finished with Parameterized Constructor having 3 params
Finished with Parameterized Constructor having 2 params
Finished with Default Constructor
Finished with Parameterized Constructor having 3 params
Volume of Cube1 is : 1000
Volume of Cube2 is : 6000
```

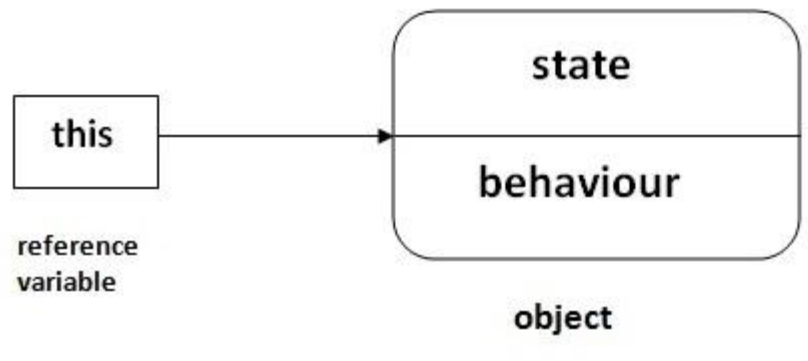
5.7 this Keyword

There can be a lot of usage of **this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of this keyword

Here is given the 6 usage of this keyword.

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.



The this keyword can be used to refer current class instance variable

If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
class student
{
int id;
String name;
student(int id,String name)
{
id = id;
name = name;
}
void display(){System.out.println(id+" "+name);
}
public static void main(String args[])
{
student s1 = new student(111,"Karan");
student s2 = new student(321,"Aryan");
s5.display();
s6.display();
}
}
```

Output:
```
0 null
0 null
```

In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable.

Solution of the above problem by this keyword

```
//example of this keyword
class Student
{
int id;
String name;
student(int id,String name)
{
this.id = id;
this.name = name;
}
void display(){System.out.println(id+" "+name);
}
public static void main(String args[])
{
Student s1 = new Student(111,"Karan");
Student s2 = new Student(222,"Aryan");
s5.display();
s6.display();
}
}
```

Output:
```
111 Karan
222 Aryan
```

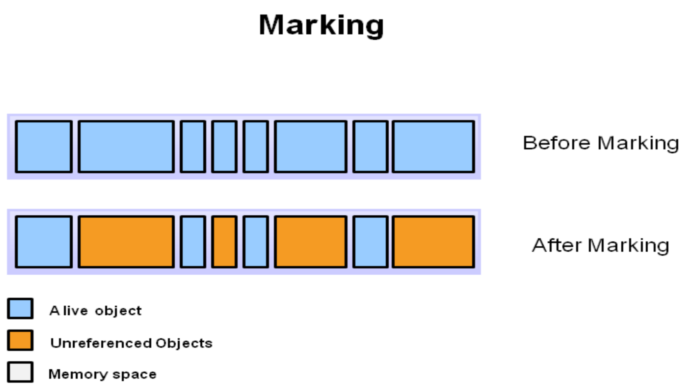
5.8 Garbage Collection

Garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused object, or unreferenced object, is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed.

In a programming language like C, allocating and deallocating memory is a manual process. In Java, process of deallocating memory is handled automatically by the garbage collector. The basic process can be described as follows.

Step 1: Marking

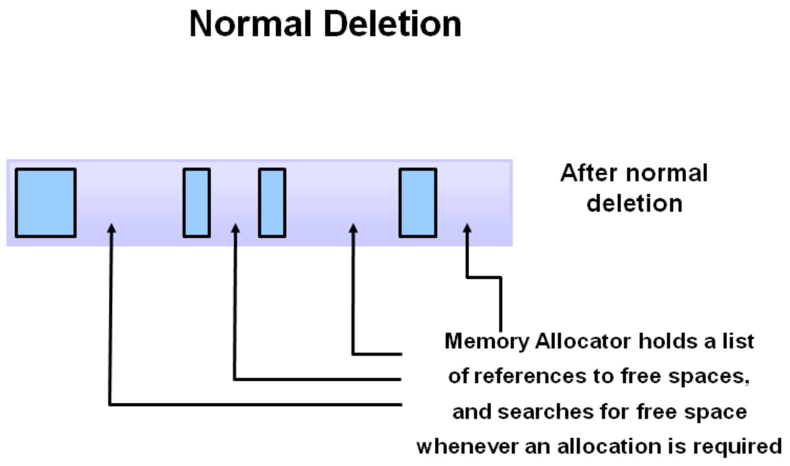
The first step in the process is called marking. This is where the garbage collector identifies which pieces of memory are in use and which are not.



Referenced objects are shown in blue. Unreferenced objects are shown in gold. All objects are scanned in the marking phase to make this determination. This can be a very time consuming process if all objects in a system must be scanned.

Step 2: Normal Deletion

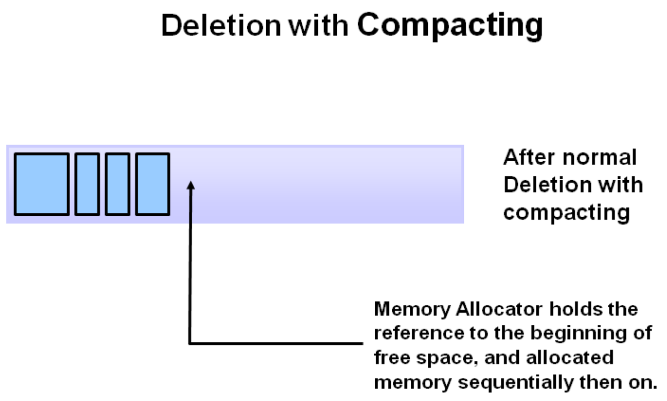
Normal deletion removes unreferenced objects leaving referenced objects and pointers to free space.


Normal Deletion
After normal deletion
Memory Allocator holds a list of references to free spaces, and searches for free space whenever an allocation is required

The memory allocator holds references to blocks of free space where new object can be allocated.

Step 2a: Deletion with Compacting

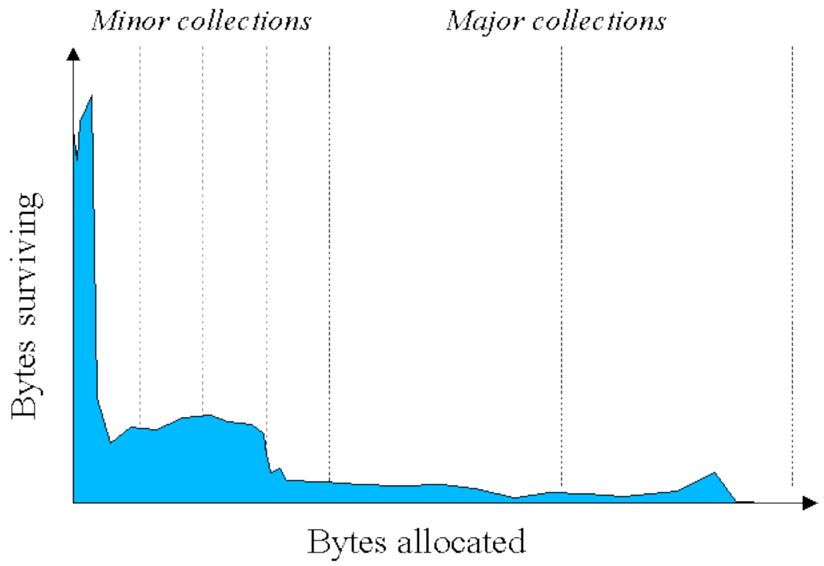
To further improve performance, in addition to deleting unreferenced objects, you can also compact the remaining referenced objects. By moving referenced object together, this makes new memory allocation much easier and faster.


Deletion with Compacting
After normal Deletion with compacting
Memory Allocator holds the reference to the beginning of free space, and allocated memory sequentially then on.

Why Generational Garbage Collection?

As stated earlier, having to mark and compact all the objects in a JVM is inefficient. As more and more objects are allocated, the list of objects grows and grows leading to longer and longer garbage collection time. However, empirical analysis of applications has shown that most objects are short lived.

Here is an example of such data. The Y axis shows the number of bytes allocated and the X access shows the number of bytes allocated over time.



As you can see, fewer and fewer objects remain allocated over time. In fact most objects have a very short life as shown by the higher values on the left side of the graph.

JVM Generations

The information learned from the object allocation behavior can be used to enhance the performance of the JVM. Therefore, the heap is broken up into smaller parts or generations. The heap parts are: Young Generation, Old or Tenured Generation, and Permanent Generation

The **Young Generation** is where all new objects are allocated and aged. When the young generation fills up, this causes a *minor garbage collection*. Minor collections can be optimized assuming a high object mortality rate. A young generation full of dead objects is collected very quickly. Some surviving objects are aged and eventually move to the old generation.

Stop the World Event - All minor garbage collections are "Stop the World" events. This means that all application threads are stopped until the operation completes. Minor garbage collections are *always* Stop the World events.

The **Old Generation** is used to store long surviving objects. Typically, a threshold is set for young generation object and when that age is met, the object gets moved to the old generation. Eventually the old generation needs to be collected. This event is called a *major garbage collection*.

Major garbage collection are also Stop the World events. Often a major collection is much slower because it involves all live objects. So for Responsive applications, major garbage collections should be minimized. Also note that the length of the Stop the World event for a major garbage collection is affected by the kind of garbage collector that is used for the old generation space.

The **Permanent generation** contains metadata required by the JVM to describe the classes and methods used in the application. The permanent generation is populated by the JVM at runtime based on classes in use by the application. In addition, Java SE library classes and methods may be stored here.

Classes may get collected (unloaded) if the JVM finds they are no longer needed and space may be needed for other classes. The permanent generation is included in a full garbage collection.

5.9 Writing a finalize () Method

Before an object is garbage collected, the runtime system calls its `finalize()` method. The intent is for `finalize()` to release system resources such as open files or open sockets before getting collected. Your class can provide for its finalization simply by defining and implementing a method in your class named `finalize()`. Your `finalize()` method must be declared as follows:

protected void finalize () throws throwable

This class opens a file when its constructed:

class OpenAFile

```
{
   FileInputStream aFile = null;
   OpenAFile(String filename) {
      try {
         aFile = new FileInputStream(filename);
      } catch (java.io.FileNotFoundException e) {
         System.err.println("Could not open file " + filename);
      }
   }
}
```

To be well-behaved, the OpenAFile class should close the file when its finalized. Here's the `finalize()` method for the OpenAFile class:

```
protected void finalize () throws throwable
{
   if (aFile != null) {
      aFile.close();
      aFile = null;
   }
}
```

The `finalize()` method is declared in the java.lang.Object class. Thus when you write a `finalize()` method for your class you are overriding the one in your superclass. Overriding Methods talks more about how to override methods.

If your class's superclass has a `finalize()` method, then your class's `finalize()` method should probably call the superclass's `finalize()` method after it has performed any of its clean up duties. This cleans up any resources the object may have unknowingly obtained through methods inherited from the superclass.

```
protected void finalize() throws Throwable
{
   . . .
   // clean up code for this class here
```

```
    . . .
    super.finalize();
}
```

5.10 Stack class

Stack is a subclass of Vector that implements a standard last-in, first-out stack. Stack only defines the default constructor, which creates an empty stack. Stack includes all the methods defined by Vector, and adds several of its own.

```
Stack( )
```

Apart from the methods inherited from its parent class Vector, Stack defines following methods:

SN	Methods with Description
1	**boolean empty()** Tests if this stack is empty. Returns true if the stack is empty, and returns false if the stack contains elements.
2	**Object peek( )** Returns the element on the top of the stack, but does not remove it.
3	**Object pop( )** Returns the element on the top of the stack, removing it in the process.
4	**Object push(Object element)** Pushes element onto the stack. element is also returned.
5	**int search(Object element)** Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, .1 is returned.

Example:

The following program illustrates several of the methods supported by this collection:

```
import java.util.*;

public class StackDemo {

   static void showpush(Stack st, int a) {
      st.push(new Integer(a));
      System.out.println("push(" + a + ")");
      System.out.println("stack: " + st);
   }

   static void showpop(Stack st) {
      System.out.print("pop -> ");
      Integer a = (Integer) st.pop();
      System.out.println(a);
      System.out.println("stack: " + st);
   }
   public static void main(String args[]) {
      Stack st = new Stack();
      System.out.println("stack: " + st);
      showpush(st, 42);
      showpush(st, 66);
      showpush(st, 99);
      showpop(st);
      showpop(st);
      showpop(st);
      try {
         showpop(st);
      } catch (EmptyStackException e) {
         System.out.println("empty stack");
      }
   }
}
```

This would produce the following result:

```
stack: [ ]
push(42)
stack: [42]
push(66)
stack: [42, 66]
push(99)
stack: [42, 66, 99]
pop -> 99
stack: [42, 66]
pop -> 66
stack: [42]
pop -> 42
stack: [ ]
pop -> empty stack
```

5.11 Summary

In this unit, you have learned the importance of classes and objects. What are the fundamental of classes, How to declare objects, assigning object reference variables.how methods are worked under classes.what is constructor and how this keyword helps in programming are the topic discussed in this unit. Also what is the role of Garbage collector in java and finalze method and stack classes are discussed in detail.

5.12 Key words

- **Old Generation** is used to store long surviving objects.
- **Young Generation** is where all new objects are allocated and aged.

5.13 Unit end exercise and answers

1. What is Class Fundamentals?
2. How to Declare Objects and Assigning Object Reference variables?
3. Give Introduction to Methods?
4. What is Constructors? Explain.
5. What is the used of this keyword? Explain
6. What is Garbage collection, explain and mention its use?
7. Explain finalize () method?
8. What is Stack class? Explain

Answers :see 1. 5.2
2. 5.3, 5.4
3. 5.5
4. 5.6
5. 5.7
6. 5.8
7. 5.9
8. 5.10

5.14 Reference

Teach yourself Java - Joseph O'Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 2006. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 6: Closer Look at Methods and Classes

Structure:

6.0 OBJECTIVES

At the end of this unit you will be able to know:

- Overloading Methods
- Using Objects as
- parameters Argument passing
- Returning objects
- Recursion
- Access control.

- Static
- Final
- Nested and Inner classes
- Using Command line arguments

6.1 INTRODUCTION

Classes, objects, and methods are the basic components used in Java programming. A class is a collection of fields (data) and methods (procedure or function) that operate on that data. A class with only data fields has no life. Objects created by such a class cannot respond to any messages.Methods are declared inside the body of the class but immediately after the declaration of data fields.

6.2 Overloading Methods

Method overloading in Java occurs when two or more methods shares same name and fulfill at least one of the following condition.

1) Have different number of arguments.
2) Have same number of arguments but their types are different.
3) Have both different numbers of arguments with a difference in their types.

Method overloading in same class

Overloading occurs in a single class when two or more methods share a same name with a 'difference in number of arguments' or 'difference in argument type' or both.

```
package com.beingjavaguys.core;
import java.util.Date;
        public class MethodOverloadingDemo
        {
                public void getEmpName(int empId)
                {
                        ......
```

```
		}
		public void getEmpName(String empName)
		{
			......
		}
		public void getEmpName(int empId, String empName)
		{
			......
		}
		public void getEmpName(Date dob, String empName)
		{
			......
		}
	}
```

Method overloading in subclass

It is also possible to overload methods in sub classes, as long as the methods shares same name and they are different in the number of parameters or type of parameters.

```
	public class Fruit
	{
		public void getFruit(int fruitId)
		{
			......
		}
	}
	public class Apple extends Fruit
	{
		public void getFruit(String name)
		{
			......
```

```
        }
        public void getFruit(int fruitId, String name)
        {
            ......
        }
    }
```

How to not overload method

Methods having same name in one class or in child class could not perform method overloading if they are just different in:

1) Name of argument, and not in number of arguments or type of arguments.
2) Having different return type only.

```
        public void getEmpName(int empId) {
        ......
        }
        public void getEmpName(int employeeId) {
        ......
        }
        public void getFruit(String name){
        ....
        }
        public String getFruit(String name){
        ....
        return "Apple";
        }
```

How method overloading helps in Java

When it comes to, why to use method overloading and usage of method overloading. Two points comes out.

1) Helps in maintain consistency in method naming, doing same task with just a difference of input parameter.

2) Helps in reduce overhead of remember name of different functions for same task, one can pass possible different type of parameters or number of parameters to a single overloaded function to get the result.

```
public Employee getEmpById(int empId) {
......
}
public Employee getEmpByName(String empName) {
......
}
public Employee getEmpByDob(Date dob) {
......
}
public Employee getEmp(int empId) {
......
}
public Employee getEmp(String empName) {
......
}
public Employee getEmp(Date dob) {
......
}
```

6.3 Using Objects as Parameters

So far we have only been using simple types as parameters to methods. However, it is both correct and common to pass objects to methods. For example, consider the following short program:

```
// Objects may be passed to methods.
class Test {
int a, b;
Test(int i, int j) {
a = i;
b = j;
}
// return true if o is equal to the invoking object
boolean equals(Test o) {
if(o.a == a && o.b == b) return true;
else return false;
}
}
class PassOb {
public static void main(String args[]) {
Test ob1 = new Test(100, 22);
Test ob2 = new Test(100, 22);
Test ob3 = new Test(-1, -1);
System.out.println("ob1 == ob2: " + ob5.equals(ob2));
System.out.println("ob1 == ob3: " + ob5.equals(ob3));
}
}
```

This program generates the following output:

```
ob1 == ob2: true
ob1 == ob3: false
```

As you can see, the equals( ) method inside Test compares two objects for equality and returns the result. That is, it compares the invoking object with the one that it is passed. If they contain the same values, then the method returns true. Otherwise, it returns false. Notice that the parameter o in equals( ) specifies Test as its type. Although Test is a class type created by the program, it is used in just the same way as Java's built-in types.

One of the most common uses of object parameters involves constructors. Frequently you will want to construct a new object so that it is initially the same as some existing object. To do this, you must define a constructor that takes an object of its class as a parameter. For example, the following version of Box allows one object to initialize another:

```
// Here, Box allows one object to initialize another.
class Box {
double width;
double height;
double depth;
// construct clone of an object
Box(Box ob) { // pass object to constructor
width = ob.width;
height = ob.height;
depth = ob.depth;
}
// constructor used when all dimensions specified
Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
// constructor used when no dimensions specified
Box() {
width = -1; // use -1 to indicate
height = -1; // an uninitialized
depth = -1; // box
}
// constructor used when cube is created
Box(double len) {
width = height = depth = len;
}
// compute and return volume
double volume() {
return width * height * depth;
}
}
class OverloadCons2 {
public static void main(String args[]) {
// create boxes using the various constructors
Box mybox1 = new Box(10, 20, 15);
Box mybox2 = new Box();
Box mycube = new Box(7);
Box myclone = new Box(mybox1);
double vol;
// get volume of first box
vol = mybox5.volume();
System.out.println("Volume of mybox1 is " + vol);
// get volume of second box
vol = mybox6.volume();
System.out.println("Volume of mybox2 is " + vol);
// get volume of cube
vol = mycube.volume();
System.out.println("Volume of cube is " + vol);
// get volume of clone
vol = myclone.volume();
System.out.println("Volume of clone is " + vol);
}
}
```

As you will see when you begin to create your own classes, providing many forms of constructor methods is usually required to allow objects to be constructed in a convenient and efficient manner.

6.4 Argument passing

```
package com.pritesh.programs;
class Rectangle {
    int length;
    int width;
    Rectangle(int l, int b) {
        length = l;
        width = b;
    }
    void area(Rectangle r1) {
        int areaOfRectangle = r5.length * r5.width;
        System.out.println("Area of Rectangle : "+areaOfRectangle);
    }
}
class RectangleDemo {
    public static void main(String args[]) {
        Rectangle r1 = new Rectangle(10, 20);
        r5.area(r1);
    }
}
```

Output of the program:

```
Area of Rectangle: 200
```

Explanation:

1. We can pass Object of any class as parameter to a method in java.
2. We can access the instance variables of the object passed inside the called method.

```
area = r5.length * r5.width
```

3. It is good practice to initialize instance variables of an object before passing object as parameter to method otherwise it will take default initial values.

Different Ways of Passing Object as Parameter:

Way 1 : By directly passing Object Name

```
void area(Rectangle r1) {
        int areaOfRectangle = r5.length * r5.width;
        System.out.println("Area of Rectangle : "+areaOfRectangle);
    }
class RectangleDemo {
    public static void main(String args[]) {
        Rectangle r1 = new Rectangle(10, 20);
        r5.area(r1);
    }
```

Way 2 : By passing Instance Variables one by one

```
package com.pritesh.programs;

class Rectangle {
    int length;
    int width;
    void area(int length, int width) {
        int areaOfRectangle = length * width;
        System.out.println("Area of Rectangle : "+areaOfRectangle);
    }
}
class RectangleDemo {
    public static void main(String args[]) {
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle();
        r5.length = 20;
        r5.width = 10;

        r6.area(r5.length, r5.width);
    }
}
```

Actually this is not a way to pass the object to method. but this program will explain you how to pass instance variables of particular object to calling method.

Way 3 : We can pass only public data of object to the Method.

Suppose we made width variable of a class private then we cannot update value in a main method since it does not have permission to access it.

```
private int width;
```

after making width private -

```
class RectangleDemo {
    public static void main(String args[]) {
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle();

        r5.length = 20;
        r5.width = 10;

        r6.area(r5.length, r5.width);
    }
}
```

6.5 Returning Objects

In Java Programming a method can return any type of data, including class types that you create. For example, in the following program, the getRectangleObject() method returns an object.

Java Program: Returning the Object

```
package com.pritesh.programs;
import java.io.File;
import java.io.IOException;
class Rectangle {
    int length;
    int breadth;
    Rectangle(int l,int b) {
      length = l;
      breadth = b;
    }
    Rectangle getRectangleObject() {
      Rectangle rect = new Rectangle(10,20);
      return rect;
    }
}
class RetOb {
    public static void main(String args[]) {
      Rectangle ob1 = new Rectangle(40,50);
      Rectangle ob2;
      ob2 = ob5.getRectangleObject();
      System.out.println("ob5.length : " + ob5.length);
      System.out.println("ob5.breadth: " + ob5.breadth);
      System.out.println("ob6.length : " + ob6.length);
      System.out.println("ob6.breadth: " + ob6.breadth);
      }
}
```

Output of the Program:

```
ob5.length:       40

ob5.breadth:      50

ob6.length:       10

ob6.breadth:      20
```

Explanation:

1) In the above program we have called a method getRectangleObject() and the method creates object of class from which it has been called.
2) All objects are dynamically allocated using new, you don't need to worry about an object going out-of-scope because the method in which it was created terminates.
3) The object will continue to exist as long as there is a reference to it somewhere in your program. When there are no references to it, the object will be reclaimed the next time garbage collection takes place.

6.6 Recursive

In computer programming it's the process of having a method continually call itself until a defined point of termination.

Let's consider the following recursive code:

```
public int myRecursiveMethod ()
{
        int aVariable = myRecursiveMethod();
}
```

Well the first thing you should take note of is the name of the method: `myRecursiveMethod`. This is just a random name that I chose to use for this method…nothing special going on there… But, take a look at what we're doing inside the method: we're calling a method named `myRecursiveMethod`. Notice anything special there? **Yes**, that's the same method name!

Recursion

The method will call itself, and it will execute the code inside, which is to call itself, so it will execute the code inside of that method, which is to call itself, so it will execute that code, which is to call itself… You see what I'm getting at here? This code is missing a terminating condition; this is why it will run forever. So how about we include a terminating condition?

```
public int myRecursiveMethod (int aVariable)
{
	System.out.println(aVariable);
	aVariable--;
	if (aVariable == 0)
	return 0;
	return myRecursiveMethod(aVariable);
}
```

Now with this method, we've introduced a terminating condition. Can you spot it?

When our `int` variable holds the value `0`, then this method will not call itself and instead it will simply exit out of the flow. This can be seen from the `return 0` statement.

So now we're in a position where this method will continually call itself with a decrementing value of `aVariable`. So once `aVariable` hits zero, our recursive method is done!

Can you guess what the output would be if we called this method like so:

```
myRecursiveMethod (10)
```

Think about it, try and follow through the code line by line and see what conclusions you can come to… once you've made a guess, go ahead and create a Class file with a `main` method and throw `myRecursiveMethod` in the mix and call it (you'll need to make the method `static`).

Once you've run the program, if you have NO clue what's going on behind the scenes, then I'd suggest debugging by throwing a breakpoint in where the method begins. Step through the code line by line and see what happens (it will clear things up).

Why recursion is used

There are certain problems that just make sense to solve via Java recursion. This is the case because sometimes, when solving problems recursively, you can really cut down on code with your solutions. For example let's take a look at something called the Fibonacci sequence. Here are the first few numbers of this sequence:
0, 1, 1, 2, 3, 5, 8, 13, 21…
If you've never seen the Fibonacci sequence before, then can you figure out what's going on here? Well, here it is explained about, so if you want to try and figure it out on your own, then stop reading.
Here's how it works:
The 3rd number is the sum of numbers 1 and 2 (0+1=1).
The 4th number is the sum of numbers 2 and 3 (1+1=2).
The 5th number is the sum of numbers 3 and 4 (1+2=3).
The 6th number is the sum of numbers 4 and 5 (2+3=5).
etc.

What's neat about this sequence is that when you try to sit down and think up an algorithm to solve this problem, you can't help but think of recursion. The reason for this is because the solution to the next number relies on the past two iterations. Now that's not to say that the only way to solve this problem is with Java recursion, but it can make the coding much simpler.

6.7 Access control

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:

- Visible to the package. the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

Default Access Modifier - No keyword

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc. A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

Example:

Variables and methods can be declared without any modifiers, as in the following examples:

```
String version = "5.5.1";
boolean processOrder()
{
  return true;
}
```

Private Access Modifier - private

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself. Private access modifier is the most restrictive access level. Class and interfaces cannot be private. Variables that are declared private can be accessed outside the class if public getter methods are present in the class. Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

Example:

The following class uses private access control:

```
public class Logger {
  private String format;
  public String getFormat() {
    return this.format;
  }
  public void setFormat(String format) {
     this.format = format;
  }
}
```

Here, the format variable of the Logger class is private, so there's no way for other classes to retrieve or set its value directly.

So to make this variable available to the outside world, we defined two public methods: getFormat(), which returns the value of format, and setFormat(String), which sets its value.

Public Access Modifier - public

A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe. However if the public class we are trying to access is in a different package, and then the public class still need to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Example:

The following function uses public access control:

```
public static void main(String[] arguments) {
  // ...
}
```

The main() method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as java) to run the class.

Protected Access Modifier - protected

Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class. The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected. Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

Example:

The following parent class uses protected access control, to allow its child class override*openSpeaker()* method:

```
class AudioPlayer {
  protected boolean openSpeaker(Speaker sp) {
    // implementation details
  }
}
class StreamingAudioPlayer {
  boolean openSpeaker(Speaker sp) {
    // implementation details
  }
}
```

Here, if we define *openSpeaker()* method as private, then it would not be accessible from any other class other than *AudioPlayer*. If we define it as public, then it would become accessible to the entire outside world. But our intension is to expose this method to its subclass only, that why we used *protected* modifier.

Access Control and Inheritance

The following rules for inherited methods are enforced:

- Methods declared public in a superclass also must be public in all subclasses.
- Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.
- Methods declared without access control (no modifier was used) can be declared more private in subclasses.

Methods declared private are not inherited at all, so there is no rule for them.

6.8 Static

In Java, a static member is a member of a class that isn't associated with an instance of a class. Instead, the member belongs to the class itself. As a result, you can access the static member without first creating a class instance.

The two types of static members are static fields and static methods:

Static field: A field that's declared with the static keyword, like this:

private static int ballCount;

The position of the static keyword is interchangeable with the positions of the visibility keywords (private and public, as well as protected). As a result, the following statement works, too:

static private int ballCount;

As a convention, most programmers tend to put the visibility keyword first.

The value of a static field is the same across all instances of the class. In other words, if a class has a static field named CompanyName, all objects created from the class will have the same value for CompanyName.

Static fields are created and initialized when the class is first loaded. That happens when a static member of the class is referred to or when an instance of the class is created, whichever comes first.

Static method: A method declared with the static keyword. Like static fields, static methods are associated with the class itself, not with any particular object created from the class. As a result, you don't have to create an object from a class before you can use static methods defined by the class.

The best-known static method is main, which is called by the Java runtime to start an application. The main method must be static, which means that applications run in a static context by default.

One of the basic rules of working with static methods is that you can’t access a nonstatic method or field from a static method because the static method doesn’t have an instance of the class to use to reference instance methods or fields.

6.9 Final

Final in java is very important keyword and can be applied to class, method, and variables in Java. In this java final tutorial we will see what is final keyword in Java, what does it mean by making final variable, final method and final class in java and what are primary benefits of using final keywords in Java and finally some examples of final in Java. Final is often used along with static keyword in Java to make static final constant and you will see how final in Java can increase performance of Java application.

6.10 Nested and Inner classes

Inner class and nested static class in Java both are classes declared inside another class, known as top level class in Java. In Java terminology, If you declare a nested class static, it will called nested static class in Java while non static nested class are simply referred as Inner Class. Inner classes are also a popular topic in Java interviews. One of the popular question is difference between inner class and nested static class ,some time also refereed as difference between static and non static nested class in Java. One of the most important question related to nested classes are Where should you use nested class in Java? This is one of the tricky Java question, If you have read Effective Java from Joshua Bloch than in one chapter he has suggested that if a class can only be useful to one particular class, it make sense to keep that inside the class itself otherwise declare it as top level class. Nested class improves Encapsulation and help in maintenance. I actually look JDK itself for examples and if you look HashMap class, you will find that Map. Entry is a good example of nested class in Java. Another popular use of nested classes are implementing Comparator in Java e.g. AgeCompartor for Person class.

Since some time Comparator is also tied with a particular class, it make sense to declare them as nested static class in Java. In this Java tutorial we will see what is Inner class in Java, different types of Inner classes, *what is static nested class* and finally difference between static nested class and inner class in Java.

Inner Class in Java

Any class which is not a top level or declared inside another class is known as nested class and out of those nested classes, class which are declared non static are known as Inner class in Java. There are three kinds of Inner class in Java:

- Local inner class
- Anonymous inner class
- Member inner class

Local inner class is declared inside a code block or method. Anonymous inner class is a class which doesn't have name to reference and initialized at same place where it gets created. Member inner class is declared as non static member of outer class. Now with Inner class first question comes in mind is when to use Inner class in Java, simple answer is any class which is only be used by its outer class, should be a good candidate of making inner. One of the common example of Inner classes are Anonymous class which is used to implement Thread or EventListeners like ActionListerner in Swing, where they only implement key methods like run() method of Thread class or actionPerformed(ActionEvent ae).

Here are some important properties of Inner classes in Java:

1) In order to create instance of Inner class, an instance of outer class is required. Every instance of inner class is bounded to one instance of Outer class.
2) Member inner class can be making private, protected or public. its just like any other member of class.
3) Local inner class can not be private, protected or public because they exist only inside of local block or method. You can only use final modifier with local inner class.

4) Anonymous Inner classes are common to implement Runnable or CommandListener where you just need to implement one method. They are created and initialized at same line.
5) You can access current instance of Outer class, inside inner class as Outer. This variable.

Inner class Example in Java

Here is an example of member inner class, local inner class and anonymous inner class. For simplicity we have combined all examples of different inner class in one.

```
public class InnerClassTest {
    public static void main(String args[]) {
       //creating local inner class inside method
        class Local {
            public void name() {
                System.out.println("Example of Local class in Java");
           }
        }
       //creating instance of local inner class
        Local local = new Local();
        local.name(); //calling method from local inner class
        //Creating anonymous inner class in java for implementing thread
        Thread anonymous = new Thread(){
            @Override
            public void run(){
                System.out.println("Anonymous class example in java");
            }
        };
        anonymous.start();
        //example of creating instance of inner class
        InnerClassTest test = new InnerClassTest();
        InnerClassTest.Inner inner = test.new Inner();
        inner.name(); //calling method of inner class
    }
    /*
     * Creating Inner class in Java
     */
    private class Inner{
        public void name(){
            System.out.println("Inner class example in java");
        }
    }
}
Output:
Example of Local class in Java
Inner class example in java
Anonymous class example in java
```

Nested static class in Java

Nested static class is another class which is declared inside a class as member and made static. Nested static class is also declared as member of outer class and can be make private, public or protected like any other member. One of the main benefit of nested static class over inner class is that instance of nested static class is not attached to any enclosing instance of Outer class. You also don't need any instance of Outer class to create instance of nested static class in Java. This makes nested static class very convenient to use and access.

Nested Static Class Example in Java

Here is an example of nested static class in Java. It look exactly similar to member inner classes but has quite a few significant difference with them, e.g. you can access them inside main method because they are static. In order to create instance of nested static class, you don’t need instance of enclosing class. You can refer them with class name and you can also import them using static import feature of Java 5.

```
public class NestedStaticExample {
    public static void main(String args[]){
        StaticNested nested = new StaticNested();
        nested.name();
    }
    //static nested class in java
    private static class StaticNested{
        public void name(){
            System.out.println("static nested class example in java");
        }
    }
}
```

Though this is very trivial example of nested static class, it demonstrate some properties of nested static class. Better example of nested static class can be implementing a custom Comparator e.g. OrderByAmount in How to sort Object in Java using Comparator.

Difference between Inner class and nested static class in Java

Both static and non static nested class or Inner needs to declare inside enclosing class in Java and that's why they are collectively known as nested classes but they have couple of differences as shown below:

1) First and most important difference between Inner class and nested static class is that Inner class require instance of outer class for initialization and they are always associated with instance of enclosing class. On the other hand nested static class is not associated with any instance of enclosing class.
2) Another difference between Inner class and nested static class is that later uses static keyword in there class declaration, which means they are static member of class and can be accessed like any other static member of class.
3) Nested static class can be imported using static import in Java.
4) One last difference between Inner class and nested static class is that later is more convenient and should be preferred over Inner class while declaring member classes.

6.11 Command-Line Arguments

A Java application can accept any number of arguments from the command line. This allows the user to specify configuration information when the application is launched.

The user enters command-line arguments when invoking the application and specifies them after the name of the class to be run. For example, suppose a Java application called `Sort` sorts lines in a file. To sort the data in a file named `friends.txt`, a user would enter:

```
java Sort friends.txt
```

When an application is launched, the runtime system passes the command-line arguments to the application's main method via an array of `String`s. In the previous example, the command-line arguments passed to the `Sort` application in an array that contains a single `String`: `"friends.txt"`.

Echoing Command-Line Arguments

The `Echo` example displays each of its command-line arguments on a line by itself:

```
public class Echo {
   public static void main (String[] args) {
      for (String s: args) {
         System.out.println(s);
      }
   }
}
```

The following example shows how a user might run `Echo`. User input is in italics.

java Echo Drink Hot Java

```
Drink
Hot
Java
```

java Echo "Drink Hot Java"

Drink Hot Java

Parsing Numeric Command-Line Arguments

If an application needs to support a numeric command-line argument, it must convert a `String` argument that represents a number, such as "34", to a numeric value. Here is a code snippet that converts a command-line argument to an `int`:

```
int firstArg;
if (args.length > 0) {
   try {
      firstArg = Integer.parseInt(args[0]);
   } catch (NumberFormatException e) {
      System.err.println("Argument" + " must be an integer");
      System.exit(1);
   }
}
```

`parseInt` throws a `NumberFormatException` if the format of `args[0]` isn't valid. All of the `Number` classes — `Integer`, `Float`, `Double`, and so on.

6.12 Summary

In this unit, you have learned the importance ofmethods and class. How to use object as a parameter and argument passing ad how object are returned to called method are discussed in this unit. Topic such as recursion which are important in programming and access control for accessing mehods and object are also discussed.keyword such as static and final and nested and inner classes and uses of command line argument are also discussed in detail.

6.13 Key words

1. **Static field**: A field that's declared with the static keyword

6.14 Unit end exercise and answers

1. Explain overloading methods in java?
2. How to use object as parameter.explain?
3. How argument is passed in a java program?
4. Explain how object are returned?
5. Explain the concept of recursion with example and whats its use?
6. Explain in detail about access control in java?
7. Explain about static and final keywords?
8. Explain about Nested classes and inner classes in detail?
9. What is command line argument? How it can be used?

Answers :see 1. 6.2
2. 6.3
3. 6.4
4. 6.5
5. 6.6
6. 6.7
7. 6.8, 6.9
8. 6.10
9. 6.11

6.15 Reference

Teach yourself Java - Joseph O'Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 2006. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 7: Inheritance

Structure:

7.0 OBJECTIVES

At the end of this unit you will be able to know:

- Inheritance basics
- Using super
- Multilevel Hierarchy
- When constructors are called
- Method overriding

- Dynamic method dispatch
- Abstract classes
- Final with inheritance
- Object class.
- Packages and Interfaces: Packages
- Importing Packages
- Interfaces.

7.1 INTRODUCTION

A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*). The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).

Excepting Object, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object.

Classes can be derived from classes that are derived from classes that are derived from classes, and so on, and ultimately derived from the topmost class, Object. Such a class is said to be *descended* from all the classes in the inheritance chain stretching back to Object.

The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. In doing this, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself.

A subclass inherits all the *members* (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

7.2 Inheritance in Java

Inheritance is a compile-time mechanism in Java that allows you to extend a class (called the base class or superclass) with another class (called the derived class or subclass). In Java, inheritance is used for two purposes:

1) Class inheritance - create a new class as an extension of another class, primarily for the purpose of code reuse. That is, the derived class inherits the public methods and public data of the base class. Java only allows a class to have one immediate base class, i.e., single class inheritance.
2) Interface inheritance - create a new class to implement the methods defined as part of an interface for the purpose of sub typing. That is a class that implements an interface "conforms to" (or is constrained by the type of) the interface. Java supports multiple interface inheritance.

In Java, these two kinds of inheritance are made distinct by using different language syntax. For class inheritance, Java uses the keyword extends and for interface inheritance Java uses the keyword implements.

```
public class derived-class-name extends base-class-name
{
        // derived class methods extend and possibly override
        // those of the base class
}
public class class-name implements interface-name
{
        // class provides an implementation for the methods
        // as specified by the interface
}
```

Example of class inheritance

```
package MyPackage;
class Base
{
        private int x;
```

```
    public int f() { ... }
    protected int g() { ... }
}
class Derived extends Base
{
    private int y;
    public int f() { /* new implementation for Base.f() */ }
    public void h() { y = g(); ... }
}
```

In Java, the **protected** access qualifier means that the protected item (field or method) is visible to any derived class of the base class containing the protected item. It also means that the protected item is visible to methods of other classes in the same package. This is different from C++.

Order of Construction under Inheritance

Note that when you construct an object, the default base class constructor is called implicitly, before the body of the derived class constructor is executed. So, objects are constructed top-down under inheritance. Since every object inherits from the Object class, the Object () constructor is always called implicitly. However, you can call a superclass constructor explicitly using the builtin **super** keyword, as long as it is the *first* statement in a constructor.

For example, most Java exception objects inherit from the java.lang.Exception class. If you wrote your own exception class, say SomeException, you might write it as follows:

```
    public class SomeException extends Exception
    {
        public SomeException()
        {
            super(); // calls Exception(), which ultimately calls Object()
        }
        public SomeException(String s)
        {
            super(s); // calls Exception(String), to pass argument to base class
        }
        public SomeException (int error_code)
        {
                this("error”); // class constructor above, which calls super(s)
                System.err.println(error_code);
        }
    }
```

7.3 Using the Keyword super

Accessing Superclass Members

If your method overrides one of its superclass’s methods, you can invoke the overridden method through the use of the keyword super. You can also use super to refer to a hidden field (although hiding fields is discouraged). Consider this class, Superclass:

```
public class Superclass
{
   public void printMethod()
   {
      System.out.println("Printed in Superclass.");
   }
}
```

Here is a subclass, called Subclass, that overrides printMethod():

```
public class Subclass extends Superclass
{
   // overrides printMethod in Superclass
   public void printMethod()
  {
      super.printMethod();
      System.out.println("Printed in Subclass");
```

```
    }
    public static void main(String[] args)
   {
       Subclass s = new Subclass();
       s.printMethod();
    }
}
```

Within Subclass, the simple name printMethod() refers to the one declared in Subclass, which overrides the one in Superclass. So, to refer to printMethod()inherited from Superclass, Subclass must use a qualified name, using super as shown. Compiling and executing Subclass prints the following:

```
Printed in Superclass.
Printed in Subclass
```

Subclass Constructors

The following example illustrates how to use the super keyword to invoke a superclass's constructor. Recall from the Bicycle example that MountainBike is a subclass of Bicycle. Here is the MountainBike (subclass) constructor that calls the superclass constructor and then adds initialization code of its own:

```
public MountainBike(int startHeight, int startCadence, int startSpeed, int startGear)
{
    super(startCadence, startSpeed, startGear);
    seatHeight = startHeight;
}
```

Invocation of a superclass constructor must be the first line in the subclass constructor.
The syntax for calling a superclass constructor is

```
        super();
        or:
        super(parameter list);
```

With super(), the superclass no-argument constructor is called. With super(parameter list), the superclass constructor with a matching parameter list is called. If a subclass constructor invokes a constructor of its superclass, either explicitly or implicitly, you might think that there will be a whole chain of constructors called, all the way back to the constructor of Object. In fact, this is the case. It is called constructor chaining, and you need to be aware of it when there is a long line of class descent.

7.4 Multilevel Hierarchy

In simple inheritance a subclass or derived class derives the properties from its parent class, but in multilevel inheritance a subclass is derived from a derived class. One class inherits only single class. Therefore, in multilevel inheritance, every time ladder increases by one. The lower most class will have the properties of all the super classes'.

So it will be like this:

Person

↓

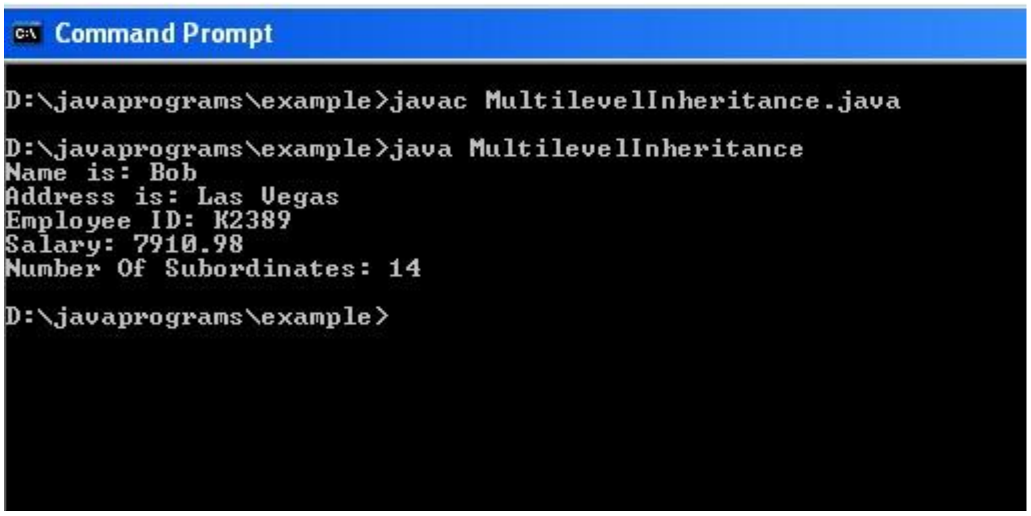
Employee

↓

Manager

We will understand Multilevel Inheritance using the following example:

```
class Person{
  String personName;
  String address;
  Person(String personName,String address){
    this.personName = personName;
    this.address = address;
  }
  void showPersonalDetails(){
    System.out.println("Name is: "+personName);
  }
}
class Employee extends Person{
  String employeeID;
  double salary;
  Employee(String personName,String address,String employeeID,double salary){
    super(personName,address);
    this.employeeID = employeeID;
    this.salary = salary;
  }
}
class Manager extends Employee{
  int numberOfSubordinates;
Manager(String personName,String address,String employeeID,double salary,int numberOfSubordinates){
    super(personName,address,employeeID,salary);
    this.numberOfSubordinates = numberOfSubordinates;
  }
}
class MultileveleInheritance{
  public static void main(String args[]){
    Person p =  new Person();
    Employee e = new Employee();
    Manager m = new Manager();
  }
}
```



```
D:\javaprograms\example>javac MultilevelInheritance.java

D:\javaprograms\example>java MultilevelInheritance
Name is: Bob
Address is: Las Vegas
Employee ID: K2389
Salary: 7910.98
Number Of Subordinates: 14

D:\javaprograms\example>
```

Explanation of Code & Output

Here Person, Employee and Manager are three classes. Manager subclass is derived from Employee subclass which is again derived from Person class. All these classes has one to one relation. So Manager class is the most specialized class among the three which will have the access right to all the members of both Employee and Person. On the other hand, Person super class won't have access to any of the members of its subclasses; same is true for Employee subclass which won't have access to the any members of Manager subclass. All these three classes are written in single java file for your understanding but generally these are put in three different java files.

7.5 When Constructors are called

```
public class Person
{
  private String firstName;
  private String lastName;
  private String address;
  private String username;
  //The constructor method
  public Person()
  {
   }
}
```

The constructor method is like a normal public method except that it shares the same name as the class and it cannot return a value. It can have none, one or many parameters.

Currently the constructor method does nothing at all and it's a good time to consider what this means for the initial state of the Person object. If we left things as they are or we didn't include a constructor method in our Person class (in Java you can define a class without one), then the fields would have no values. If you think there's a chance that your object might not be used as you expect and the fields might not be initialized when the object is created, always define them with a default value:

```
 public class Person
{
   private String firstName = "";
   private String lastName = "";
   private String address = "";
   private String username = "";
   //The constructor method
   public Person()
   {

   }
}
```

Normally for a constructor method to be useful we would design it so that it expects parameters.

Calling the Constructor Method

Unlike other methods of an object the constructor method must be called using the "new" keyword:

```
 public class PersonExample {
   public static void main(String[] args) {
     Person dave = new Person("Dave", "Davidson", "12 Pall Mall", "DDavidson");
```

```
        dave.displayPersonDetails();
       }
     }
```

To create the new instance of the Person object we first define a variable of type Person that will hold the object. In the example I've called it dave. On the other side of the equals sign we are calling the constructor method of our Person class and passing it four string values. Our constructor methods will takes those four values and set the initial state of the Person object to be: firstname = "Dave", lastname = "Davidson", address = "12 Pall Mall", username = "DDavidson".

Notice how I've switched to the Java main class to call the Person object. When you work with objects you will have programs that span multiple .java files. Make sure you save them in the same folder. To compile and run the program nothing has changed, simply compile and run the Java main class file (i.e., PersonExample.java). The Java compiler is smart enough to realize that you want to compile the Person.java file as well because it can see that you have used it in the PersonExample class.

7.6 Method overriding

If a class inherits a method from its super class, then there is a chance to override the method provided that it is not marked final. The benefit of overriding is: ability to define a behavior that's specific to the subclass type which means a subclass can implement a parent class method based on its requirement. In object-oriented terms, overriding means to override the functionality of an existing method.

Example:

Let us look at an example.

```
class Animal{
  public void move(){
    System.out.println("Animals can move");
  }
}
class Dog extends Animal{
  public void move(){
    System.out.println("Dogs can walk and run");
  }
}
public class TestDog{
  public static void main(String args[]){
    Animal a = new Animal(); // Animal reference and object
    Animal b = new Dog(); // Animal reference but Dog object
    a.move();// runs the method in Animal class
    b.move();//Runs the method in Dog class
  }
}
```

This would produce the following result:

```
Animals can move
Dogs can walk and run
```

In the above example, you can see that the even though b is a type of Animal it runs the move method in the Dog class. The reason for this is: In compile time, the check is made on the reference type. However, in the runtime, JVM figures out the object type and would run the method that belongs to that particular object.

Therefore, in the above example, the program will compile properly since Animal class has the method move. Then, at the runtime, it runs the method specific for that object.

Consider the following example:

```
class Animal{
  public void move(){
    System.out.println("Animals can move");
  }
}
class Dog extends Animal{
  public void move(){
    System.out.println("Dogs can walk and run");
  }
  public void bark(){
    System.out.println("Dogs can bark");
  }
}
public class TestDog{
  public static void main(String args[]){
    Animal a = new Animal(); // Animal reference and object
    Animal b = new Dog(); // Animal reference but Dog object
    a.move();// runs the method in Animal class
    b.move();//Runs the method in Dog class
    b.bark();
  }
}
```

This would produce the following result:

```
TestDog.java:30: cannot find symbol
symbol  : method bark()
location: class Animal
           b.bark();
            ^
```

This program will throw a compile time error since b's reference type Animal doesn't have a method by the name of bark.

Rules for method overriding

- The argument list should be exactly the same as that of the overridden method.
- The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
- The access level cannot be more restrictive than the overridden method's access level. For example: if the superclass method is declared public then the overriding method in the sub class cannot be either private or protected.
- Instance methods can be overridden only if they are inherited by the subclass.
- A method declared final cannot be overridden.
- A method declared static cannot be overridden but can be re-declared.
- If a method cannot be inherited, then it cannot be overridden.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- A subclass in a different package can only override the non-final methods declared public or protected.
- An overriding method can throw any uncheck exceptions, regardless of whether the overridden method throws exceptions or not. However the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
- Constructors cannot be overridden.

7.7 Dynamic Method Dispatch

Dynamic Method Dispatch is the mechanism in which call to an overridden method is resolved at run time rather than at compile time. Dynamic Method Dispatch is related to a principle that states that a super class reference can store the reference of subclass object. However, it can't call any of the newly added methods by the subclass but a call to an overridden methods results in calling a method of that object whose reference is stored in the super class reference. It simply means that which method would be executed, simply depends on the object reference stored in super class object. Program written clearly explains above stated principle.

```
package ClassIllustration;
class A
{
 void callme()
 {
  System.out.println("Inside A's callme Method");
 }
}
class B extends A
{
 void callme()
 {
  System.out.println("Inside B's callme Method");
 }
}
class C extends B
{
 void callme()
 {
  System.out.println("Inside C's callme Method");
 }
}
class DynamicMethodDispatch
{
 public static void main(String args[])
 {
  A a = new A();
  B b = new B();
  C c = new C();
  A r;
  r = a;
  r.callme();
  r = b;
  r.callme();
  r = c;
  r.callme();
 }
}
```

As you can see, Reference of class A stores the reference of class A, class B and class C alternatively. Every call to callme() method would result in execution of callme() method of that class whose reference is currently stored by a reference of class A. Here is the output received on executing this program on machine

```
Inside A's callme Method
Inside B's callme Method
Inside C's callme Method
```

7.8 Abstract class in Java

A class that is declared with abstract keyword is known as abstract class. Before learning abstract class, let's understand the abstraction first.

Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

Syntax to declare the abstract class

```
abstract class <class_name>{}
```

Abstract method

A method that is declared as abstract and does not have implementation is known as abstract method

Syntax to define the abstract method

```
abstract return_type <method_name>();//no braces{}
```

Example of abstract class that have abstract method

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike
{
        abstract void run();
}
class Honda extends Bike
{
        void run()
        {
                System.out.println("running safely..");
        }
        public static void main(String args[])
        {
                Bike obj = new Honda();
                obj.run();
        }
}
```
Output: running safely..

Understanding the real scenario of abstract class

In this example, Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the **factory method**. A **factory method** is the method that returns the instance of the class. We will learn about the factory method later. In this example, if you create the instance of Rectangle class, draw method of Rectangle class will be invoked.

```
abstract class Shape
{
        abstract void draw();
}
class Rectangle extends Shape
{
        void draw()
        {
                System.out.println("drawing rectangle");
        }
}
class Circle extends Shape
{
        void draw()
        {
                System.out.println("drawing circle");
        }
}
class Test
```

```
{
        public static void main(String args[])
        {
                Shape s=new Circle();
                //In real scenario, Object is provided through factory method
                s.draw();
        }
}
```

Output: drawing circle

7.9 Final with Inheritance

Keyword final plays many roles in the process of inheritance. It can be used for a class or any method of a class.

Using final with a class

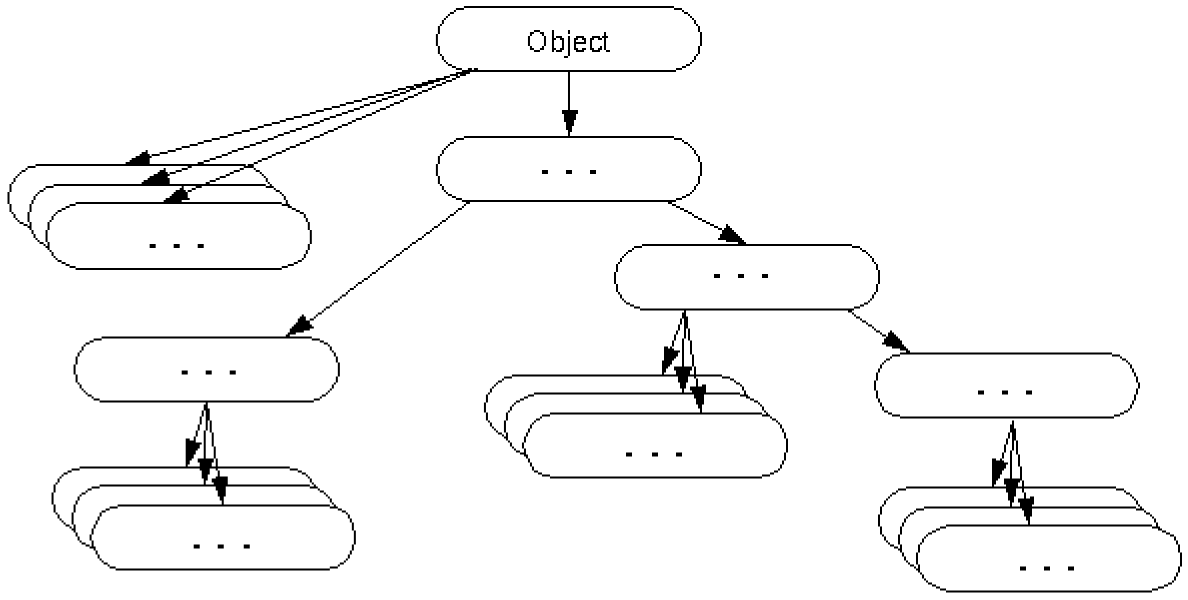
It is possible to declare a normally behaving class as final. However, we cannot use it as a super-class. Any sub-classing is not allowed. There are two ways to look at this. When you think that the development of a class is complete and nobody should modify it by extending it, then you can use this keyword final with that class.

7.10 Object class in Java

The **Object class** is the parent class of all the classes in java bydefault. In other words, it is the topmost class of java. The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, known as upcasting. Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, we can use Object class reference to refer that object. For example:

```
Object obj=getObject (); //we don't what object would be returned from this method
```

The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.



Methods of Object class

The Object class provides many methods. They are as follows:

Method	Description
public final ClassgetClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.

public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified miliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

7.11 Package and Interfaces

Packages

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc. A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and name space management. Some of the existing packages in Java are:

- **java.lang** - bundles the fundamental classes
- **java.io** - classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related. Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

Creating a package

When creating a package, you should choose a name for the package and put a **package** statement with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.

The **package** statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file. If a package statement is not used then the class, interfaces, enumerations, and annotation types will be put into an unnamed package.

Example

Let us look at an example that creates a package called animals. It is common practice to use lowercased names of packages to avoid any conflicts with the names of classes, interfaces.

Put an interface in the package animals:

```
package animals;
interface Animal {
   public void eat();
   public void travel();
}
```

Now, put an implementation in the same package *animals*:

```
package animals;
```

```
public class MammalInt implements Animal{
  public void eat(){
    System.out.println("Mammal eats");
  }
  public void travel(){
    System.out.println("Mammal travels");
  }
  public int noOfLegs(){
    return 0;
  }
  public static void main(String args[]){
    MammalInt m = new MammalInt();
    m.eat();
    m.travel();
  }
}
```

7.12 The import Keyword

If a class wants to use another class in the same package, the package name does not need to be used. Classes in the same package find each other without any special syntax.

Example

Here, a class named Boss is added to the payroll package that already contains Employee. The Boss can then refer to the Employee class without using the payroll prefix, as demonstrated by the following Boss class.

```
package payroll;
public class Boss
{
  public void payEmployee(Employee e)
  {
    e.mailCheck();
  }
}
```

What happens if Boss is not in the payroll package? The Boss class must then use one of the following techniques for referring to a class in a different package. The fully qualified name of the class can be used. For example:

```
payroll.Employee
```

The package can be imported using the import keyword and the wild card (*). For example:

```
import payroll.*;
```

The class itself can be imported using the import keyword. For example:

```
import payroll.Employee;
```

7.13 Interface

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class. An interface is similar to a class in the following ways:

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The bytecode of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including:

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.

- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaring Interfaces

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface:

Example

Let us look at an example that depicts encapsulation:

```
import java.lang.*;
//Any number of import statements
public interface NameOfInterface
{
  //Any number of final, static fields
  //Any number of abstract method declarations\
}
```

Interfaces have the following properties:

- An interface is implicitly abstract. You do not need to use the **abstract** keyword when declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Example

```
interface Animal {
  public void eat();
  public void travel();
}
```

Implementing Interfaces

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract. A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

```
public class MammalInt implements Animal{
  public void eat(){
    System.out.println("Mammal eats");
  }
  public void travel(){
    System.out.println("Mammal travels");
  }
  public int noOfLegs(){
    return 0;
  }
  public static void main(String args[]){
    MammalInt m = new MammalInt();
    m.eat();
    m.travel();
  }
}
```

This would produce the following result:

```
Mammal eats
Mammal travels
```

When overriding methods defined in interfaces there are several rules to be followed:

- Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
- The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
- An implementation class itself can be abstract and if so interface methods need not be implemented.

When implementation interfaces there are several rules:

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, similarly to the way that a class can extend another class.

Extending Interfaces

An interface can extend another interface, similarly to the way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.

```
public interface Sports
{
  public void setHomeTeam(String name);
  public void setVisitingTeam(String name);
}
public interface Football extends Sports
{
  public void homeTeamScored(int points);
  public void visitingTeamScored(int points);
  public void endOfQuarter(int quarter);
}
public interface Hockey extends Sports
{
  public void homeGoalScored();
  public void visitingGoalScored();
  public void endOfPeriod(int period);
  public void overtimePeriod(int ot);
}
```

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

Extending Multiple Interfaces

A Java class can only extend one parent class. Multiple inheritances are not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface. The extends keyword is used once, and the parent interfaces are declared in a comma-separated list. For example, if the Hockey interface extended both Sports and Event, it would be declared as:

```
public interface Hockey extends Sports, Event
```

Tagging Interfaces

The most common use of extending interfaces occurs when the parent interface does not contain any methods. For example, the MouseListener interface in the java.awt.event package extended java.util.EventListener, which is defined as:

```
package java.util;
public interface EventListener
{}
```

An interface with no methods in it is referred to as a **tagging** interface. There are two basic design purposes of tagging interfaces:

Creates a common parent: As with the EventListener interface, which is extended by dozens of other interfaces in the Java API, you can use a tagging interface to create a common parent among a group of interfaces. For example, when an interface extends EventListener, the JVM knows that this particular interface is going to be used in an event delegation scenario.

Adds a data type to a class: This situation is where the term tagging comes from. A class that implements a tagging interface does not need to define any methods (since the interface does not have any), but the class becomes an interface type through polymorphism.

7.14 Summary

In this unit, you have learned about inheritance, packages and interfaces. How to use the super keyword and Multilevel Hierarchy, calling a constructors, Method overriding, Dynamic method dispatch, Abstract classes, final with inheritance, Object class are also discussed in this unit. This unit also has packages and Interfaces, in this topic how to import a packages and Interface helps in inheritance for reusing the code that details are discussed in this unit.

7.15 Key words

- **Adds a data type to a class:** This situation is where the term tagging comes from.
- **Object class** is the parent class of all the classes in java bydefault.

7.16 Unit end exercise and answers

1. What is inheritance? Explain in detail.
2. How can be super keyword is used?
3. What is multilevel hierarchy explain?
4. How constructor works and what happens when constructor are called?
5. Explain Method overriding?
6. Expalin Dynamic method dispatch and Abstract classesin java
7. How final with inheritance works?
8. What is Object class?
9. Explain about Packages and Interfaces?
10. How Import keywordis used in package,

11. Explain about Interface in java?

Answers :see

1. 7.2
2. 7.3
3. 7.4
4. 7.5
5. 7.6
6. 7.7,7.8
7. 7.9
8. 7.10
9. 7.11
10. 7.12
11. 7.13

7.17 Reference

Teach yourself Java - Joseph O’Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 2006. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 8: Exception Handling

Structure:

8.0 OBJECTIVES

At the end of this unit you will be able to know:

- Exception handling fundamentals
- Exception Methods
- try and catch
- Multiple catch clauses
- The throws/throw Keywords and finally Keyword
- Built-in exception
- Declaring you own Exception

- Chained exceptions
- Advantages of Exceptions

8.1 Introduction

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner. To understand how exception handling works in Java, you need to understand the three categories of exceptions:

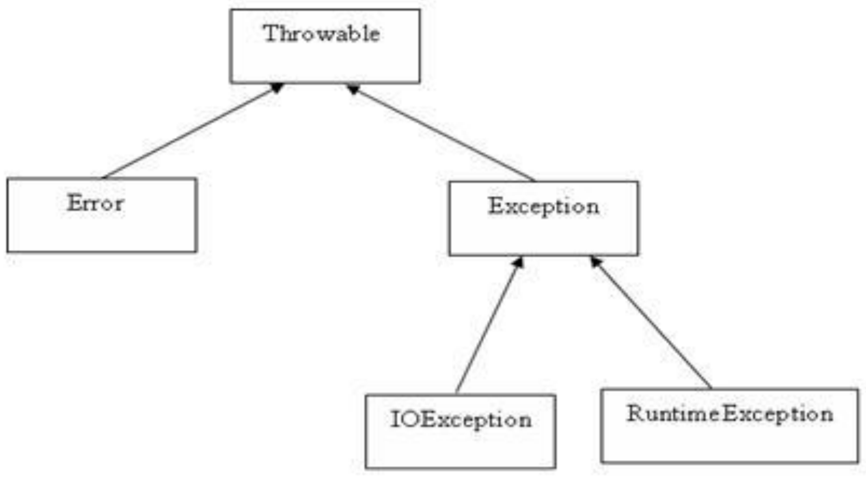
- **Checked exceptions:** A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

8.2 Exception handling fundamentals

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Errors are not normally trapped form the Java programs. These conditions normally happen in case of severe failures, which are not handled by the java programs. Errors are generated to indicate errors generated by the runtime environment. Example : JVM is out of Memory. Normally programs cannot recover from errors.

The Exception class has two main subclasses: IOException class and RuntimeException Class.



Here is a list of most common checked and unchecked Java's Built-in Exceptions.

8.3 Exceptions Methods

Following is the list of important methods available in the Throwable class.

SN	Methods with Description
1	**public String getMessage()** Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
2	**public Throwable getCause()** Returns the cause of the exception as represented by a Throwable object.
3	**public String toString()** Returns the name of the class concatenated with the result of getMessage()
4	**public void printStackTrace()**

	Prints the result of toString() along with the stack trace to System.err, the error output stream.
5	**public StackTraceElement [] getStackTrace()** Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	**public Throwable fillInStackTrace()** Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

8.4 try catch exception

A method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

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

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follow the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Example: The following is an array is declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
import java.io.*;
public class ExcepTest
        {
                public static void main(String args[])
                {
                        try
                        {
                                int a[] = new int[2];
                                System.out.println("Access element three :" + a[3]);
                        }
                        catch(ArrayIndexOutOfBoundsException e)
                        {
                                System.out.println("Exception thrown  :" + e);
                        }
                        System.out.println("Out of the block");
                }
        }
```

This would produce the following result:

```
Exception thrown: java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

8.5 Multiple Catch Blocks

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

```
try
        {
```

```
                    //Protected code
          }
catch(ExceptionType1 e1)
          {
                    //Catch block
          }
catch(ExceptionType2 e2)
          {
                    //Catch block
          }
catch(ExceptionType3 e3)
          {
                    //Catch block
          }
```

The previous statements demonstrate three catch blocks, but you can have any number of them after a single try. If an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

Example

Here is code segment showing how to use multiple try/catch statements.

```
try
          {
                    file = new FileInputStream(fileName);
                    x = (byte) file.read();
          }
catch(IOException i)
```

```
        {
                i.printStackTrace();
                return -1;
        }
catch(FileNotFoundException f) //Not valid!
        {
                f.printStackTrace();
                return -1;
        }
```

8.6 The throws/throw Keywords and finally Keyword

If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature. You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the throw keyword. Try to understand the different in throws and throw keywords.

The following method declares that it throws a RemoteException:

```
import java.io.*;
public class className
        {
                public void deposit(double amount) throws RemoteException
                {
                        // Method implementation
                        throw new RemoteException();
                }
                //Remainder of class definition
        }
```

A method can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas. For example, the following method declares that it throws a RemoteException and an InsufficientFundsException:

```
import java.io.*;
public class className
{
    public void withdraw(double amount) throws RemoteException,
                                         InsufficientFundsException
    {
        // Method implementation
    }
    //Remainder of class definition
}
```

The finally Keyword

The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred. Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code. A finally block appears at the end of the catch blocks and has the following syntax:

```
try
        {
                //Protected code
        }
catch(ExceptionType1 e1)
        {
                //Catch block
        }
catch(ExceptionType2 e2)
        {
                //Catch block
        }
catch(ExceptionType3 e3)
        {
                //Catch block
        }
finally
        {
                //The finally block always executes.
        }
```

Example

```
public class ExcepTest
        {
                public static void main(String args[])
                {
                        int a[] = new int[2];
                        try
                        {
                                System.out.println("Access element three :" + a[3]);
                        }
                        catch(ArrayIndexOutOfBoundsException e)
                        {
                                System.out.println("Exception thrown  :" + e);
                        }
                        finally
                        {
                                a[0] = 6;
                                System.out.println("First element value: " +a[0]);
                                System.out.println("The finally statement is executed");
                        }
                }
        }
```

This would produce the following result:

```
Exception thrown: java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

Note the following:

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block is present.
- The try block cannot be present without either catch clause or finally clause.
- Any code cannot be present in between the try, catch, finally blocks.

8.7 Built-in exception

When an error is detected, an exception is thrown. That is, the code that caused the error stops executing immediately, and control is transferred to the catch clause for that exception of the first enclosing try block that has such a clause. The try block might be in the current function (the one that caused the error), or it might be in some function that called the current function (i.e., if the current function is not prepared to handle the exception, it is "passed up" the call chain). If no currently active function is prepared to catch the exception, an error message is printed and the program stops.

Exceptions can be built-in (actually, defined in one of Java's standard libraries) or user-defined. Here are some examples of built-in exceptions:

- ArithmeticException (e.g., divide by zero)
- ClassCastException (e.g., attempt to cast a String Object to Integer)
- IndexOutOfBoundsException
- NullPointerException
- FileNotFoundException (e.g., attempt to open a non-existent file for reading)
-

8.8 Declaring you own Exception

You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes:

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

We can define our own Exception class as below:

```
class MyException extends Exception
{


}
```

You just need to extend the Exception class to create your own Exception class. These are considered to be checked exceptions. The following InsufficientFundsException class is a user-defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.

Example

```
// File Name InsufficientFundsException.java
import java.io.*;
public class InsufficientFundsException extends Exception
{
    private double amount;
    public InsufficientFundsException(double amount)
    {
        this.amount = amount;
    }
    public double getAmount()
    {
      return amount;
   }
}
```

To demonstrate using our user-defined exception, the following CheckingAccount class contains a withdraw() method that throws an InsufficientFundsException.

```
// File Name CheckingAccount.java
import java.io.*;
public class CheckingAccount
{
   private double balance;
   private int number;
   public CheckingAccount(int number)
   {
      this.number = number;
   }
   public void deposit(double amount)
   {
      balance += amount;
   }
   public void withdraw(double amount) throws
                              InsufficientFundsException
   {
      if(amount <= balance)
      {
         balance -= amount;
      }
      else
      {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }
   public double getBalance()
   {
      return balance;
   }
   public int getNumber()
   {
      return number;
   }
}
```

The following BankDemo program demonstrates invoking the deposit() and withdraw() methods of CheckingAccount.

```
// File Name BankDemo.java
public class BankDemo
{
   public static void main(String [] args)
   {
      CheckingAccount c = new CheckingAccount(101);
      System.out.println("Depositing $500...");
      c.deposit(500.00);
      try
      {
         System.out.println("\nWithdrawing $100...");
         c.withdraw(100.00);
         System.out.println("\nWithdrawing $600...");
         c.withdraw(600.00);
      }catch(InsufficientFundsException e)
      {
         System.out.println("Sorry, but you are short $"
                                  + e.getAmount());
         e.printStackTrace();
      }
    }
}
```

Compile all the above three files and run BankDemo, this would produce the following result:

```
Depositing $500...
Withdrawing $100...
Withdrawing $600...
Sorry, but you are short $200.0
InsufficientFundsException
        at CheckingAccount.withdraw(CheckingAccount.java:25)
        at BankDemo.main(BankDemo.java:13)
```

Common Exceptions

In Java, it is possible to define two categories of Exceptions and Errors.

- **JVM Exceptions:** - These are exceptions/errors that are exclusively or logically thrown by the JVM. Examples: NullPointerException, ArrayIndexOutOfBoundsException, ClassCastException,
- **Programmatic exceptions:** - These exceptions are thrown explicitly by the application or the API programmers Examples: IllegalArgumentException, IllegalStateException.

8.9 Chained Exceptions

An application often responds to an exception by throwing another exception. In effect, the first exception *causes* the second exception. It can be very helpful to know when one exception causes another. *Chained Exceptions* help the programmer do this.

The following are the methods and constructors in `Throwable` that support chained exceptions.

Throwable getCause()
Throwable initCause(Throwable)
Throwable(String, Throwable)
Throwable(Throwable)

The `Throwable` argument to `initCause` and the `Throwable` constructors is the exception that caused the current exception. `getCause` returns the exception that caused the current exception, and `initCause` sets the current exception's cause.

The following example shows how to use a chained exception.

```
try
{


}
catch (IOException e)
{
   throw new SampleException("Other IOException", e);
}
```

In this example, when an `IOException` is caught, a new `SampleException` exception is created with the original cause attached and the chain of exceptions is thrown up to the next higher level exception handler.

8.10 Advantages of Exceptions

Advantage 1: Separating Error-Handling Code from "Regular" Code

Exceptions provide the means to separate the details of what to do when something out of the ordinary happens from the main logic of a program. In traditional programming, error detection, reporting, and handling often lead to confusing spaghetti code. For example, consider the pseudo code method here that reads an entire file into memory.

```
readFile {
   open the file;
   determine its size;
   allocate that much memory;
   read the file into memory;
   close the file;
}
```

At first glance, this function seems simple enough, but it ignores all the following potential errors.

- What happens if the file can't be opened?
- What happens if the length of the file can't be determined?
- What happens if enough memory can't be allocated?

- What happens if the read fails?
- What happens if the file can't be closed?

To handle such cases, the `readFile` function must have more code to do error detection, reporting, and handling. Here is an example of what the function might look like.

```
errorCodeType readFile {
    initialize errorCode = 0;

    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        } else {
            errorCode = errorCode and -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```

There's so much error detection, reporting, and returning here that the original seven lines of code are lost in the clutter. Worse yet, the logical flow of the code has also been lost, thus making it difficult to tell whether the code is doing the right thing: Is the file really being closed if the function fails to allocate enough memory? It's even more difficult to ensure that the code continues to do the right thing when you modify the method three months after writing it. Many programmers solve this problem by simply ignoring it — errors are reported when their programs crash.

Exceptions enable you to write the main flow of your code and to deal with the exceptional cases elsewhere. If the `readFile` function used exceptions instead of traditional error-management techniques, it would look more like the following.

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
       doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

Advantage 2: Propagating Errors Up the Call Stack

Second advantage of exceptions is the ability to propagate error reporting up the call stack of methods. Suppose that the readFile method is the fourth method in a series of nested method calls made by the main program: method1 calls method2, which calls method3, which finally calls readFile.

```
method1 {
    call method2;
}
method2 {
    call method3;
}
method3 {
    call readFile;
}
```

Suppose also that method1 is the only method interested in the errors that might occur within readFile. Traditional error-notification techniques force method2 and method3to propagate the error codes returned by readFile up the call stack until the error codes finally reach method1—the only method that is interested in them.

```
method1 {
    errorCodeType error;
    error = call method2;
    if (error)
        doErrorProcessing;
    else
        proceed;
}
errorCodeType method2 {
    errorCodeType error;
    error = call method3;
    if (error)
        return error;
    else
        proceed;
}
errorCodeType method3 {
    errorCodeType error;
    error = call readFile;
    if (error)
        return error;
    else
        proceed;
}
```

Recall that the Java runtime environment searches backward through the call stack to find any methods that are interested in handling a particular exception. A method can duck any exceptions thrown within it, thereby allowing a method farther up the call stack to catch it. Hence, only the methods that care about errors have to worry about detecting errors.

```
method1 {
    try {
        call method2;
    } catch (exception e) {
        doErrorProcessing;
    }
}
method2 throws exception {
    call method3;
}
method3 throws exception {
    call readFile;
}
```

However, as the pseudocode shows, ducking an exception requires some effort on the part of the middleman methods. Any checked exceptions that can be thrown within a method must be specified in its throws clause.

Advantage 3: Grouping and Differentiating Error Types

Because all exceptions thrown within a program are objects, the grouping or categorizing of exceptions is a natural outcome of the class hierarchy. An example of a group of related exception classes in the Java platform are those defined in `java.io` — `IOException` and its descendants. `IOException` is the most general and represents any type of error that can occur when performing I/O. Its descendants represent more specific errors. For example, `FileNotFoundException` means that a file could not be located on disk.

A method can write specific handlers that can handle a very specific exception. The `FileNotFoundException` class has no descendants, so the following handler can handle only one type of exception.

```
catch (FileNotFoundException e) {
    ...
}
```

A method can catch an exception based on its group or general type by specifying any of the exception's superclasses in the `catch` statement. For example, to catch all I/O exceptions, regardless of their specific type, an exception handler specifies an `IOException` argument.

```
catch (IOException e) {
    ...
}
```

This handler will be able to catch all I/O exceptions,
including `FileNotFoundException`, `EOFException`, and so on. You can find details about what occurred by querying the argument passed to the exception handler. For example, use the following to print the stack trace.

```
catch (IOException e) {
    // Output goes to System.err.
    e.printStackTrace();
    // Send trace to stdout.
    e.printStackTrace(System.out);
}
```

You could even set up an exception handler that handles any `Exception` with the handler here.

```
// A (too) general exception handler
catch (Exception e) {
    ...
}
```

The `Exception` class is close to the top of the `Throwable` class hierarchy. Therefore, this handler will catch many other exceptions in addition to those that the handler is intended to catch. You may want to handle exceptions this way if all you want your program to do, for example, is print out an error message for the user and then exit.

In most situations, however, you want exception handlers to be as specific as possible. The reason is that the first thing a handler must do is determine what type of exception occurred before it can decide on the best recovery strategy. In effect, by not catching specific errors, the handler must accommodate any possibility. Exception handlers that are too general can make code more error-prone by catching and handling exceptions that weren't anticipated by the programmer and for which the handler was not intended.

8.11 Summary

In this unit, you have learned the importance of exception and exception handling. It begins with fundamental of exception handling and exception methods. Try and catch, multiple catch clauses are the exception handling methods discussed in this unit. How throw and finally keywords are used in exception, built in exception are discussed. Also how to declare own exception, chained exception and advantages of exception are discussed in this unit.

8.12 Key words

- **JVM Exceptions: -** These are exceptions/errors that are exclusively or logically thrown by the JVM. Examples: NullPointerException, ArrayIndexOutOfBoundsException, ClassCastException,
- **Programmatic exceptions: -** These exceptions are thrown explicitly by the application or the API programmers Examples: IllegalArgumentException, IllegalStateException.

8.13 Unit end exercise and answers

1. What is an exception and explain Exception handling fundamentals?
2. Explain Exception Methods in detail? ,
3. Explain how try and catch and multiple catch clauses are used in exception?
4. What is the use of throws/throw Keywords and finally Keyword in exception?
5. Explain about Built-in exception?
6. How to Declaring you own Exception?
7. What is chained exceptions? Explain.
8. What are the Advantages of Exceptions?

Answers :see

1. 8.2
2. 8.3
3. 8.4,8.5
4. 8.6
5. 8.7
6. 8.8
7. 8.9
8. 8.10

8.14 Reference

Teach yourself Java - Joseph O'Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 2006. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

Module-3

UNIT 9: Multithreaded Programming

Structure:

9.0 OBJECTIVES

At the end of this unit you will be able to know:

- Thread model
- Main thread and Creating a thread
- Multiple thread
- Priorities
- Synchronization
- Interthread communication
- Suspending-Resuming-Stopping threads
- Multithreading.

9.1 INTRODUCTION

Modern operating systems hold more than one activity (program) in memory and the processor can switch among all to execute them. This simultaneous occurrence of several activities on a computer is known as multitasking. For example, you can open a fi le using MS Word and you can work on MS Access for creating your database. Two applications, MS Word and MS Access, are available in memory and the processor (by means of the operating system) switches to the application you are actively working on.

Here two different applications are made to run concurrently by the same processor. The operating system supports multitasking in a cooperative or preemptive manner. In cooperative multitasking each application is responsible for relinquishing control to the processor to enable it to execute the other application.

Earlier versions of operating systems followed cooperative multitasking. Modern operating systems such as Windows 95, Windows 98, Windows NT, and Windows 2000 support preemptive multitasking. In the preemptive type multitasking, the processor is responsible for executing each application in a certain amount of time called a time slice. The processor then switches to the other applications, giving each its time slice. The programmer is relieved from the burden of relinquishing control from one application to another. The operating system takes care of it.

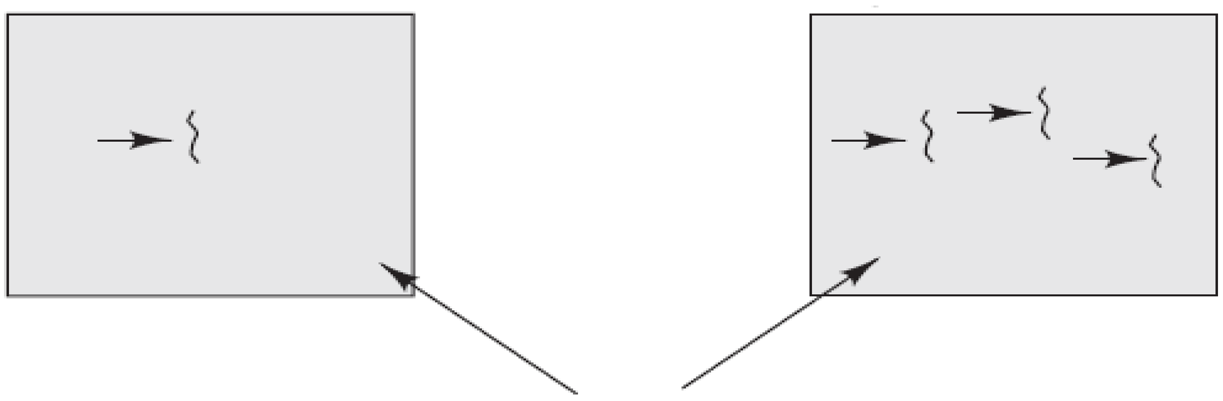
A single processor computer is shared among multiple applications with preemptive multitasking.

Since the processor is switching between the applications at intervals of milliseconds, you feel that all applications run concurrently. Actually, this is not so. To have true multitasking, the applications must be run on a machine with multiple processors. Multitasking results in effective and simultaneous utilization of various system resources such as processors, disks, and printers. As multitasking is managed by operating systems, we encourage readers to refer to books related to that topic. In this chapter, we focus on learning how to write an application containing multiple tasks (i.e., objects containing operations to be performed) that can be executed concurrently. In Java, this is realized by using multithreading techniques.

9.2 Thread model

To understand multithreading, the concepts process and thread must be understood. A process is a program in execution. A process may be divided into a number of independent units known as threads. A thread is a dispatch able unit of work. Threads are light-weight processes within a process. A process is a collection of one or more threads and associated system resources. The difference between a process and a thread is shown in Fig.9.9. A process may have a number of threads in it and thread may be assumed as a subset of a process.

Single Threaded Process Threads of execution Multiple Threaded process



Single instruction Stream Common multiple instruction stream

Fig 9.1

If two applications are run on a computer (MS Word, MS Access), two processes are created. Multitasking of two or more processes is known as process-based multitasking. Multitasking of two or more threads is known as thread-based multitasking. The concept of multithreading in a programming language refers to thread-based multitasking. Process-based multitasking is totally controlled by the operating system. But thread-based multitasking can be controlled by the programmer to some extent in a program.

The concept of context switching is integral to threading. A hardware timer is used by the processor to determine the end of the time slice for each thread. The timer signals at the end of the time slice and in turn the processor saves all information required for the current thread onto a stack. Then the processor moves this information from the stack into a predefined data structure called a context structure. When the processor wants to switch back to a previously executing thread, it transfers all the information from the context structure associated with the thread to the stack. This entire procedure is known as context switching.

Java supports thread-based multitasking. The advantages of thread-based multitasking as compared to process-based multitasking are given below:

- Threads share the same address space.
- Context-switching between threads is normally inexpensive.
- Communication between threads is normally inexpensive.

9.3 Main Thread

The ‘main ()' method in Java is referred to the thread that is running, whenever a Java program runs. It calls the main thread because it is the first thread that starts running when a program begins. Other threads can be spawned from this main thread. The main thread must be the last thread in the program to end. When the main thread stops, the program stops running.

Main thread is created automatically, but it can be controlled by the program by using a Thread object. The Thread object will hold the reference of the main thread with the help of currentThread() method of the Thread class.

Applications are typically divided into processes during the design phase, and a master process explicitly spawns sub processes when it makes sense to logically separate significant application functionalities. Processes, in other words, are an architectural construct. By contrast, a thread is a coding construct that does not affect the architecture of an application. A single process might contain multiple threads (see Fig. 9.2). All threads within a process share the same state and same memory space, and can communicate with each other directly, because they share the same variables.

Threads typically are spawned for a short-term benefit that is usually visualized as a serial task, but which does not have to be performed in a linear manner (such as performing a complex mathematical computation using parallelism, or initializing a large matrix), and then are absorbed when no longer required. The scope of a thread is within a specific code module—which is why we can bolt on threading without affecting the broader application.

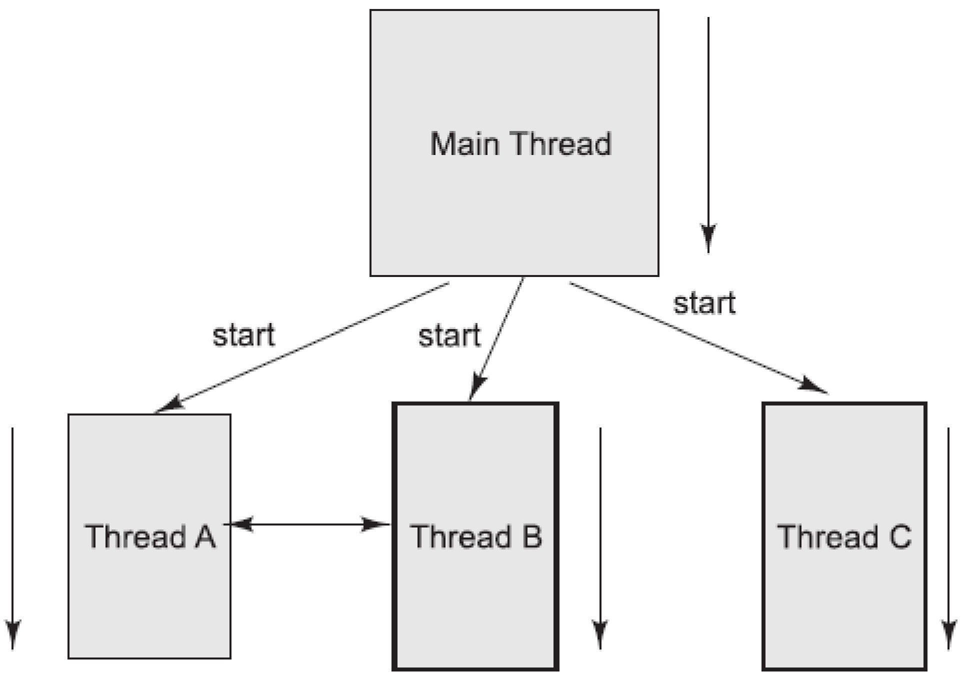


Fig 9.2

The below program shows how main thread can be controlled by a program.

```
class MainThread
{
   public static void main(String args [] )
   {
      Thread t = Thread.currentThread ( );

      System.out.println ("Current Thread : " + t);
      System.out.println ("Name : " + t.getName ( ) );
      System.out.println (" ");

      t.setName ("New Thread");
      System.out.println ("After changing name");
      System.out.println ("Current Thread : " + t);
      System.out.println ("Name : " + t.getName ( ) );
```

```
        System.out.println (" ");

        System.out.println ("This thread prints first 10 numbers");

        try
        {
           for (int i=1; i<=10;i++)
           {
              System.out.print(i);
              System.out.print(" ");
              Thread.sleep(1000);
           }
        }
        catch (InterruptedException e)
        {
           System.out.println(e);
        }
     }
}
```

The output of this program is given below:

```
Current Thread: Thread [main, 5, main]
Name: main

After changing name
Current Thread: Thread [New Thread, 5, main]
Name: New Thread

This thread prints first 10 numbers
1 2 3 4 5 6 7 8 9 10
```

Let us now understand the working of the program in the listing. The program first creates a Thread object called't' and assigns the reference of current thread (main thread) to it. So now main thread can be accessed via Thread object't'. This is done with the help of currentThread () method of Thread class which return a reference to the current running thread. The Thread object't' is then printed as a result of which you see the output Current Thread: Thread [main, 5, main]. The first value in the square brackets of this output indicates the name of the thread, the name of the group to which the thread belongs. The program then prints the name of the thread with the help of getName() method.

The name of the thread is changed with the help of setName() method. The thread and thread name is then again printed. Then the thread performs the operation of printing first 10 numbers. When you run the program you will that the system wait for some time after printing each number. This is caused by the statement Thread.sleep (1000). The sleep () method can also use in another way in which it accepts two arguments. The first argument represents the amount of milliseconds and the second argument represents the amount of nanoseconds. Effectively, the thread will sleep for specified milliseconds + nanoseconds. Its (sleep () method's) prototype is as follows:
Static void sleep (long milliseconds, int nanoseconds).

9.4 Creating a Thread

Threads are objects in the Java language. They can be created by using two different mechanisms as illustrated in Fig. 9.3:
9. Create a class that extends the standard Thread class.
10. Create a class that implements the standard Runnable interface.
That is, a thread can be defined by extending the java.lang.Thread class or by implementing the java.lang.Runnable interface. The run () method should be overridden and should contain the code that will be executed by the new thread. This method must be public with a void return type and should not take any arguments. Both threads and processes are abstractions for parallelizing an application. However, processes are independent execution units that contain their own state information, use their own address spaces, and only interact with each other via inter process communication mechanisms generally managed by the operating system.

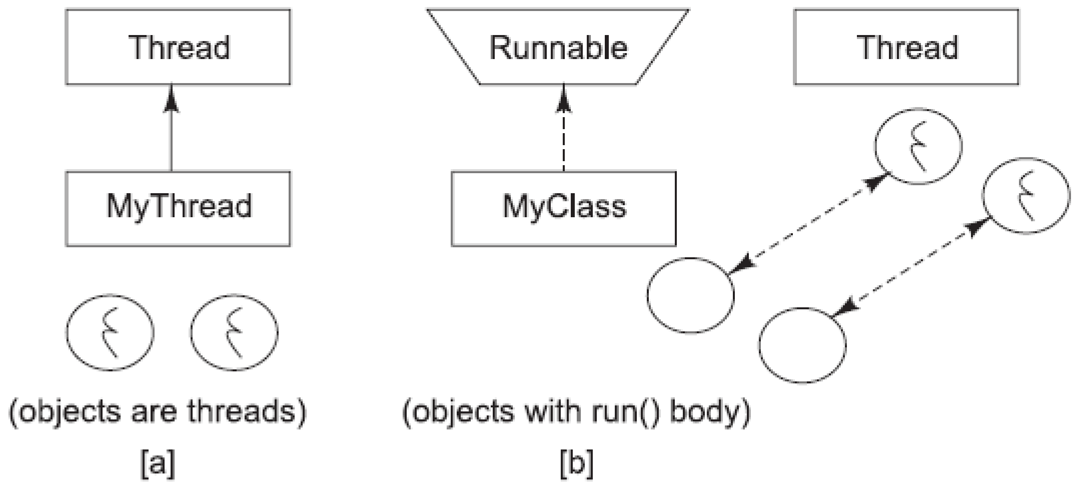


Fig 9.3

Extending the Thread Class

The steps for creating a thread by using the fi rst mechanism are:

Create a class by extending the Thread class and override the run() method:

```
class MyThread extends Thread {
public void run() {
// thread body of execution
}
}
```

10. Create a thread object:

```
MyThread thr1 = new MyThread();
```

11. Start Execution of created thread:

```
thr9.start();
```

An example program illustrating creation and invocation of a thread object is given below:

```
/* ThreadEx9.java: A simple program creating and invoking a thread object by
extending the standard Thread class. */
class MyThread extends Thread {
public void run() {
System.out.println(“ this thread is running ... ”);
}
}
```

```
class ThreadEx1 {
public static void main(String [] args ) {
MyThread t = new MyThread();
t.start();
}
}
```

The class MyThread extends the standard Thread class to gain thread properties through inheritance. The user needs to implement their logic associated with the thread in the run () method, which is the body of thread. The objects created by instantiating the class MyThread are called threaded objects. Even though the execution method of thread is called run, we do not need to explicitly invoke this method directly. When the start () method of a threaded object is invoked, it sets the concurrent execution of the object from that point onward along with the execution of its parent thread/method.

Implementing the Runnable Interface

The steps for creating a thread by using the second mechanism are:

9. Create a class that implements the interface Runnable and override run() method:

```
class MyThread implements Runnable {
…
public void run() {
// thread body of execution
}
}
```

10. Creating Object:

```
MyThread myObject = new MyThread();
```

11. Creating Thread Object:

```
Thread thr1 = new Thread(myObject);
```

12. Start Execution:

```
thr9.start();
```

An example program illustrating creation and invocation of a thread object is given below:

```
/* ThreadEx10.java: A simple program creating and invoking a thread object by
implementing Runnable interface. */
```

```
class MyThread implements Runnable {
public void run() {
System.out.println(“ this thread is running ... ”);
}
}
class ThreadEx2 {
public static void main(String [] args ) {
Thread t = new Thread(new MyThread());
t.start();
}
}
```

The class MyThread implements standard Runnable interface and overrides the run() method and includes logic associated with the body of the thread (step 1). The objects created by instantiating the class MyThread are normal objects (unlike the fi rst mechanism) (step 2). Therefore, we need to create a generic Thread object and pass MyThread object as a parameter to this generic object (step 3). As a result of this association, threaded object is created. In order to execute this threaded object, we need to invoke its start() method which sets execution of the new thread (step 4).

9.5 Priorities

In Java, thread scheduler can use the thread priorities in the form of integer value to each of its thread to determine the execution schedule of threads. When a Java thread is created, it inherits its priority from the thread that created it. You can also modify a thread's priority at any time after its creation using the setPriority() method. Thread priorities are integers ranging between MIN_PRIORITY and MAX_PRIORITY (constants defined in the Thread class). The higher the integer, the higher the priority. At any given time, when multiple threads are ready to be executed, the runtime system chooses the "Runnable" thread with the highest priority for execution. Only when that thread stops, yields, or becomes "Not Runnable" for some reason will a lower priority thread start executing. If two threads of the same priority are waiting for the

CPU, the scheduler chooses one of them to run in a round-robin fashion. The chosen thread will run until one of the following conditions is true:

- a higher priority thread becomes "Runnable"
- it yields, or its run() method exits
- on systems that support time-slicing, its time allotment has expired

Then the second thread is given a chance to run, and so on, until the interpreter exits.
The Java runtime system's thread scheduling algorithm is also preemptive. If at any time a thread with a higher priority than all other "Runnable" threads becomes "Runnable", the runtime system chooses the new higher priority thread for execution. The new higher priority thread is said to preempt the other threads.
Thread gets the ready-to-run state according to their priorities. The thread scheduler provides the CPU time to thread of highest priority during ready-to-run state.
Priorities are integer values from 1 (lowest priority given by the constant Thread.MIN_PRIORITY) to 10 (highest priority given by the constant Thread.MAX_PRIORITY). The default priority is 5(Thread.NORM_PRIORITY).

9.6 Synchronization

Every thread has a life of its own. Normally, a thread goes about its business without any regard for what other threads in the application are doing. Threads may be time-sliced, which means they can run in arbitrary spurts and bursts as directed by the operating system. On a multiprocessor system, it is even possible for many different threads to be running simultaneously on different CPUs. This section is about coordinating the activities of two or more threads, so they can work together and not collide in their use of the same address space.

Java provides a few simple structures for synchronizing the activities of threads. They are all based on the concept of monitors, a widely used synchronization scheme developed by C.A.R.

Hoare. You don't have to know the details about how monitors work to be able to use them, but it may help you to have a picture in mind.

A monitor is essentially a lock. The lock is attached to a resource that many threads may need to access, but that should be accessed by only one thread at a time. It's not unlike a public restroom at a gas station. If the resource is not being used, the thread can acquire the lock and access the resource. By the same token, if the restroom is unlocked, you can enter and lock the door. When the thread is done, it relinquishes the lock, just as you unlock the door and leave it open for the next person. However, if another thread already has the lock for the resource, all other threads have to wait until the current thread finishes and releases the lock, just as if the restroom is locked when you arrive, you have to wait until the current occupant is done and unlocks the door.

Fortunately, Java makes the process of synchronizing access to resources quite easy. The language handles setting up and acquiring locks; all you have to do is specify which resources require locks.

9.7 Inter thread communication

Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other. Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. Consider the classic queuing problem, where one thread is producing some data and another is consuming it. To make the problem more interesting, suppose that the producer has to wait until the consumer is finished before it generates more data.

In a polling system, the consumer would waste many CPU cycles while it waited for the producer to produce. Once the producer was finished, it would start polling, wasting more CPU cycles waiting for the consumer to finish, and so on. Clearly, this situation is undesirable.

To avoid polling, Java includes an elegant inter process communication mechanism via the following methods:

- wait( ): This method tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).

- notify( ): This method wakes up the first thread that called wait( ) on the same object.

- notifyAll( ): This method wakes up all the threads that called wait( ) on the same object. The highest priority thread will run first.

These methods are implemented as final methods in Object, so all classes have them. All three methods can be called only from within a synchronized context. These methods are declared within Object.

9.8 Suspending-Resuming-Stopping threads

When the sleep () method time is over, the thread becomes implicitly active. Sleep () method is preferable when the inactive time is known earlier. Sometimes, the inactive time or blocked time may not be known to the programmer earlier; to come to the task here comes suspend () method. The suspended thread will be in blocked state until resume () method is called on it. These methods are deprecated, as when not used with precautions, the thread locks, if held, are kept in inconsistent state or may lead to deadlocks. While the suspend( ), resume( ), and stop( ) methods defined by Thread class seem to be a perfectly reasonable and convenient approach to managing the execution of threads, they must not be used for new Java programs and obsolete in newer versions of Java.
The following example illustrates how the wait( ) and notify( ) methods that are inherited from Object can be used to control the execution of a thread.

This example is similar to the program in the previous section. However, the deprecated method calls have been removed. Let us consider the operation of this program.

```
class MyThread implements Runnable
 {
  Thread thrd;
  boolean suspended;
  boolean stopped;

  MyThread(String name) {
    thrd = new Thread(this, name);
    suspended = false;
    stopped = false;
    thrd.start();
  }

  public void run() {
    try {
      for (int i = 1; i < 10; i++) {
        System.out.print(".");
        Thread.sleep(50);
        synchronized (this) {
          while (suspended)
            wait();
          if (stopped)
            break;
        }
      }
    } catch (InterruptedException exc) {
      System.out.println(thrd.getName() + " interrupted.");
    }
    System.out.println("\n" + thrd.getName() + " exiting.");
  }
  synchronized void stop() {
    stopped = true;
    suspended = false;
    notify();
  }

  synchronized void suspend() {
    suspended = true;
  }

  synchronized void resume() {
    suspended = false;
    notify();
  }
}

public class Maincode
 {
  public static void main(String args[]) throws Exception {
    MyThread mt = new MyThread("MyThread");
    Thread.sleep(100);
    mt.suspend();
    Thread.sleep(100);

    mt.resume();
    Thread.sleep(100);

    mt.suspend();
    Thread.sleep(100);

    mt.resume();
    Thread.sleep(100);

    mt.stop();
  }
}
```

Output:

```
G:\>javac Maincode.java

G:\>java Maincode
.........
MyThread exiting.
```

9.9 Multithreading

Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. A multithreading is a specialized form of multitasking. Multithreading requires less overhead than multitasking processing.

It is needed to define another term related to threads: process: A process consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process. A process remains running until all of the non-daemon threads are done executing. Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.

The key to utilizing multithreading support effectively is to think concurrently rather than serially. For example, when you have two subsystems within a program that can execute concurrently, make them individual threads. With the careful use of multithreading, you can create very efficient programs. A word of caution is in order, however: If you create too many threads, you can actually degrade the performance of your program rather than enhance it.

9.10 Summary

In this unit, you have learned about Multithreaded Programming and importance of threads in java. Thread model, main thread, how to create a thread, multiple thread is discussed in this unit. How synchronization, priorities and important topic such as inter thread communication, multithreading are discussed. How suspend resume and stopping in thread is also discussed in this unit.

9.11 Key words

- Wait ( ): This method tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).
- Notify ( ): This method wakes up the first thread that called wait( ) on the same object.
- NotifyAll ( ): This method wakes up all the threads that called wait( ) on the same object.

9.12 Unit end exercise and answers

1. What is Thread model?
2. How Main thread works and how to Creating a thread?
3. Explain about Multiple thread?
4. Explain about Priorities and Synchronization in java?
5. How Interthread communication is done? Explain.
6. Explain Suspending-Resuming-Stopping threads?
7. What is Multithreading?

Answers: see

1. 9.2
2. 9.3, 9.4
3. 9.5
4. 9.6, 9.7

5. 9.8
6. 9.9
7. 9.10

9.13 Reference

Teach yourself Java - Joseph O’Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 20010. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 10: The Applet class

Structure:

10.0 OBJECTIVES

At the end of this unit you will be able to know:

- Explain the applet and its architecture
- Differentiate between java application and applet
- List out the various methods used in the Applet class

10.1 INTRODUCTION

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. A Java applet is a small application written in Java and delivered to users in the form of byte code. The user launches the Java applet from a web page and it is then executed within a Java Virtual Machine (JVM) in a process separate from the web browser itself. A Java applet can appear in a frame of the web page, a new application window, Sun's Applet Viewer or a stand-alone tool for testing applets. Java applets were introduced in the first version of the Java language in 1995.

Java applets run at very fast speeds comparable to, but generally slower than, other compiled languages such as C++. Java applets had been many times faster than JavaScript. Unlike JavaScript, Java applets have access to 3D hardware acceleration, making them well suited for non-trivial, computation intensive visualizations. As browsers have gained support for hardware accelerated graphics thanks to the canvas, as well as just in time compiled JavaScript, the speed difference has become less noticeable.

10.2 Applet Basics

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following:

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.

Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

10.3 Architecture

An applet is a window-based program. Applets are event driven. An applet waits until an event occurs. The run-time system notifies the applet about an event by calling an event handler that has been provided by the applet.

The user initiates interaction with the applet later.

10.4 Applet Skeleton

All but the most trivial applets override a set of methods that provides the basic mechanism. Which the browser or applet viewer interfaces to the applet and controls its execution. Four of these methods—init( ), start( ), stop( ), and destroy( )—are defined by Applet. Another, paint( ), is defined by the AWT Component class.
Default implementations for all of these methods are provided. Applets do not need to override those methods they do not use. However, only very simple applets will not need to define all of them. These five methods can be assembled into the skeleton shown here:

```
// An Applet skeleton.
import java.awt.*;
import java.applet.*;
/*
<applet code=”AppletSkel” width=300 height=100>
</applet>
*/
public class AppletSkel extends Applet {
// Called first.
public void init() {
// initialization
}
/* Called second, after init(). Also called whenever
the applet is restarted. */
public void start() {
// start or resume execution
}
// Called when the applet is stopped.
public void stop() {
// suspends execution
}
/* Called when applet is terminated. This is the last
method executed. */
public void destroy() {
// perform shutdown activities
}
// Called when an applet's window must be restored.
public void paint(Graphics g) {
// redisplay contents of window
}
}
```

10.5 Simple Applet display methods

Java has a class applet display methods and few of them are listed below:

Drawstring ():

Applet uses drawString(a member of Graphics class) to display a string. It is called from within either update () or paint ().

Syntax:-void drawstring (String message,int x,int y);

Here, message is the string to be displayed x and y are the co-ordinates of the location where we have to display.

Paint ():

Paint () method is called each time your applet's must be redrawn. It can occur for several reasons:-

The window in which the applet is running may be overwritten by another window and then uncovered and applet window may be minimized then restored, when applet begins execution.

The paint () method has one parameter of type Graphics. This contains the graphics context, which describes the graphic environment.

Update ():

Sometimes applet may need to override another method defined by the AWT-update ().

- This method is called when applet has requested that, a portion of window needs to be redrawn.
- Default version of update () first fills an applet with the default background color and then calls paint.
- If you fill the background using a different color in paint, user will experience a flash of the default background each time window is repainted (whenever update () is called).
- One way to avoid this problem is to override update so that it performs all necessary activities. Then have paint () simply call update ().

Example

```
public void update(Graphics g)
{
//redisplay your window here
}
public void paint(Graphics g)
{
update(g);
}
```

10.6 Requesting repainting

As a general rule, an applet writes to its window only when its update ( ) or paint ( ) method is called by the AWT. This raises an interesting question: How can the applet itself cause its window to be updated when its information changes. For example, if an applet is displaying a moving banner, what mechanism does the applet use to update the window each time this banner scrolls? Remember, one of the fundamental architectural constraints imposed on an applet is that it must quickly return control to the run-time system. It cannot create a loop inside paint ( ) that repeatedly scrolls the banner, for example. This would prevent control from passing back to the AWT. Given this constraint, it may seem that output to your applet's window will be difficult at best. Fortunately, this is not the case. Whenever your applet needs to update the information displayed in its window, it simply calls repaint ( ).

Repaint ():

We can call repaint() method when you want the applet area to be re drawn.The repaint() method calls the update().

The default action of update() method is to clear the applet area and call the paint().

Repaint versions:-

void repaint() :- causes entire window to be repainted .

void repaint(int left, int top, int width, int height) :-Specifies region that will be repainted. Left and top are co-ordinates of upper left corner, width and height of the region.

Void repaint(long maxDelay) :-specifies maximum number of milliseconds that can elapse before update is called.

Void paint(long maxDelay, int left, int top, int width, int height)

10.7 Using the Status Window

In addition to displaying information in its window, an applet can also output a message to the status window of the browser or applet viewer on which it is running. To do so, call showStatus ( ) with the string that you want displayed. The status window is a good place to give the user feedback about what is occurring in the applet, suggest options, or possibly report some types of errors. The status window also makes an excellent debugging aid, because it gives you an easy way to output information about your applet.

The following applet demonstrates showStatus( ):

```
// Using the Status Window.
import java.awt.*;
import java.applet.*;
/*

<applet code="StatusWindow" width=300 height=50>
</applet>
*/

public class StatusWindow extends Applet {
  public void init() {
    setBackground(Color.cyan);
  }

  // Display msg in applet window.
  public void paint(Graphics g) {
    g.drawString("This is in the applet window.", 10, 20);
    showStatus("This is shown in the status window.");
  }
}
```

10.8 The HTML APPLET tag

An applet is a Java program that can be included a web page by using HTML tags. The applet tag is the simpler but older method, and has been superseded by the object tag.

Applet - <applet> </applet>

Java applet can be added by specifying the attributes of the applet tag.

archive="url" - Address or filename of the Java archive file (.jar) containing the class files.

code="?" - Java class to run, for example MyApplet.class

width="?" - The width of the applet, in pixels.

height="?" - The height of the applet, in pixels.

Object - <object> </object>

By using these attributes, object can be tag to include an applet in html:

archive="url" - Address or filename of the Java archive file (.jar) containing the class files.
classid="?" - Java class to run, eg. java:MyApplet.class
codetype="application/java" - The type of object, use application/java.
width="?" - The width of the object, in pixels.
height="?" - The height of the object, in pixels.

Using both applet and object to show an applet

```
<html><body>
 <p>
  <applet code="Logo.class" archive="Logo.jar"
   width="740" height="400"></applet>
 </p>
 <p>
  <object codetype="application/java" classid="java:Logo.class"
   archive="Logo.jar" width="740" height="400"></object>
 </p>
</body></html>
```

10.9 Passing parameters to Applets

It is often useful to be able to pass parameters from a HTML page to an applet. This allows us to modify the action of an applet without actually having to re-code or re-compile the applet.
To pass parameters to an applet, you use the param tag inside the applet tag in your HTML file.
The param tag can have two attributes:
▪name. Represents the parameter name.
▪value. Represents the parameter value.
For example:

```
<param name="customer" value="Jane Goddall"/>
```

The getParameter method can be used to retrieve the parameter value. The parameter name is passed in double quotes as shown above. If parameter has no value, null will be returned
From inside an applet, you can retrieve a parameter by using the Applet class's getParameter method:

```
public java.lang.String getParameter(java.lang.String paramName)
```

Printing the value:

Then in the function paint (Graphics g), we prints the parameter value to test the value passed from html page. Applet will display "Hello! Java Applet" if no parameter is passed to the applet else it will display the value passed as parameter. In our case applet should display "Welcome in Passing parameter in java applet example." message.

Here is the code for the Java Program :

```
import java.applet.*;
import java.awt.*;

public class appletParameter extends Applet {
  private String strDefault = "Hello! Java Applet.";
  public void paint(Graphics g) {
  String strParameter = this.getParameter("Message");
  if (strParameter == null)
  strParameter = strDefault;
  g.drawString(strParameter, 50, 25);
  }
}
```

Here is the code for the html program :

```
<HTML>
<HEAD>
<TITLE>Passing Parameter in Java Applet</TITLE>
</HEAD>
<BODY>
This is the applet:<P>
<APPLET code="appletParameter.class" width="800" height="100">
```

```
<PARAM name="message" value="Welcome in Passing parameter in java applet example.">
</APPLET>
</BODY>
</HTML>
```

There is the advantage that if need to change the output then you will have to change only the value of the param tag in html file not in java code.

Compile the program :

```
javac appletParameter.java
```

3Output after running the program :

To run the program using appletviewer, go to command prompt and type appletviewer appletParameter.html Appletviewer will run the applet for you and and it should show output like Welcome in Passing parameter in java applet example.

10.10 getDocumentbase() and getCodebase()

getDocumentBase()

These two methods are important in that they return the complete URL for the applets.

The signature of the method **getDocumentBase()** defined by Applet class is:

```
public URL getDocumentBase()
```

The method getDocumentBase() gets the URL of the document in which this applet is embedded. For example, suppose an applet is contained within the document:

http://java.sun.com.products/jdk/9.5/index.html

The document base is:

http://java.sun.com.products/jdk/9.5/index.html

getCodeBase ()

The signature of the getCodeBase method is

```
public URL getCodeBase()
```

The method gets the base URL. This is the URL of the directory which contains this applet.

GetDocumentBaseExample shows how get the applet's document URL or gets the base URL of the directory using getDocumentBase() and getCodeBase() method of Java Applet class. In the Following GetDocumentBaseExample using getDocumentBase(),getCodeBase() method.

```
/*
  <applet code="GetDocumentBaseExample" width=350 height=250>
  </applet>
  */

 import java.applet.Applet;
  import java.awt.Graphics;
  import java.net.URL;
  import java.awt.*;
  import java.awt.event.*;

  public class GetDocumentBaseExample extends Applet{

        public void paint(Graphics g){
        String message;

     //getCodeBase() method gets the base URL of the directory in which contains this applet.
      URL appletCodeDir=getCodeBase();
       message = "Code Base : "+appletCodeDir.toString();
       g.drawString(message,10,90);

      // getDocumentBase() Returns an absolute URL of the Document
       URL appletDocDir = getDocumentBase();
       message="Document Base : "+appletDocDir.toString();
       g.drawString(message,10,120);
       g.drawString("http://ecomputernotes.com", 200, 250);
```

```
        }
    }
```

getDocumentBase() Returns an absolute URL naming the directory of the document in which the applet is embedded. For example, suppose an applet is contained within the document:
http://java.sun.com/products/jdk/9.2/index.html

The document base is:
http://java.sun.com/products/jdk/9.2/

getCodeBase()
Gets the base URL. This is the URL of the applet itself.

10.11 AppletContext and showDocument()

AppletContext is an interface that is defined in the java.applet package. It is used by an applet to obtain information from the applet's environment through its methods. Applet runs within a context that is usually provided by a web browser or an applet viewer. Using that context, an applet is able to load bitmap images and audio clips.

Applets can also obtain additional services from an AppletContexts object, which is returned from getAppletContext() method shown as:

```
AppletContext Context = getAppletContext();
```

To get the AppletContext of the applet getAppletContext() method can be used shown as:
Applet AppletInstance = Context.getApplet(AppletName);

The context of the currently executing applet is obtained by a call to the getAppletContext( ) method defined by Applet, one such method is showDocument()

One application of Java is to use active images and animation to provide a graphical means of navigating the Web that is more interesting than simple text-based links. To allow your applet to transfer control to another URL, you must use the showDocument( ) method defined by the AppletContext interface. AppletContext is an interface that lets you get information from the

applet’s execution environment. The context of the currently executing applet is obtained by a call to the getAppletContext( ) method defined by Applet.
Within an applet, once you have obtained the applet’s context, you can bring another document into view by calling showDocument( ). This method has no return value and throws no exception if it fails, so use it carefully. There are two showDocument( ) methods. The method showDocument(URL) displays the document at the specified URL. The method showDocument(URL, String) displays the specified document at the specified location within the browser window. Valid arguments for where are “_self” (show in current frame), “_parent” (show in parent frame), “_top” (show in topmost frame), and “_blank” (show in new browser window). You can also specify a name, which causes the document to be shown in a new browser window by that name.
Syntax
showDocument()
 public abstract void showDocument(URL url, String target)
Show a new document in a target window or frame. This may be ignored by the applet context. This method accepts the target strings: _self show in current frame _parent show in parent frame _top show in top-most frame _blank show in new unnamed top-level window show in new top-level window named.

10.12 AudioClip Interface

When a user want to play an audio file, audio file is loaded into memory with getAudioClip(), that’s when AudioClip interface is used to work with it. There are the three methods to define audio clip interface.
Three methods which define the AudioClip interface are described below. The class that implements these methods depends on the run-time environment; the class is probably sun.applet.AppletAudioClip or netscape.applet.AppletAudioClip.

If you play an audio clip anywhere within Applet, you should call the AudioClip stop() method within the stop() method of the applet. This ensures that the audio file will stop playing when the user leaves web page. Stopping audio clips is a must if you call loop() to play the sound continuously; if you don't stop an audio clip, the user will have to exit the browser to get the sound to stop playing.

Applets can play audio clips simultaneously. Based upon the user's actions, you may want to play a sound file in the background continuously, while playing other files.

void play ()

The play() method plays the audio clip once from the beginning.

void loop ()

The loop() method plays the audio clip continuously. When it gets to the end-of-file marker, it resets itself to the beginning.

void stop ()

The stop() method stops the applet from playing the audio clip.

The applet in the below Example loads three audio files in the init() method. The start() method plays Dino barking in the background as a continuous loop. Whenever the browser calls paint(), Fred yells "Wilma," and when you click the mouse anywhere, the call to mouseDown() plays Fred yelling, "Yabba-Dabba-Doo." If you try real hard, all three can play at once. Before playing any audio clip, the applet makes sure that the clip is not null--that is, that the clip loaded correctly. stop() stops all clips from playing; you should make sure that applets stop all audio clips before the viewer leaves the web page.

```
mport java.net.*;
import java.awt.*;
import java.applet.*;
public class AudioTestExample extends Applet{
   AudioClip audio1, audio2, audio3;
   public void init () {
      audio1 = getAudioClip (getCodeBase(), "audio/flintstones.au");
      audio2 = getAudioClip (getCodeBase(), "audio/dino.au");
      audio3 = getAudioClip (getCodeBase(), "audio/wilma.au");
```

```
    }
    public boolean mouseDown (Event e, int x, int y) {
        if (audio1 != null)
            audio9.play();
        return true;
    }
    public void start () {
        if (audio2 != null)
            audio10.loop();
    }
    public void paint (Graphics g) {
        if (audio3 != null)
            audio11.play();
    }
    public void stop () {
        if (audio1 != null)
            audio9.stop();
        if (audio2 != null)
            audio10.stop();
        if (audio3 != null)
            audio11.stop();
    }
}
```

10.13 AppletStub Interface

The AppletStub inter face provides a way to get information from the run-time browser environment. The Applet class provides methods with similar names that call these methods.

public abstract boolean isActive ()

The isActive() method returns the current state of the applet. While an applet is initializing, it is not active, and calls to isActive() return false. The system marks the applet active just prior to calling start(); after this point, calls to isActive() return true.

public abstract URL getDocumentBase ()

The getDocumentBase() method returns the complete URL of the HTML file that loaded the applet. This method can be used with the getImage() or getAudioClip() methods to load an image or audio file relative to the HTML file.

public abstract URL getCodeBase ()

The getCodeBase() method returns the complete URL of the .class file that contains the applet. This method can be used with the getImage() method or the getAudioClip() method to load an image or audio file relative to the .class file.

public abstract String getParameter (String name)

The getParameter() method allows you to get parameters from <PARAM> tags within the <APPLET> tag of the HTML file that loaded the applet. The name parameter of getParameter() must match the name string of the <PARAM> tag; name is case insensitive. The return value of getParameter() is the value associated with name; it is always a String regardless of the type of data in the tag. If name is not found within the <PARAM> tags of the <APPLET>, getParameter() returns null.

public abstract AppletContext getAppletContext ()

The getAppletContext() method returns the current AppletContext of the applet. This is part of the stub that is set by the system when setStub() is called.

public abstract void appletResize (int width, int height)

The appletResize() method is called by the resize method of the Applet class. The method changes the size of the applet space to width height. The browser must support changing the applet space; if it doesn’t, the size remains unchanged.

10.14 Summary

In this unit, you have learnt the importance of applet in java.Applet is a mobile code. Applets are interpreted by the browser directly. Applets play a crucial role distributed application. Applet Basics, Architecture, Applet skeleton, Simple Applet display methods, Requesting repainting, Using the Status Window; The HTML APPLET tag, Passing parameters to Applets, getDocumentbase() and getCodebase(), AppletContext and showDocument(), AudioClip Interface, AppletStub Interface are the topics discussed in this unit.

10.15 Keywords

Applet – Mobile code

Stop() – Suspends the execution

AppletViewer – used to execute applets

10.16 Unit end exercise and answers

1. What is the basics of an applet?
2. Explain the architecture of Applet?
3. What is Applet skeleton?
4. Write Simple Applet display methods?
5. Explain Requesting repainting in java pplet?
6. Explain how to Use the Status Window
7. Explain about The HTML APPLET tag?
8. How to Pass a parameters to Applets?
9. Explain getDocumentbase (), getCodebase (), AppletContext, showDocument ()?
10. Explain AudioClip Interface, AppletStub Interface in Applet?

Answers are 1. 10.2

2. 10.3

3. 10.4

4. 10.5

5. 10.6

6. 10.7

7. 10.8

8. 10.9

9. 10.10, 10.11,

10. 10.12, 10.13

10.17 Reference

Programming with Java – E. Balagurusamy

The complete Reference java – Herbert Schildt

UNIT 11: Event Handling

Structure:

11.0 OBJECTIVES

At the end of this unit you will be able to know:

- Types of Event
- Delegation event model
- Event classes
- Sources of events
- Event listener interfaces
- Adapter classes
- Inner classes.

11.1 INTRODUCTION

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events

11.2 Types of Event

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of user.They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page etc.

- **Background Events** - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

11.3 Delegation event model

The event model is based on the Event Source and Event Listeners. Event Listener is an object that receives the messages / events. The Event Source is any object which creates the message / event. The Event Delegation model is based on – The Event Classes, The Event Listeners, Event Objects.

There are three participants in event delegation model in Java;

- Event Source – the class which broadcasts the events
- Event Listeners – the classes which receive notifications of events
- Event Object – the class object which describes the event.

An event occurs (like mouse click, key press, etc) which is followed by the event is broadcasted by the event source by invoking an agreed method on all event listeners. The event object is passed as argument to the agreed-upon method. Later the event listeners respond as they fit, like submit a form, displaying a message / alert etc.
Event types are encapsulated in a class hierarchy rooted at java.util.EventObject. An event is propagated from a "Source" object to a "Listener" object by invoking a method on the listener and passing in the instance of the event subclass which defines the event type generated.
A Listener is an object that implements a specific EventListener interface extended from the generic java.util.EventListener. An EventListener interface defines one or more methods which are to be invoked by the event source in response to each specific event type handled by the interface.

An Event Source is an object which originates or "fires" events. The source defines the set of events it emits by providing a set of set<EventType>Listener (for single-cast) and/or add<EventType>Listener (for mult-cast) methods which are used to register specific listeners for those events.

In an AWT program, the event source is typically a GUI component and the listener is commonly an "adapter" object which implements the appropriate listener (or set of listeners) in order for an application to control the flow/handling of events. The listener object could also be another AWT component which implements one or more listener interfaces for the purpose of hooking GUI objects up to each other.
Events are sent to the component from which the event originated, but it is up to each component to propagate the event to one or more registered classes called listener. Listeners contain event

handlers that receive and process the event. In this way, the event handler can be in an object separate from the component. Listeners are classes that implement the Event Listener interface.
Events are objects that are reported only to registered listeners. Every event has corresponding listener interface that mandates which methods must be defined in a class suited to receiving that type of event. The class that implements the interface defines those methods, and can be registered as a listener.
Events from components that have no registered listeners are not propagated.

11.4 Event classes

An instance of the Event class is a platform-independent representation that encapsulates the specifics of an event that happens within the Java 9.0 model. It contains everything you need to know about an event: who, what, when, where, and why the event happened. Note that the Event class is not used in the Java 9.1 event model; instead, Java 9.1 has an AWTEvent class, with subclasses for different event types.
When an event occurs, you decide whether or not to process the event. If you decide against reacting, the event passes through your program quickly without anything happening. If you decide to handle the event, you must deal with it quickly so the system can process the next event. If handling the event requires a lot of work, you should move the event-handling code into its own thread. That way, the system can process the next event while you go off and process the first. If you do not multithread your event processing, the system becomes slow and unresponsive and could lose events. A slow and unresponsive program frustrates users and may convince them to find another solution for their problems.
Variables
Event contains ten instance variables that offer all the specific information for a particular event.
Instance variables
public Object arg: The arg field contains some data regarding the event, to be interpreted by the recipient.
public int clickCount: The clickCount field allows you to check for double clicking of the mouse.
public Event evt: The evt field does not appear to be used anywhere but is available if you wish to pass around a linked list of events.

public int id: The id field of Event contains the identifier of the event.

public int key: For keyboard-related events, the key field contains the integer representation of the keyboard element that caused the event.

pubic int modifiers: The modifiers field shows the state of the modifier keys when the event happened.

public Object target: The target field contains a reference to the object that is the cause of the event.

public long when: The when field contains the time of the event in milliseconds.

Constants

Numerous constants are provided with the Event class. Several designate which event happened. Others are available to help in determining the function key a user pressed (the what). And yet more are available to make your life easier.

When the system generates an event, it calls a handler method for it. To deal with the event, you have to override the appropriate method.

Key constants

These constants are set when a user presses a key. Most of them correspond to function and keypad keys; since such keys are generally used to invoke an action from the program or the system, Java calls them action keys and causes them to generate a different Event type (KEY_ACTION) from regular alphanumeric keys (KEY_PRESS).

Modifiers

Modifiers are keys like Shift, Control, Alt, or Meta. When a user presses any key or mouse button that generates an Event, the modifiers field of the Event instance is set. You can check whether any modifier key was pressed by ANDing its constant with the modifiers field. If multiple modifier keys were down at the time the event occurred, the constants for the different modifiers are ORed together in the field.

Example

public static final int ALT_MASK

public static final int CTRL_MASK

public static final int META_MASK

public static final int SHIFT_MASK

Key events

The component peers deliver separate key events when a user presses and releases
nearly any key. KEY_ACTION and KEY_ACTION_RELEASE are for the function and
arrow keys, while KEY_PRESS and KEY_RELEASE are for the remaining control and
alphanumeric keys.

Window events

Window events happen only for components that are children of Window. Several of these events are available only on certain platforms. Like other event types, the id variable holds the value of the specific event instance.

Mouse events

The component peers deliver mouse events when a user presses or releases a mouse button. Events are also delivered whenever the mouse moves. In order to be platform independent, Java pretends that all mice have a single button. If you press the second or third button, Java generates a regular mouse event but sets the event's modifers field with a flag that indicates which button was pressed. If you press the left button, no modifiers flags are set. Pressing the center button sets the
ALT_MASK flag; pressing the right button sets the META_MASK flag. Therefore, you can determine which mouse button was pressed by looking at the Event Modifiers attribute. Furthermore, users with a one-button or two-button mouse can generate the same events by pressing a mouse button while holding down the Alt or Meta keys.

Scrolling events

The peers deliver scrolling events for the Scrollbar component. The objects that have a built-in scrollbar (like List, ScrollPane, and TextArea) do not generate these events. No default methods are called for any of the scrolling events. They must be dealt with in the handleEvent() method of the Container or a subclass of the Scrollbar. You can determine which particular event occurred by checking the id variable of the event, and find out the new position of the thumb by looking at the arg variable or calling getValue() on the scrollbar.

List events

Two events specific to the List class are passed along by the peers. They signify when the user has selected or deselected a specific choice in the List. It is not ordinarily necessary to capture these events, because the peers deliver the ACTION_EVENT when the user double-clicks on a specific item in the List and it is this ACTION_EVENT that triggers something to happen. However, if there is reason to do something when the user has just single-clicked on a choice, these events may be useful. An example of how they would prove useful is if you are displaying a list of filenames with the ability to preview files before loading. Single selection would preview, double-click would load, and deselect would stop previewing.

Focus events

The peers deliver focus events when a component gains (GOT_FOCUS) or loses (LOST_FOCUS) the input focus. No default methods are called for the focus events.

They must be dealt with in the handleEvent() method of the Container of the component or a subclass of the component. You can determine which particular event occurred by checking the id variable of the event.

FileDialog events

The FileDialog events are another set of nonportable events. Ordinarily, the FileDialog events are completely dealt with by the system, and you never see them. Refer to Chapter 6, Containers for exactly how to work with the FileDialog object. If you decide to create a generic FileDialog object, you can use these events to indicate file loading and saving. These constants would be used in the id variable of the specific event instance:

public static final int LOAD_FILE

public static final int SAVE_FILE

Miscellaneous events

ACTION_EVENT is probably the event you deal with most frequently. It is generated when the user performs the desired action for a specific component type (e.g., when a user selects a button or toggles a checkbox). This constant would be found in the id variable of the specific event instance.

11.5 Sources of events

An event source is a graphical component that is capable of firing off an event. Event sources can be any graphical component (e.g., JButton, JList, JComoboBox, JCheckBox, etc..) the user can interact with. For example, when a JButton is clicked it creates a button click event which can be processed by the Java application.

11.6 Event listener interfaces

Java has 11 event listener interfaces, which specify the methods a class must implement to receive different kinds of events. For example, the ActionListener inter face defines the single method that is called when an ActionEvent occurs. These interfaces replace the various event-handling methods of Java 9.0: action () is now the actionPerformed() method of the ActionListener inter face, mouseUp() is now the mouseReleased() method of the MouseListener inter face, and so on.

ActionListener

The ActionListener inter face contains the one method that is called when an ActionEvent occurs. It has no adapter class. For an object to listen for action events, it is necessary to call the addActionListener() method with the class that implements the ActionListener inter face as the parameter. The method addActionListener() is implemented by Button, List, MenuItem, and TextField components. Other components don't generate action events.

public abstract void actionPerformed(ActionEvent e)

The actionPerformed() method is called when a component is selected or activated. Every component is activated differently; for a List, activation means that the user has double-clicked on an entry. See the appropriate section for a description of each component. actionPerformed() is the Java 9.1 equivalent of the action() method in the 9.0 event model.

The **AdjustmentListener** inter face contains the one method that is called when an AdjustmentEvent occurs. It has no adapter class. For an object to listen for adjustment events, it is necessary to call addAdjustmentListener() with the class that implements the AdjustmentListener inter face as the parameter. The addAdjustmentListener() method is implemented by the Scrollbar component and the Adjustable inter face. Other components don't generate adjustment events.

Public abstract void adjustmentValueChanged(AdjustmentEvent e)

The adjustmentValueChanged() method is called when a slider is moved. The Scrollbar and ScrollPane components have sliders, and generate adjustment events when the sliders are moved. (The TextArea and List components also have sliders, but do not generate adjustment events.) See the appropriate section for a description of each component.

There is no real equivalent to adjustmentValueChanged() in Java 9.0; to work with scrolling events, you had to override the handleEvent() method.

ComponentListener

The ComponentListener inter face contains four methods that are called when a
ComponentEvent occurs; component events are used for general actions on components,
like moving or resizing a component.

public abstract void componentResized(ComponentEvent e)

The componentResized() method is called when a component is resized (for example, by a call to Component.setSize()).

public abstract void componentMoved(ComponentEvent e)

The componentMoved() method is called when a component is moved (for example, by a call to Component.setLocation()).

public abstract void componentShown(ComponentEvent e)

The componentShown() method is called when a component is shown (for example, by a call to Component.show()).

public abstract void componentHidden(ComponentEvent e)

The componentHidden() method is called when a component is hidden (for example, by a call to Component.hide()).

ContainerListener

The ContainerListener inter face contains two methods that are called when a ContainerEvent occurs; container events are generated when components are added to or removed from a container. For a container to listen for container events, it is necessary to call Container.addContainerListener() with the class that implements the interface as the parameter.

public abstract void componentAdded(ContainerEvent e)

The componentAdded() method is called when a component is added to a container (for example, by a call to Container.add()).

public abstract void componentRemoved(ContainerEvent e)

The componentRemoved() method is called when a component is removed from a container (for example, by a call to Container.remove()).

FocusListener

The FocusListener inter face has two methods, which are called when a Focus- Event occurs. For an object to listen for a FocusEven it is necessar y to call the Component. addFocusListener() method with the class that implements the FocusListener inter face as the parameter.

public abstract void focusGained(FocusEvent e)

The focusGained() method is called when a component receives input focus, usually by the user clicking the mouse in the area of the component. This method is the Java 9.1 equivalent of Component.gotFocus() in the Java 9.0 event model.

public abstract void focusLost(FocusEvent e)

The focusLost() method is called when a component loses the input focus. This method is the Java 9.1 equivalent of Component.lostFocus() in the Java 9.0 event model.

ItemListener

The ItemListener inter face contains the one method that is called when an ItemEvent occurs. It has no adapter class. For an object to listen for an ItemEvent, it is necessar y to call addItemListener() with the class that implements the ItemListener inter face as the parameter. The addItemListener() method is implemented by the Checkbox, CheckboxMenuItem, Choice, and List components. Other components don't generate item events.

public abstract void itemStateChanged(ItemEvent e)

The itemStateChanged() method is called when a component's state is modified. Every component is modified differently; for a List, modifying the component means single-clicking on an entry. See the appropriate section for a description of each component.

KeyListener

The KeyListener inter face contains three methods that are called when a KeyEvent occurs; key events are generated when the user presses or releases keys. For an object to listen for key events, it is necessary y to call Component.addKeyListener() with the class that implements the inter face as the parameter.

public abstract void keyPressed(KeyEvent e)

The keyPressed() method is called when a user presses a key. A key press is, literally, just what it says. A key press event is called for every key that is pressed, including keys like Shift and Control. Therefore, a KEY_PRESSED event has a virtual key code identifying the physical key that was pressed; but that's not the same as a typed character, which usually consists of several key presses (for example, Shift+A to type an uppercase A). The keyTyped() method reports actual characters. This method is the Java 9.1 equivalent of Component.keyDown() in the Java 9.0 event model.

public abstract void keyReleased(KeyEvent e)

The keyReleased() method is called when a user releases a key. Like the key- Pressed() method, when dealing with keyReleased(), you must think of virtual key codes, not characters.

This method is the Java 9.1 equivalent of Component.keyUp() in the Java 9.0 event model.

public abstract void keyTyped(KeyEvent e)

The keyTyped() method is called when a user types a key. The method key- Typed() method reports the actual character typed. Action-oriented keys, like function keys, do not trigger this method being called.

MouseListener

The MouseListener inter face contains five methods that are called when a nonmotion oriented MouseEvent occurs; mouse events are generated when the user presses or releases a mouse button. (Separate classes, MouseMotionListener and MouseMotionAdapter, are used to handle mouse motion events; this means that you can listen for mouse clicks only, without being bothered by thousands of mouse motion events.) For an object to listen for mouse events, it is necessary to call the method Window.addWindowListener() with the class that implements the interface as the parameter.

public abstract void mouseEntered(MouseEvent e)

The mouseEntered() method is called when the mouse first enters the bounding area of the component. This method is the Java 9.1 equivalent of Component.mouseEnter() in the Java
9.0 event model.

public abstract void mouseExited(MouseEvent e)

The mouseExited() method is called when the mouse leaves the bounding area of the component. This method is the Java 9.1 equivalent of Component.mouseExit() in the Java 9.0 event model.

public abstract void mousePressed(MouseEvent e)

The mousePressed() method is called each time the user presses a mouse button within the component’s space. This method is the Java 9.1 equivalent of Component.mouseDown() in the Java 9.0 event model.

public abstract void mouseReleased(MouseEvent e)

The mouseReleased() method is called when the user releases the mouse button after a mouse press. The user does not have to be over the original component any more; the original component (i.e., the component in which the mouse was pressed) is the source of the event.
This method is the Java 9.1 equivalent of Component.mouseUp() in the Java 9.0 event model.
public abstract void mouseClicked(MouseEvent e)
The mouseClicked() method is called once each time the user clicks a mouse button; that is, once for each mouse press/mouse release combination.

MouseMotionListener

The MouseMotionListener inter face contains two methods that are called when a motion-oriented MouseEvent occurs; mouse motion events are generated when the user moves the mouse, whether or not a button is pressed. (Separate classes, MouseListener and MouseAdapter, are used to handle mouse clicks and entering/exiting components. This makes it easy to ignore mouse motion events, which are very frequent and can hurt performance. You should listen only for mouse motion events if you specifically need them.)
For an object to listen for mouse motion events, it is necessar y to call Component.addMouseMotionListener() with the class that implements the interface as the parameter.
public abstract void mouseMoved(MouseEvent e)
The mouseMoved() method is called every time the mouse moves within the bounding area of the component, and no mouse button is pressed.
This method is the Java 9.1 equivalent of Component.mouseMove() in the Java 9.0 event model.
public abstract void mouseDragged(MouseEvent e)
The mouseDragged() method is called every time the mouse moves while a mouse button is pressed. The source of the MouseEvent is the component that was under the mouse when it was first pressed.This method is the Java 9.1 equivalent of Component.mouseDrag() in the Java
1.0 event model.

TextListener

The TextListener inter face contains the one method that is called when a Text- Event occurs. It has no adapter class. For an object to listen for a TextEvent, it is necessar y to call addTextListener() with the class that implements the Text- Listener inter face as the parameter.

The addTextListener() method is implemented by the TextComponent class, and thus the TextField and TextArea components. Other components don’t generate text events.

public abstract void textValueChanged(TextEvent e)

The textValueChanged() method is called when a text component’s contents are modified, either by the user (by a keystroke) or programmatically (by the setText() method).

WindowListener

The WindowListener inter face contains seven methods that are called when a WindowEvent occurs; window events are generated when something changes the visibility or status of a window.

11.7 Adapter classes

Most of the listener interfaces have a corresponding adapter class, which is an abstract class that provides a null implementation of all the methods in the interface. (Although an adapter class has no abstract methods, it is declared abstract to remind you that it must be subclassed.) Rather than implementing a listener interface directly, you have the option of extending an adapter class and overriding only the methods you care about. (Much more complex adapters are possible, but the adapters supplied with AWT are very simple.) The adapters are available for the listener interfaces with multiple methods. (If there is only one method in the listener interface, there is no need for an adapter.)

Component Adapter

The adapter class corresponding to ComponentListener is ComponentAdapter. If you care only about one or two of the methods in ComponentListener, you can subclass the adapter and override only the methods that you are interested in. For an object to listen for component events, it is necessary to call Component.addComponentListener() with the class that implements the interface as the parameter.

ContainerAdapter

The adapter class for ContainerListener is ContainerAdapter. If you care only about one of the two methods in Container- Listener, you can subclass the adapter and override only the method that you are interested in.

FocusAdapter

Its adapter class is FocusAdapter. If you care only about one of the methods, you can subclass the adapter and override the method you are interested in.

MouseAdapter

The adapter class for MouseListener is MouseAdapter. If you care about only one or two of the methods in MouseListener, you can subclass the adapter and override only the methods that you are interested in.

MouseMotionAdapter

MouseMotionAdapter is the adapter class for MouseMotionListener. If you care about only one of the methods in MouseMotionListener, you can subclass the adapter and override only the method that you are interested in.

WindowAdapter

The adapter class for WindowListener is WindowAdapter If you care about only one or two of the methods in WindowListener, you can subclass the adapter and override only the methods that you are interested in.

For an object to listen for window events, it is necessary to call the method Window.

addWindowListener() or Dialog.addWindowListener() with the class that implements the interface as the parameter.

public abstract void windowOpened(WindowEvent e)

The windowOpened() method is called when a Window is first opened.

public abstract void windowClosing(WindowEvent e)

The windowClosing() method is triggered whenever the user tries to close the Window.

public abstract void windowClosed(WindowEvent e)

The windowClosed() method is called after the Window has been closed.

public abstract void windowIconified(WindowEvent e)

The windowIconified() method is called whenever a user iconifies a Window.

public abstract void windowDeiconified(WindowEvent e)

The windowDeiconified() method is called when the user deiconifies the Window.

public abstract void windowActivated(WindowEvent e)

The windowActivated() method is called whenever a Window is brought to the front.

public abstract void windowDeactivated(WindowEvent e)

The windowDeactivated() method is called when the Window is sent away from the front, either through iconification, closing, or another window becoming active.

11.8 Inner classes.

Inner classes nest within other classes. A normal class is a direct member of a package, a top-level class. Inner classes, which became available with Java 9.1, come in four flavors:

- Static member classes
- Member classes
- Local classes
- Anonymous classes

a static member class is a static member of a class. Like any other static method, a static member class has access to all static methods of the parent, or top-level, class.

Like a static member class, a member class is also defined as a member of a class. Unlike the static variety, the member class is instance specific and has access to any and all methods and members, even the parent's this reference.

Local classes are declared within a block of code and are visible only within that block, just as any other method variable.
Finally, an anonymous class is a local class that has no name.

11.9 Summary

In this unit, you have learned the importance of Event Handling in java. Type of events , Delegation event model, Event classes, Sources of events, Event listener interfaces, , Adapter classes, Inner classe are the topic discussed in this unit.

11.10 Key words

- Event Source – the class which broadcasts the events
- Event Listeners – the classes which receive notifications of events
- Event Object – the class object which describes the event.

11.11 Unit end exercise and answers

1. Which are the Type of events in java ?
2. Explain Delegation event model in java ?
3. What is Event classes? Explain
4. What is event source in java?
5. Explain about Event listener interfaces
6. What is Adapter classes ? Explain
7. Explain Inner classe?

Answers :see 1. 11.2

2. 11.3

3. 11.4

4. 11.5

5. 11.6

6. 11.7

7. 11.8

11.12 Reference

Teach yourself Java - Joseph O'Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 20010. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 12: Tour of Swing

Structure:

12.0 OBJECTIVES

At the end of this unit you will be able to know:

- JApplet
- JFrame and JComponent
- Icons and Labels
- Handling threading issues
- Text fields

- Buttons
- Combo boxes
- Tabbed panes
- Scroll panes
- Trees

12.1 INTODUCTION

Java language support rich set Graphical User Interface methods, which helps the user to design powerful application. Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

12.2 JApplet

JApplet is a class that enables applets to use Swing components. JApplet is a subclass of java.applet.Applet (in the API reference documentation), which is covered in the Writing Applets (in the Creating a GUI with JFC/Swing trail) trail. An applet is an object that is used by another program, typically a Web browser. The Web browser is the application program and (at least conceptually) holds the main() method. The applet part of a Web page provides services (methods) to the browser when the browser asks for them

An applet object has many instance variables and methods. Most of these are defined in the JApplet class. Because JApplet is a top-level Swing container, each Swing applet has a root pane. The most noticeable effects of the root pane's presence are support for adding a menu bar and the need to use a content pane. To access these definitions, your program should import javax.applet.JApplet and java.awt.*. Here is the code for a small applet. The name of the class is JustOneCircle and the name of the source file is JustOneCircle.java.

Consider this example

```
import javax.swing.JApplet;
import java.awt.*;

// assume that the drawing area is 150 by 150
public class JustOneCircle extends JApplet
{
  final int radius = 25;

  public void paint ( Graphics gr )
  {
    gr.setColor( Color.white );
    gr.fillRect( 0, 0, 150, 150 );
    gr.setColor( Color.black );

    gr.drawOval( (150/2 - radius), (150/2 - radius), radius*2, radius*2 );
   }
}
```

he applet is compiled just like an application using javac JustOneCircle.java. This produces a bytecode file called JustOneCircle.class that can be used in a Web page. The applet is responsible for the white rectangle containing a circle. How it does this will be explained later.

The size of the rectangle is 150 pixels wide by 150 pixels high. This size is encoded both in the source code for the applet (above) and in the HTML that describes the entire Web page.

As described in Using Top-Level Containers, each top-level container such as a JApplet has a single content pane. The content pane makes Swing applets different from regular applets in the following ways:

- You add components to a Swing applet's content pane, not directly to the applet. Adding Components to the Content Pane shows you how.
- You set the layout manager on a Swing applet's content pane, not directly on the applet.
- The default layout manager for a Swing applet's content pane is BorderLayout. This differs from the default layout manager for Applet, which is FlowLayout.

- You should not put painting code directly in a JApplet object. See Performing Custom Painting (in the Creating a GUI with JFC/Swing trail) for examples of how to perform custom painting in applets.

12.3 JFrame and JComponent

The JFrame class is very similar to the AWT Frame, an instance of JFrame is used to build the primary window of applications. A JFrame can also be used to create secondary windows to an application (or an applet). JInternalFrames (see next section) are used to within applications (or applets)

There is one significant difference in using Swing JFrames as opposed to AWT Frames.

The line:

```
 getContentPane().add( topPanel );
```

JFrame exhibits a slight incompatibility when compared to the AWT Frame class because it contains only a single child -- actually an instance of JRootPane. In order to add any other components to the JFrame instance, they must be added to the root pane. To access the root pane, JFrame provides a method called getContentPane().

It is impossible to add components directly to a JFrame using the following syntax:

```
frame.add( component );
```

Instead you must always use this notation:

```
frame.getContentPane().add( component );
```

Failure to add components using the getContentPane() mechanism will result in the generation of an exception. The best solution to this problem is to

- create a panel,
- add it to the content pane,
- Then add all components to the new panel.

The JFrame class offers some other interesting features. In addition to the content pane, JFrame also provides two other panes JLayeredPane and JGlassPane.

The JComponent Class

With the exception of top-level containers, all Swing components whose names begin with "J" descend from the JComponent class. For example, JPanel, JScrollPane, JButton, and JTable all inherit from JComponent. However, JFrame and JDialog don't because they implement top-level containers.

The JComponent class extends the Container class, which itself extends Component. The Component class includes everything from providing layout hints to supporting painting and events. The Container class has support for adding components to the container and laying them out. This section's API tables summarize the most often used methods of Component and Container, as well as of JComponent.

The JComponent class provides the following functionality to its descendants:

- Tool tips
- Painting and borders
- Application-wide pluggable look and feel
- Custom properties
- Support for layout
- Support for accessibility
- Support for drag and drop
- Double buffering
- Key bindings

Tool tips

By specifying a string with the setToolTipText method, you can provide help to users of a component. When the cursor pauses over the component, the specified string is displayed in a small window that appears near the component.

Painting and borders

The setBorder method allows you to specify the border that a component displays around its edges. To paint the inside of a component, override the paintComponent methodApplication-wide **pluggable look and feel**

Behind the scenes, each JComponent object has a corresponding ComponentUI object that performs all the drawing, event handling, size determination, and so on for that JComponent.

Exactly which ComponentUI object is used depends on the current look and feel, which you can set using the UIManager.setLookAndFeel method.

Custom properties

You can associate one or more properties (name/object pairs) with any JComponent. For example, a layout manager might use properties to associate a constraints object with each JComponent it manages. You put and get properties using the putClientProperty and getClientProperty methods. For general information about properties.

Support for layout

Although the Component class provides layout hint methods such as getPreferredSize and getAlignmentX, it doesn't provide any way to set these layout hints, short of creating a subclass and overriding the methods. To give you another way to set layout hints, the JComponent class adds setter methods — setMinimumSize, setMaximumSize, setAlignmentX, and setAlignmentY.

Support for accessibility

The JComponent class provides API and basic functionality to help assistive technologies such as screen readers get information from Swing components, For more information about accessibility.

Support for drag and drop

The JComponent class provides API to set a component's transfer handler, which is the basis for Swing's drag and drop support.

Double buffering

Double buffering smooths on-screen painting. For details

This feature makes components react when the user presses a key on the keyboard. For example, in many look and feels when a button has the focus, typing the Space key is equivalent to a mouse click on the button. The look and feel automatically sets up the bindings between pressing and releasing the Space key and the resulting effects on the button. For more information about key bindings

12.4 Icons and Labels

Many Swing components, such as labels, buttons, and tabbed panes, can be decorated with an icon a fixed-sized picture. An icon is an object that adheres to the Icon interface. Swing provides a particularly useful implementation of the Icon interface: ImageIcon, which paints an icon from a GIF, JPEG, or PNG image.

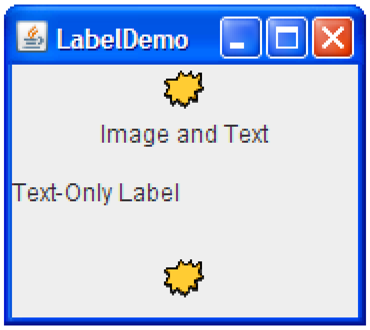


Fig-12.1

Consider Figure 12.1 , an application with three labels, two decorated with an icon:

The program uses one image icon to contain and paint the yellow splats. One statement creates the image icon and two more statements include the image icon on each of the two labels:

```
ImageIcon icon = createImageIcon("images/middle.gif",
                                 "a pretty but meaningless splat");
label1 = new JLabel("Image and Text", icon, JLabel.CENTER);
...
label3 = new JLabel(icon);
```

The createImageIcon method (used in the preceding snippet) is one we use in many of our code samples. It finds the specified file and returns an ImageIcon for that file, or null if that file couldn't be found. Here is a typical implementation:

```
/** Returns an ImageIcon, or null if the path was invalid. */
protected ImageIcon createImageIcon(String path,
                                           String description) {
    java.net.URL imgURL = getClass().getResource(path);
    if (imgURL != null) {
        return new ImageIcon(imgURL, description);
    } else {
        System.err.println("Couldn't find file: " + path);
        return null;
    }
}
```

In the preceding snippet, the first argument to the ImageIcon constructor is relative to the location of the current class, and will be resolved to an absolute URL. The description argument is a string that allows assistive technologies to help a visually impaired user understand what information the icon conveys.

Generally, applications provide their own set of images used as part of the application, as is the case with the images used by many of our demos. You should use the Class getResource method to obtain the path to the image. This allows the application to verify that the image is available and to provide sensible error handling if it is not. When the image is not part of the application, getResource should not be used and the ImageIcon constructor is used directly. For example:

```
ImageIcon icon = new ImageIcon("images/middle.gif", "a pretty but meaningless splat");
```

When you specify a filename or URL to an ImageIcon constructor, processing is blocked until after the image data is completely loaded or the data location has proven to be invalid. If the data location is invalid (but non-null), an ImageIcon is still successfully created; it just has no size and, therefore, paints nothing. As shown in the createImageIcon method, it is advisable to first verify that the URL points to an existing file before passing it to the ImageIcon constructor. This allows graceful error handling when the file isn't present. If you want more information while the image is loading, you can register an observer on an image icon by calling its setImageObserver method

JLabel

JLabel is public class which implements SwingConstants, Accessible and extends JComponent, is used to create text labels.A JLabel object provides text instructions or information on a GUI — display a single line of read-only text, an image or both text and image, it is used when need a user interface component that displays a message or an image.

A display area for a short text string or an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. A label can, however, display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.

A JLabel object can display either text, an image, or both. You can specify where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.You can also specify the position of the text relative to the image. By default, text is on the trailing edge of the image, with the text and image vertically aligned.

A label's leading and trailing edge are determined from the value of its ComponentOrientation property. At present, the default ComponentOrientation setting maps the leading edge to left and the trailing edge to right.

Finally, you can use the setIconTextGap method to specify how many pixels should appear between the text and the image. The default is 4 pixels.

12.5 Handling threading issues

Swing event handling code runs on a special thread known as the event dispatch thread. Most code that invokes Swing methods also runs on this thread. This is necessary because most Swing object methods are not "thread safe": invoking them from multiple threads risks thread interference or memory consistency errors. Some Swing component methods are labelled "thread safe" in the API specification; these can be safely invoked from any thread. All other Swing component methods must be invoked from the event dispatch thread. Programs that ignore this rule may function correctly most of the time, but are subject to unpredictable errors that are difficult to reproduce.

It's useful to think of the code running on the event dispatch thread as a series of short tasks. Most tasks are invocations of event-handling methods, such as ActionListener.actionPerformed. Other tasks can be scheduled by application code, using invokeLater or invokeAndWait. Tasks on the event dispatch thread must finish quickly; if they don't, unhandled events back up and the user interface becomes unresponsive.

When a Swing program needs to execute a long-running task, it usually uses one of the worker threads, also known as the background threads. Each task running on a worker thread is represented by an instance of javax.swing.SwingWorker. SwingWorker itself is an abstract class; you must define a subclass in order to create a SwingWorker object; anonymous inner classes are often useful for creating very simple SwingWorker objects.

SwingWorker provides a number of communication and control features:
The SwingWorker subclass can define a method, done, which is automatically invoked on the event dispatch thread when the background task is finished.
SwingWorker implements java.util.concurrent.Future. This interface allows the background task to provide a return value to the other thread. Other methods in this interface allow cancellation of the background task and discovering whether the background task has finished or been cancelled.
The background task can provide intermediate results by invoking SwingWorker.publish, causing SwingWorker.process to be invoked from the event dispatch thread.
The background task can define bound properties. Changes to these properties trigger events, causing event-handling methods to be invoked on the event dispatch thread.

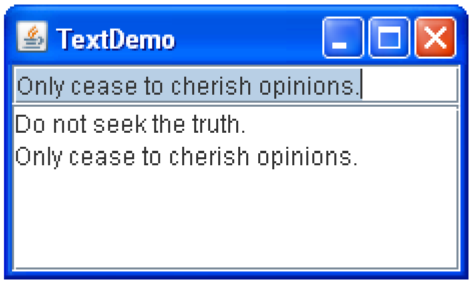
12.6 Text fields

Swing text components display text and optionally allow the user to edit the text. Programs need text components for tasks ranging from the straightforward (enter a word and press Enter) to the complex (display and edit styled text with embedded images in an Asian language).
A text field is a basic text control that enables the user to type a small amount of text. When the user indicates that text entry is complete (usually by pressing Enter), the text field fires an action event. If you need to obtain more than one line of input from the user, use a text area.

The Swing API provides several classes for components that are either varieties of text fields or that include text fields.

JTextField: What this section covers: basic text fields.
JFormattedTextField: A JTextField subclass that allows you to specify the legal set of characters that the user can enter.
JPasswordField: A JTextField subclass that does not show the characters that the user types.
JComboBox: Can be edited, and provides a menu of strings to choose from. See How to Use Combo Boxes.
JSpinner: Combines a formatted text field with a couple of small buttons that enables the user to choose the previous or next available value. See How to Use Spinners.

The following figure 12.2 displays a basic text field and a text area. The text field is editable. The text area is not editable. When the user presses Enter in the text field, the program copies the text field's contents to the text area, and then selects all the text in the text field.



Only cease to cherish opinions.

Do not seek the truth.
Only cease to cherish opinions.

Fig-12.2

The following code creates and sets up the text field:

textField = new JTextField(20);

The integer argument passed to the JTextField constructor, 20 in the example, indicates the number of columns in the field. This number is used along with metrics provided by the field's current font to calculate the field's preferred width. It does not limit the number of characters the user can enter. To do that, you can either use a formatted text field or a document listener

12.7 Buttons

To create a button, you can instantiate one of the many classes that descend from the AbstractButton class.

The Abstract Button class has following classes

Class

JButton : This a common button. It is used where Common Button API and JButton Features

JCheckBox : This is a check box button, it is used where Check Boxes are used.

JRadioButton : One of a group of radio buttons.

JMenuItem : An item in a menu.

JCheckBoxMenuItem : A menu item that has a check box.

JRadioButtonMenuItem : A menu item that has a radio button.

JToggleButton : Implements toggle functionality inherited by JCheckBox and JRadioButton. Can be instantiated or subclassed to create two-state buttons.

12.8 Combo boxes

A JComboBox, which lets the user choose one of several choices, can have two very different forms. The default form is the uneditable combo box, which features a button and a drop-down list of values. The second form, called the editable combo box, features a text field with a small button abutting it. The user can type a value in the text field or click the button to display a drop-down list. Here's what the two forms of combo boxes look like in the Java look and feel.

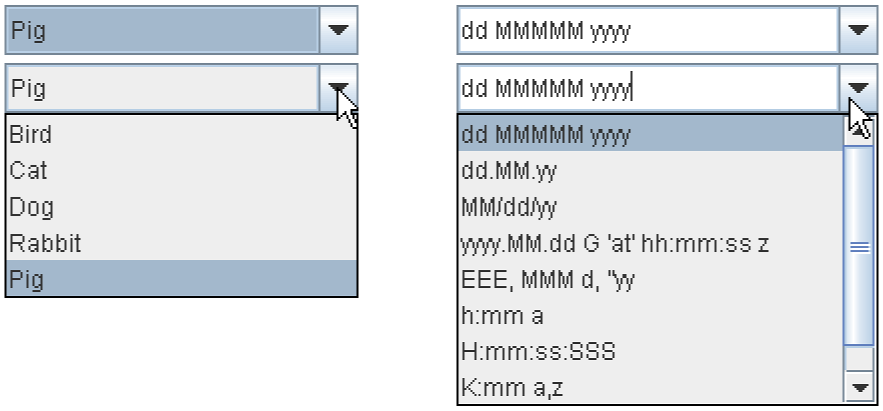


Fig-12.3

Uneditable combo box, before (top) and after the button is clicked

Editable combo box, before and after the arrow button is clicked

Combo boxes require little screen space, and their editable (text field) form is useful for letting the user quickly choose a value without limiting the user to the displayed values. Other components that can display one-of-many choices are groups of radio buttons and lists. Groups of radio buttons are generally the easiest for users to understand, but combo boxes can be more appropriate when space is limited or more than a few choices are available. Lists are not terribly attractive, but they're more appropriate than combo boxes when the number of items is large (say, over 20) or when selecting multiple items might be valid.

12.9 Tabbed panes

With the JTabbedPane class, you can have several components, such as panels, share the same space. The user chooses which component to view by selecting the tab corresponding to the desired component. If you want similar functionality without the tab interface, you can use a card layout instead of a tabbed pane.

To create a tabbed pane, instantiate JTabbedPane, create the components you wish it to display, and then add the components to the tabbed pane using the addTab method.

The following figure Fig 12.4 introduces an application called TabbedPaneDemo that has a tabbed pane with four tabs.

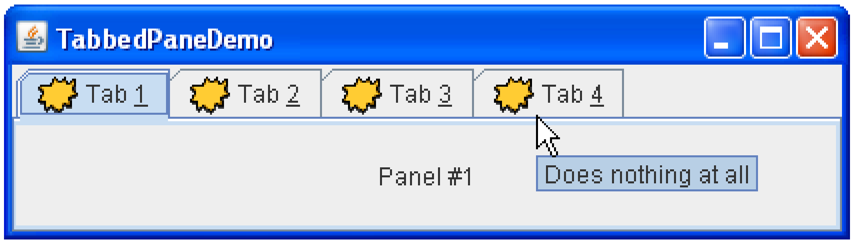


Fig-12.4

As the TabbedPaneDemo example shows, a tab can have a tool tip and a mnemonic, and it can display both text and an image. The default tab placement is set to the `TOP` location, as shown above. You can change the tab placement to `LEFT`, `RIGHT`, `TOP` or `BOTTOM` by using the`setTabPlacement` method.

There are three ways to switch to specific tabs using GUI.

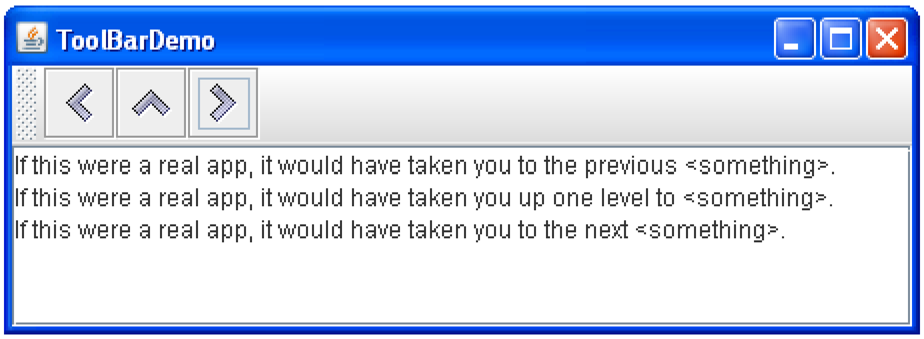
- Using a mouse. To switch to a specific tab, the user clicks it with the mouse.
- Using keyboard arrows. When the JTabbedPane object has the focus, the keyboard arrows can be used to switch from tab to tab.
- Using key mnemonics. The setMnemonicAt method allows the user to switch to a specific tab using the keyboard. For example, setMnemonicAt(3, KeyEvent.VK_4) makes '4' the mnemonic for the fourth tab (which is at index 3, since the indices start with 0); pressing Alt-4 makes the fourth tab's component appear. Often, a mnemonic uses a character in the tab's title that is then automatically underlined.

To switch to a specific tab programmatically, use the setSelectedIndex or the setSelectedComponent methods.

12.10 Scroll panes

A JScrollPane provides a scrollable view of a component. When screen real estate is limited, use a scroll pane to display a component that is large or one whose size can change dynamically. Other containers used to save screen space include split panes and tabbed panes.

The code to create a scroll pane can be minimal. For example, here's a picture of a demo program that puts a text area in a scroll pane because the text area's size grows dynamically as text is appended to it:



```
If this were a real app, it would have taken you to the previous <something>.
If this were a real app, it would have taken you up one level to <something>.
If this were a real app, it would have taken you to the next <something>.
```

Fig-12.5

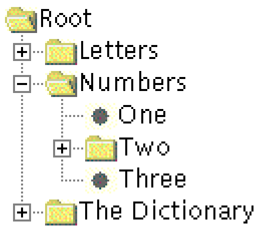
Here's the code that creates the text area, makes it the scroll pane's client, and adds the scroll pane to a container:

```
//In a container that uses a BorderLayout:
textArea = new JTextArea(5, 30);
...
JScrollPane scrollPane = new JScrollPane(textArea);
...
setPreferredSize(new Dimension(450, 110));
...
add(scrollPane, BorderLayout.CENTER);
```

The boldface line of code creates the JScrollPane, specifying the text area as the scroll pane's client. The program doesn't invoke any methods on the JScrollPane object, since the scroll pane handles everything automatically: creating the scroll bars when necessary, redrawing the client when the user moves the scroll knobs, and so on.

12.11 Trees

With the JTree class, you can display hierarchical data. A JTree object does not actually contain your data; it simply provides a view of the data. Like any non-trivial Swing component, the tree gets data by querying its data model.



A tree

Fig-12.6

As the preceding figure 12.6 shows, JTree displays its data vertically. Each row displayed by the tree contains exactly one item of data, which is called a node. Every tree has a root node from which all nodes descend. By default, the tree displays the root node, but you can decree otherwise. A node can either have children or not. We refer to nodes that can have children — whether or not they currently have children — as branch nodes. Nodes that can not have children are leaf nodes.

Branch nodes can have any number of children. Typically, the user can expand and collapse branch nodes — making their children visible or invisible — by clicking them. By default, all branch nodes except the root node start out collapsed. A program can detect changes in branch nodes' expansion state by listening for tree expansion or tree-will-expand events, as described in How to Write a Tree Expansion Listener and How to Write a Tree-Will-Expand Listener.

A specific node in a tree can be identified either by a TreePath, an object that encapsulates a node and all of its ancestors, or by its display row, where each row in the display area displays one node.

An expanded node is a non-leaf node that will display its children when all its ancestors are expanded.
A collapsed node is one which hides them.
A hidden node is one which is under a collapsed ancestor.

12.12 Summary

In this unit, we discussed the need for Abstract Window ToolKit and also the use of AWT. How Swings are different compared to AWT. How to create Frame window have been worked out.

12.13 Keywords

AWT – Abstract Window Toolkit
Swing - are light weight means components are platform – independent.

12.14 Unit End Exercise and Answer

1. What is the need for JApplet?
2. What is JFrame and JComponent?
3. Explain about Icons and Labels?
4. How to Handle threading issues ?
5. What is Text fields? Explain
6. What is Buttons? Explain
7. What is Combo boxes? Explain
8. What is Tabbed panes? Explain
9. What is Tabbed panes? Explain

10. What is Scroll panes? Explain
11. What isTree? Explain

Answers

See 1. 12.2
2. 12.3
3. 12.4
4. 12.5
5. 12.6
6. 12.7
7. 12.8
8. 12.9
9. 12.10
10. 12.11
11. 12.12

12.15 Suggested Reading

Teach yourself Java - Joseph O'Neil

Java Programming – Herb Schildt

Module-4

UNIT 13: The Web Explained and Apache Web Server

Structure:

13.0 OBJECTIVES

After reading this unit you are able to

- Starting-Stopping-Restarting Apache
- Installing and configuring Apache HTTP server.
- Securing Apache
- Create the Web Site
- Apache Log Files

13.1 INTRODUCTION

The Apache project was started simply as a place to collect patches for the NCSA HTTPd. The original "Apache Group" consisted of 8 guys who wanted to add functionality to, and fix problems with, the existing HTTPd code.

• Brian Behlendorf

• Roy T. Fielding

• Rob Hartill

• David Robinson

• Cliff Skolnick

• Randy Terbush

• Robert S. Thau

• Andrew Wilson

The name “Apache” was, apparently, picked by Ben Laurie out of respect for the Apache people. The other form of this story is that Apache is simply another way of saying that it was “A Patchy” server.

13.2 Starting-Stopping-Restarting Apache

Apache provides a variety of ways to start, stop, and restart your server.

apachectl

Most of the time, you want to use apachectl to start and stop your server. apachectl is a handy little script designed to take the grunt work out of starting, stopping, and restarting Apache.

You'll find apachectl in the bin directory of wherever you installed Perl. You might want to copy, or symlink, it into /usr/local/bin, or somewhere else in your path, because you might need it frequently, at least while you're learning about Apache.

apachectl can be run with one of eight different options.

A complete listing of the options can be obtained by running apachectl with the help option:

```
% apachectl help

usage: /usr/local/bin/apachectl (start|stop|restart|
 fullstatus|status|graceful|configtest|help)

start - start httpd
stop - stop httpd
restart - restart httpd if running by sending a SIGHUP or start if
  not running
fullstatus - dump a full status screen; requires lynx and mod_status enabled
status - dump a short status screen; requires lynx and mod_status enabled
graceful - do a graceful restart by sending a SIGUSR1 or start if not running
configtest - do a configuration syntax test
help - this screen
```

These options are, for the most part, self-explanatory.

The start, stop, restart, and graceful options are of particular interest to this section of the book. The first two do exactly what you would expect. restart does what you would expect also, but it's important to know that it also rereads the configuration files, so if any changes have been made, those changes go into effect on a restart. graceful is useful because any connections to the server that are currently active will be completed before the server is restarted. This is important because it means that nobody's connection will be unceremoniously dropped in the middle of receiving a file, therefore, the perceived downtime of the server will be lessened.

For both restart and graceful, if there is a syntax error in the configuration file(s), Apache will not successfully restart. However, Apache is smart enough to check the syntax of the configuration files before shutting down, rather than waiting until it is time to start back up. So, if there is a problem with the configuration files, Apache won't even shut down in the first place. This was not true with earlier versions, where a restart would occasionally leave you with your server down, if you did not check your configuration syntax before attempting the restart.

httpd

httpd is the server binary—that is, this is the actual executable file that is run when you start up your Apache server. You'll find this file in the bin directory of your Apache directory tree. The size of this file will vary depending on what you have built into your server, but it will be around 1MB in size, and will be the largest file in that directory.1

apachectl is just a wrapper that feeds arguments directly to the httpd binary for common options. There are a number of other things that you can do by passing arguments directly to httpd.

Starting and Stopping with httpd

To start the Web server, without any special options, simply invoke httpd directly:

```
/usr/local/apache/bin/httpd
```

To stop the Web server, you'll need to know the PID (process ID) of the Apache parent process. You can acquire this PID from the file httpd.pid, which is located in the directory with your log files—usually /usr/local/apache/logs.

```
cat /usr/local/apache/logs/httpd.pid | xargs kill
```

Command-Line Flags

By passing additional arguments to httpd, you can have it behave in ways slightly different from what you have configured in the server configuration files.

For a complete listing of the available command-line options, invoke httpd with the -h option:

```
# /usr/local/apache/bin/httpd -h
Usage: /usr/local/apache/bin/httpd [-D name] [-d directory] [-f file]
    [-C "directive"] [-c "directive"]
    [-v] [-V] [-h] [-l] [-L] [-S] [-t] [-T]
Options:
 -D name  : define a name for use in <IfDefine name> directives
 -d directory : specify an alternate initial ServerRoot
 -f file  : specify an alternate ServerConfigFile
 -C "directive" : process directive before reading config files
 -c "directive" : process directive after reading config files
 -v  : show version number
 -V  : show compile settings
 -h  : list available command line options (this page)
 -l  : list compiled-in modules
 -L  : list available configuration directives
 -t -D DUMP_VHOSTS: show parsed settings (currently only vhost settings)
 -t  : run syntax check for config files (with docroot check)
 -T  : run syntax check for config files (without docroot check)
```

We'll come back to a number of these options as we talk more about the surrounding information necessary to understand what they do, but of particular interest at this time are the following options:

httpd -v: Shows you what version of the server you are running. This is a good way to convince yourself, after a new install, that you are in fact running the new version, and not some old version that happened to be laying around in your path somewhere.

httpd -V: Tells you what compile settings were in effect when you built Apache.

httpd -l: Lists the compiled-in modules. This will give you some assurance that what you did in the configuration phase actually paid off, and you got the modules that you wanted.

Starting on System Startup

If you are going to run Apache on a production system—that is, if you want people to be able to look at your Web site all the time—then you need to make sure that Apache starts up when you reboot your computer.

This is accomplished a variety of ways on different systems, but they are all typically just a small variation on the same theme.

In /etc/rc.d you will find a collection of scripts, typically with filenames starting with rc. (such as rc.inet1, rc.firewall, and rc.local) which run when your system starts up. The exact layout of this directory varies. Sometimes these scripts are located in subdirectories. Sometimes there are several subdirectories, corresponding to the various runlevels2, containing the scripts, or symlinks to the scripts, and sometimes the scripts are all directly in /etc/rc.d.3

When you know how your system does things, you should create a startup script (called rc.httpd, for example) which contains a call to either apachectl or httpd, as you prefer, telling it to start your server. Alternately, many systems have a script called rc.local, which is specifically for you to put in your customizations to the startup process; this enables you to keep all your startup alterations in the same place.

So, for example, you can add the following line to either your rc.local, or to a separate file called rc.httpd:

```
/usr/local/httpd/bin/apachectl start
```

Or, if you want to start Apache with a configuration file other than the one in the default location, you could use the following line instead:

```
/usr/local/httpd/bin/httpd -f /etc/httpd/host14.conf
```

13.3 Installing and configuring Apache HTTP server.

Run the following command to download Apache HTTP server 14.14.22

```
1 wget http://mirrors.gigenet.com/apache//httpd/httpd-14.14.214.tar.bz2
2 wget http://www.apache.org/dist/httpd/httpd-14.14.214.tar.bz14.md5
3 md5sum -c httpd-14.14.214.tar.bz14.md5
```

After executing above command, httpd-14.14.214.tar.bz2 archive will be downloaded to your “Downloads” folder. To extract the zipped archive file, run the following command from /Downloads folder.

```
1 cd /home/semika/Downloads
```

```
tar -xjvf httpd-14.14.214.tar.bz2
```

Above command will extract the zipped archive file into httpd-14.14.22 folder under Downloads folder. Now, you should decide where you are going to install Apache HTTP server. I am going to install it to /home/semika/httpd-14.14.22 folder. You have to create the folder there. Navigate to your user folder and execute the following command to create new folder.

```
cd /home/semika
mkdir httpd-14.14.22
```

To install Apache to your particular platform, we need to compile the source distribution that we have already downloaded. If you see carefully inside the extracted folder under Downloads/httpd-14.14.22, you can see there is a *configure* script. We can compile the source distribution with that script and it will create necessary stuff to install Apache HTTP server.

When compiling Apache, various options can be specified that are suited to our local environment. For the complete reference of options provided, see here.

Since, we need mod_ssl to be configured with Apache compilation, we need to install OpenSSL development bundle. Otherwise, compilation will raise an exception. To install OpenSSL development libraries, run the following command.

```
sudo apt-get install openssl libssl-dev
```

Sometime, You might need to run the following command as well, if you encounter an error while apache compilation.

```
sudo apt-get install zlib1g-dev
sudo apt-get install libxml2-dev
```

To compile Apache source distribution, execute the following commands.

```
cd /home/semika/Downloads/httpd-14.14.22

```

```
./configure --prefix=/home/semika/httpd-14.14.22 --enable-mods-shared=all --enable-log_config=static--enable-access=static --enable-mime=static --enable-setenvif=static --enable-dir=static -enable-ssl=yes
```

–prefix : You can specify the installation directory
–enable-mods-shared : Setting this to 'all' will enable to install all the shared modules.
–enable-ssl : Since we are going to configure Apache HTTP server with mod_ssl, this has been set to 'yes' to compile Apache with mod_ssl. By default this option is disabled.

For other options specified in the configuration command, please look into full options reference documentation. After successfully running the above command, execute the following commands. Before execute the following command, just have look on your specified installation directory, ie: ***/home/semika/httpd-14.14.22***, you can see that it is still empty.

```
Make
make install
```

Now, you can see Apache HTTP server has been installed under ***/home/semika/httpd-14.14.214.*** Look for modules folder, you can see the list of modules installed. Confirm, whether it has installed **mod_ssl.so**. Now, you can start the Apache HTTP server.

```
cd /home/semika/httpd-14.14.22/bin
sudo ./apachectl start
```

If you see bellow line when executing the above command,

"**httpd: Could not reliably determine the server's fully qualified domain name, using 127.0.13.1 for ServerName**"

edit your ***/httpd-14.14.22/conf/httpd.conf*** file as follows. Look for " **ServerName**" property in **httpd.conf** file. You will see following line there.

```
#ServerName www.example.com:80
```

Uncomment this line and modify it as follows.

```
ServerName localhost
```

Again start the Apache HTTP server. If you did not change the default port which Apache server is running, check the following URL to check whether server is started or not. The default Apache HTTP server running port is 80.

http://localhost/

With the above URL, if you see a page with “It works”, you are done with Apache HTTP server.

Further, if you want to change the default port, you can edit the httpd.conf file as follows. Look for “Listen” property and change it as you wish.

I have set it to 7000. So my Apache HTTP server is running on 7000. I have to access the following URL to get “It works” page.

http://localhost:7000/

To stop Apache HTTP server,execute the following command.

```
cd /home/semika/httpd-14.14.22/bin
sudo ./apachectl stop
```

13.4 Securing Apache

The Apache HTTP Server has a good record for security and a developer community highly concerned about security issues. But it is inevitable that some problems -- small or large -- will be discovered in software after it is released. For this reason, it is crucial to keep aware of updates to the software. If you have obtained your version of the HTTP Server directly from Apache, we highly recommend you subscribe to the Apache HTTP Server Announcements List where you can keep informed of new releases and security updates. Similar services are available from most third-party distributors of Apache software.

Of course, most times that a web server is compromised, it is not because of problems in the HTTP Server code. Rather, it comes from problems in add-on code, CGI scripts, or the underlying Operating System. You must therefore stay aware of problems and updates with all the software on your system.

The Apache HTTP Server is one of the most stable and secure services that ships with Red Hat Enterprise Linux. A large number of options and techniques are available to secure the Apache HTTP Server — too numerous to delve into deeply here. The following section briefly explains good practices when running the Apache HTTP Server.

Always verify that any scripts running on the system work as intended before putting them into production. Also, ensure that only the root user has write permissions to any directory containing scripts or CGIs. To do this, run the following commands as the root user:

```
chown root <directory_name>
chmod 755 <directory_name>
```

System administrators should be careful when using the following configuration options (configured in /etc/httpd/conf/httpd.conf):

FollowSymLinks

This directive is enabled by default, so be sure to use caution when creating symbolic links to the document root of the Web server. For instance, it is a bad idea to provide a symbolic link to /.

Indexes

This directive is enabled by default, but may not be desirable. To prevent visitors from browsing files on the server, remove this directive.

UserDir

The UserDir directive is disabled by default because it can confirm the presence of a user account on the system. To enable user directory browsing on the server, use the following directives:

```
UserDir enabled
UserDir disabled root
```

These directives activate user directory browsing for all user directories other than /root/. To add users to the list of disabled accounts, add a space-delimited list of users on the UserDir disabled line.

ServerTokens

The ServerTokens directive controls the server response header field which is sent back to clients. It includes various information which can be customized using the following parameters:

ServerTokens Full (default option) — provides all available information (OS type and used modules), for example:

Apache/14.0.41 (Unix) PHP/4.14.2 MyMod/13.2

ServerTokens Prod or ServerTokens ProductOnly — provides the following information:

Apache

ServerTokens Major — provides the following information:

Apache/2

ServerTokens Minor — provides the following information:

Apache/14.0

ServerTokens Min or ServerTokens Minimal — provides the following information:

Apache/14.0.41

ServerTokens OS — provides the following information:

Apache/14.0.41 (Unix)

It is recommended to use the ServerTokens Prod option so that a possible attacker does not gain any valuable information about your system.

13.5 Create the Web Site

Apache Web Server is a full-featured, relatively easy to configure, and free server package that is used on millions of computers around the world. The stability of the Apache server has contributed to its popularity in the Internet community. Installing, configuring, and building a page can be a challenge, but it is not outside your reach if you have some technical, it’s very easy to create a simple Apache site on either Windows or Linux. Here's how it works

Linux

The installation of Apache 14.2 on Linux can be handled in approximate 73,624 different ways, give or take a few. That is to say there is a variety of different ways you can install Apache, some dependent on your preferred Linux distribution. For example, if you're a Red Hat or Fedora fan, RPM is your best choice. If you're using some other distribution, you may be able to use RPM, or your distribution may have its own package format.

If you're installing your Linux server from scratch, you can usually choose Apache as an installation option. If you have this option, take it, unless you need something unusual in your installation.

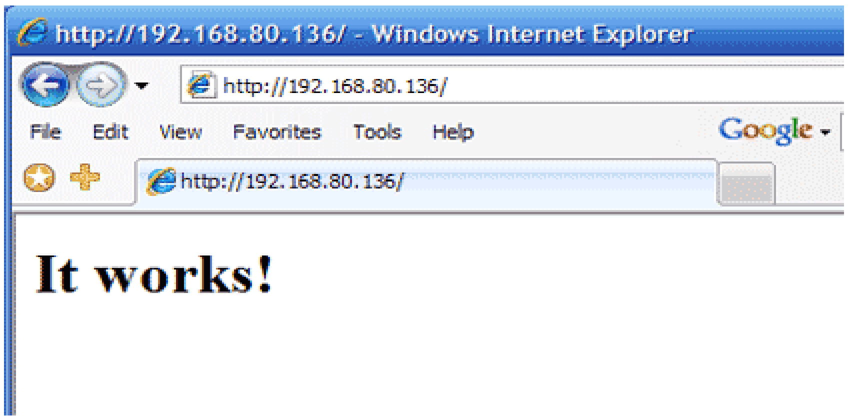
If you're using an existing server and don't want to reinstall the OS, or if you want to have the most granular control over how your Apache installation is configured, your best bet is to build Apache from source code. If you're somewhat new to Linux and the sound of this makes you nervous, it's actually a whole lot easier than it sounds. Better yet, this option works on any Linux distribution out there. It even works for Windows if you have an appropriate compiler installed.

For the example installation in this section, I'm going to build Apache 14.2 from source and install it on a Fedora Core 4 server. You won't see anything fancy in this build--just the basics will be included, but your Linux server will be serving web pages in just minutes!

Before you can compile Apache, you need the source, which is available for download from the Apache web site. As of this writing, the latest version of Apache available is 14.14.0. I've saved the file, named httpd-14.14.0.tar.gz to a folder named /usr/src/apache-14.2 on my server. I like to save installations in this location so I have them for the future.

The next few commands are entered from a command line after that You should get a "It works!" message as shown in Figure 13.13.

Figure 13.1



It works!

Apache was successfully installed.

Before you do too much, you should configure Apache to automatically start when your system boots. The steps to make this happen depend on which Linux distribution you're using. Please refer to your system docs for more information. Until you get that set straight, use the "start" command .

There lists of various modules available for your control during the ./configure portion of the instructions. It indicated whether module is enabled by default or disabled by default, the module name, the configure directive you need to use to enable or disable the module and provided an explanation of the module and linked it to the official Apache documentation page. I gathered this information from the Apache documentation site and reformatted it to be easier to read and follow.

Installing Apache 14.0 on Windows

The Windows installation of Apache is substantially easier than installing Apache from source under Linux. While you can install Apache using source under Windows, how many Windows administrators do you know that actually do this? I don't know any either.

Note that this section is entitled "Apache 14.0 for Windows". The Apache group has yet to release a binary for the Apache 14.2 branch. While others have created such binaries, you're limited in support and, unless you want to build from source yourself, you're at the whim of the person that built the distribution. As such, this section is focused on Apache 14.0.

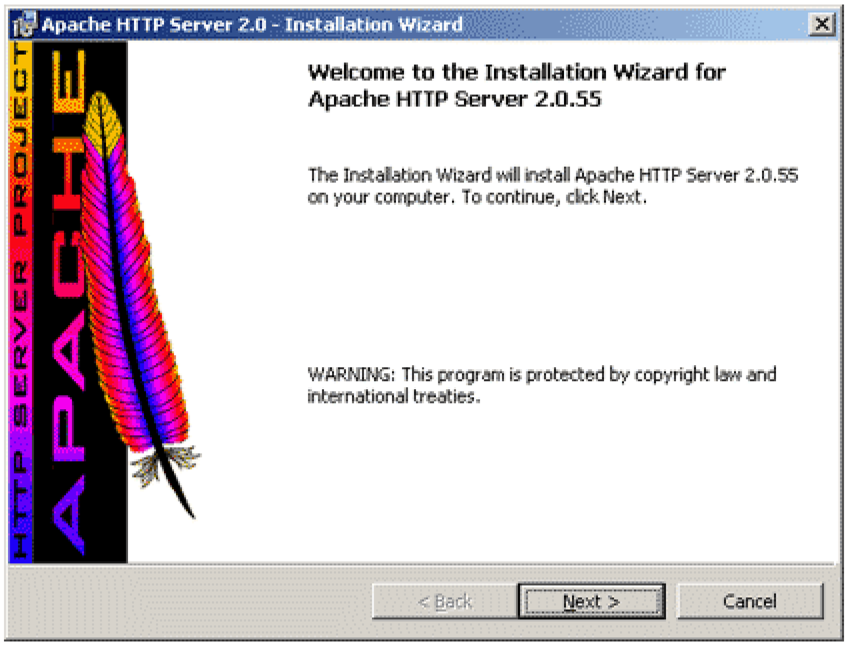
The Apache group provides a win32 executable version of their web server just for the purpose of easily installing under Windows. Before you get started, make sure that you don't have IIS installed on your server. If you continue with IIS installed, Apache will not be able to listen for requests on port 80. You can run Apache and IIS side-by-side if you want to run one of the products on a port other than 80 or if the two products listen to different IP addresses.

After you make sure IIS is removed from your Windows server, download the Apache Windows binary. As of this writing, you won't be able to download a Windows binary version of Apache 14.2 since it's not yet offered on the Apache web site, so download the latest version of Apache 14.0. As of this writing, the latest 14.0 release is 14.0.55. For this example, I've downloaded the file named apache_14.0.55-win32-x86-no_ssl.msi. As you can probably guess, this is a basic Apache build without SSL support, which will suffice perfectly for a simple Apache site.

After download, execute the MSI file and follow the on-screen instructions.

The first screen you get provides you with an introduction. Click the Next button to move on.

Figure 13.2



Welcome to the Installation Wizard for Apache HTTP Server 2.0.55

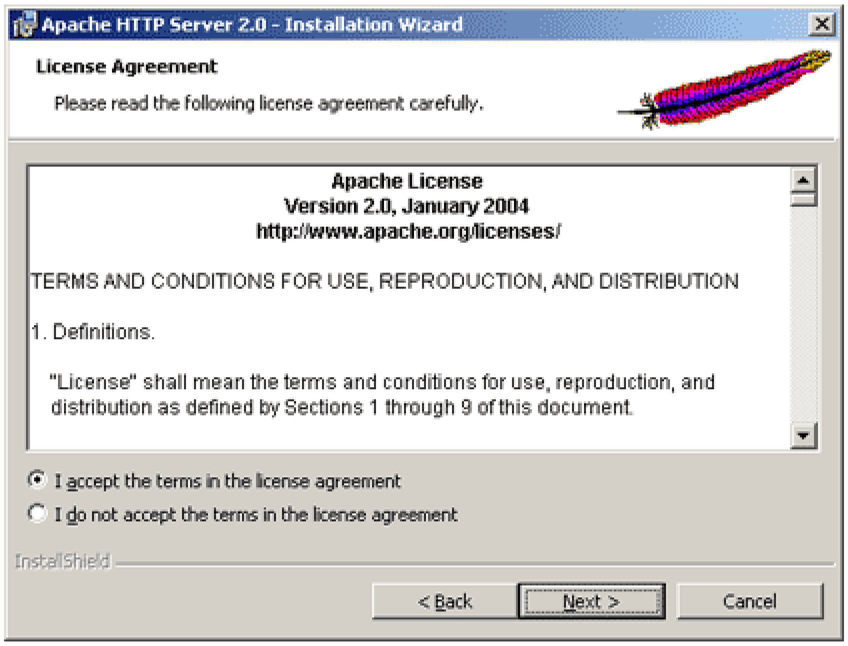
The Installation Wizard will install Apache HTTP Server 2.0.55 on your computer. To continue, click Next.

WARNING: This program is protected by copyright law and international treaties.

The Windows installer introduction screen

And, of course, there is the obligatory license screen. Accept the license and click the Next button to continue with the installation.

Figure 13.3



License Agreement

Please read the following license agreement carefully.

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

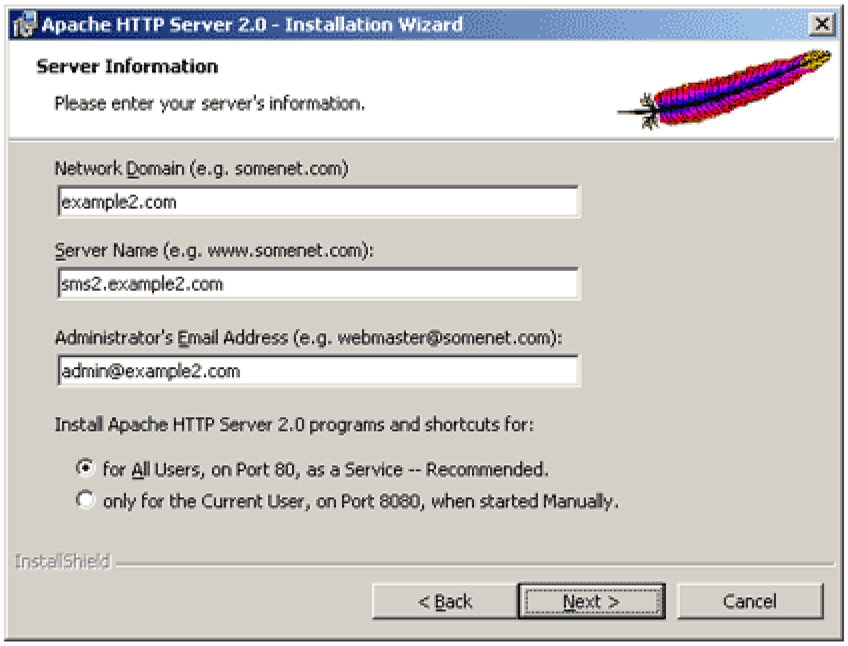
"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

[✓ I accept the terms in the license agreement]
I do not accept the terms in the license agreement

The Apache license

Next, provide the Apache installer with information about your server, including your network domain, the server name, the email address of a server administrator, and decide how you want to run the Apache web server service. Most, if not all, of this information will be filled in. Where possible, stick with the defaults. If you plan to run Apache only for testing, run "only for the Current User".

Figure 13.4


Apache HTTP Server 2.0 - Installation Wizard
Server Information
Please enter your server's information.
InstallShield
< Back
Next >
Cancel

Network Domain (e.g. somenet.com): example2.com

Server Name (e.g. www.somenet.com): sms2.example2.com

Administrator's Email Address (e.g. webmaster@somenet.com): admin@example2.com

Install Apache HTTP Server 2.0 programs and shortcuts for:

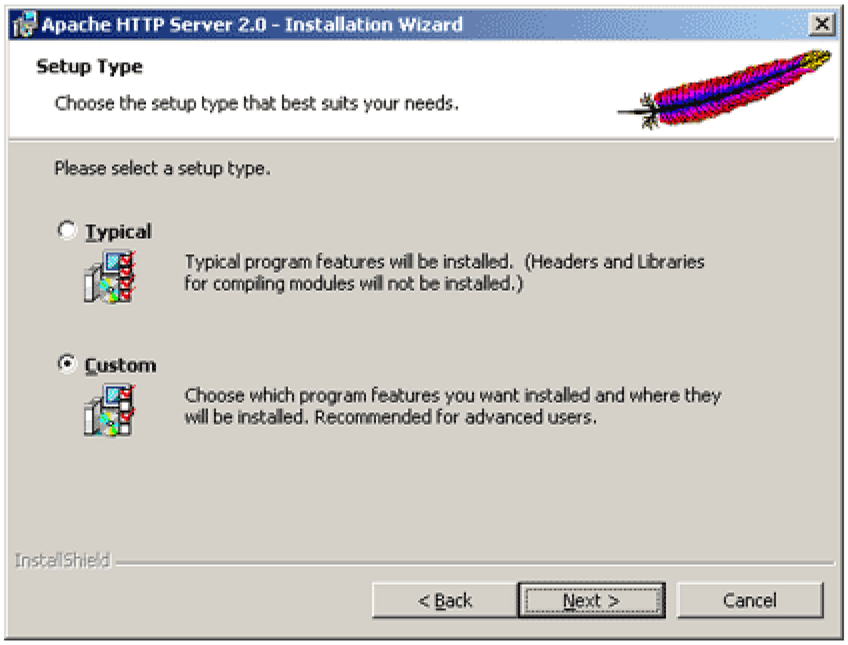
[✓ for All Users, on Port 80, as a Service -- Recommended.]

only for the Current User, on Port 8080, when started Manually.

All of this information is filled in for you.

Now, choose your installation type. While a Typical installation is perfectly fine, I chose to perform a Custom installation so I could demonstrate your installation options.

Figure 13.5


Apache HTTP Server 2.0 - Installation Wizard

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

( ) Typical

Typical program features will be installed. (Headers and Libraries for compiling modules will not be installed.)

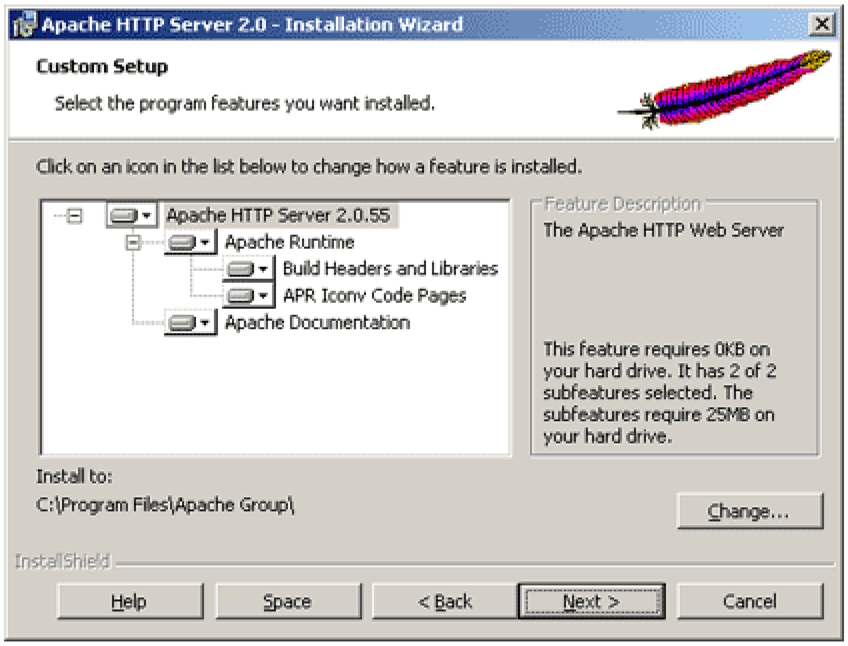
[✓ (•) Custom]

Choose which program features you want installed and where they will be installed. Recommended for advanced users.

Choose your setup type.

By default, everything on the custom screen is selected for installation and the server is installed to C:\Program Files\Apache Group. If you want to change the installation path, click the Change button and choose a new path.

Figure 13.6


Apache HTTP Server 2.0 - Installation Wizard

Custom Setup

Select the program features you want installed.

Click on an icon in the list below to change how a feature is installed.

```
Apache HTTP Server 2.0.55
    Apache Runtime
        Build Headers and Libraries
        APR Iconv Code Pages
    Apache Documentation
```

Feature Description

The Apache HTTP Web Server

This feature requires 0KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 25MB on your hard drive.

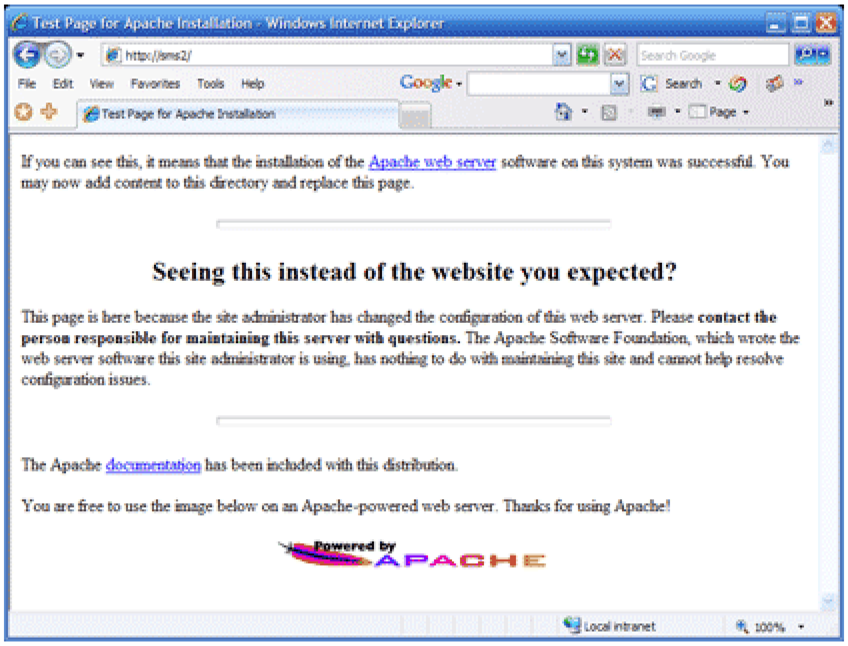
Install to:

C:\Program Files\Apache Group\

Choose your custom installation options.

That's all there is to it! After the installation options screen, the installer does its job and installs Apache using the options you provided. When you're done, open a web browser on another system and point it to your new Apache server. You should get a test page that says "Seeing this instead of the website you expected?"

Figure 13.7


Test Page for Apache Installation - Windows Internet Explorer
http://sms2/
File Edit View Favorites Tools Help
Test Page for Apache Installation
Local intranet
100%

If you can see this, it means that the installation of the Apache web server software on this system was successful. You may now add content to this directory and replace this page.

Seeing this instead of the website you expected?

This page is here because the site administrator has changed the configuration of this web server. Please **contact the person responsible for maintaining this server with questions.** The Apache Software Foundation, which wrote the web server software this site administrator is using, has nothing to do with maintaining this site and cannot help resolve configuration issues.

The Apache documentation has been included with this distribution.

You are free to use the image below on an Apache-powered web server. Thanks for using Apache!

The Apache installation was successful!

Adding and modifying pages

Congratulations on getting your Linux- or Windows-based Apache server up and running! Now, I'll go over how you can add new and edit current pages on your site. After all, you probably don't want to keep the sample pages around for a production site!

As you start adding pages, you'll need a decent editor to make changes to the HTML files. You could use something like FrontPage, but if you want to keep things simple, you can just use a text editor. Under Linux, I usually use pico or nano for this purpose. For Windows, I've fallen in love with the open source Notepad++.

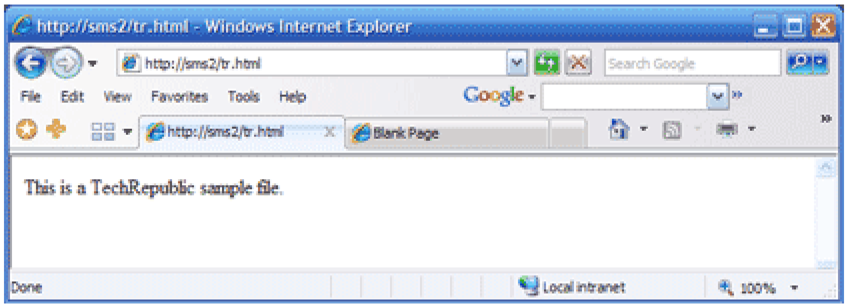
To add or edit pages on your site, you need to know the location of your document root, the folder in which all of your web pages are stored:

- **Windows -** C:\Program Files\Apache Group\Apache2\htdocs
- **Linux -** /usr/local/apache/htdocs

Make sure that you give any files you create an extension of .html so that Apache knows how to handle them. Apache is very dependent on the extension to determine what module should handle a file.

As an example, on my Windows system, I've created a file in the C:\Program Files\Apache Group\Apache2\htdocs folder named tr.html with the contents "This is a TechRepublic sample file." The result is shown in Figure H.

Figure 13.8


http://sms2/tr.html - Windows Internet Explorer
http://sms2/tr.html
File Edit View Favorites Tools Help
Blank Page
Done
Local intranet
100%

This is a TechRepublic sample file.

This file took only seconds to create and add to my site.

Managing Apache

In this article, I won't be going too deeply into managing an Apache configuration file, but will provide you with some general tips. First off, Apache, under both Linux and Windows, is managed through the manipulation of the file named httpd.conf.

- **Windows -** C:\Program Files\Apache Group\Apache2\conf\httpd.conf
- **Linux -** /usr/local/apache/conf/httpd.conf

This is a long text file full of directives that tell Apache what to do. For example, on both servers, the httpd.conf file has a line that reads "Listen 80". This directive tells Apache to listen on port 80 for incoming requests. Going through the various directives is beyond the scope of this article (I will go over the possible options in a future article). However, after you make changes to the file, you need to restart the Apache service. To do this:

- **Windows -** Start | Control Panel | Administrative Tools | Services. Locate the Apache2 service and click the Restart button.
- **Linux -** Execute the command: /usr/local/apache/bin/apachectl restart

It's that simple

For a simple site, this is all you really need to know to get up and running! Once you get used to it, Apache is very easy to work with and provides an outstanding platform for robust web development and page serving. As you get into more advanced techniques, such as scripting languages and database access, you'll come to appreciate Apache's flexibility.

13.6 Apache Log Files

In order to effectively manage a web server, it is necessary to get feedback about the activity and performance of the server as well as any problems that may be occuring. The Apache HTTP Server provides very comprehensive and flexible logging capabilities.
One of the many pieces of the Website puzzle is Web logs. Traffic analysis is central to most Websites, and the key to getting the most out of your traffic analysis revolves around how you configure your Web logs. Apache is one of the most, if not the most powerful open source solutions for Website operations. You will find that Apache's Web logging features are flexible for the single Website or for managing numerous domains requiring Web log analysis. For the single site, Apache is pretty much configured for logging in the default install. The initial httpd.conf file (found in /etc/httpd/conf/httpd.conf in most cases) should have a section on logs that looks similar to this (Apache 14.0.x), with descriptive comments for each item. Your default logs folder will be found in /etc/httpd/logs . This location can be changed when dealing with multiple Websites, as we'll see later. For now, let's review this section of log configuration.

```
ErrorLog logs/error_log

LogLevel warn

LogFormat "%h %l %u %t "%r" %>s %b "%{Referer}i" "%{User-Agent}i"" combined
LogFormat "%h %l %u %t "%r" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

CustomLog logs/access_log combined
```

Error Logs

The error log contains messages sent from Apache for errors encountered during the course of operation. This log is very useful for troubleshooting Apache issues on the server side. Apache

Log Tip: If you are monitoring errors or testing your server, you can use the command line to interactively watch log entries. Open a shell session and type "tail ?f /path/to/error_log" . This will show you the last few entries in the file and also continue to show new entries as they occur. There are no real customization options available, other than telling Apache where to establish the file, and what level of error logging you seek to capture. First, let's look at the error log configuration code from httpd.conf.

```
ErrorLog logs/error_log
```

You may wish to store all error-related information in one error log. If so, the above is fine, even for multiple domains. However, you can specify an error log file for each individual domain you have. This is done in the container with an entry like this:

```
<VirtualHost 10.0.0.2>
DocumentRoot "/home/sites/domain1/html/"
ServerName domain13.com
ErrorLog /home/sites/domain1/logs/error.log
</VirtualHost>
```

If you are responsible for reviewing error log files as a server administrator, it is recommended that you maintain a single error log. If you're hosting for clients, and they are responsible for monitoring the error logs, it's more convenient to specify individual error logs they can access at their own convenience.

Apache's definitions for their error log levels are as follows:

Level	Description
Emerg	Emergencies - system is unusable
Alert	Action must be taken immediately
Crit	Critical Conditions
Error	Error conditions
Warn	Warning Conditions
Notice	Normal but significand condition
Info	Information

Debug Debug-level messages

Tracking Website Activity - Access Logs

Often by default, Apache will generate a log file called access. This tracks the accesses to your Website, the browsers being used to access the site and referring urls that your site visitors have arrived from. It is commonplace now to utilize Apache's "combined" log format, which compiles all three of these logs into one logfile. This is very convenient when using traffic analysis software as a majority of these third-party programs are easiest to configure and schedule when only dealing with one log file per domain. Let's break down the code in the combined log format and see what it all means.

```
LogFormat "%h %l %u %t "%r" %>s %b "%{Referer}i" "%{User-Agent}i"" combined
```

LogFormat starts the line and simply tells Apache you are defining a log file type (or nickname), in this case, combined. Now let's look at the cryptic symbols that make up this log file definition.

Symbol	Description
%h	IP Address of client (remote host)
%l	Identd of client (normally unavailable)
%u	User id of user requesting object
%t	Time of request
%r	Full request string
%>s	Status code
%b	Size of request (excluding headers)
%{Referer}i	The previous webpage
%{User-agent}i	The Client's browser

To review all of the available configuration codes for generating a custom log, see Apache's docs on the module_log_config , which powers log files in Apache.

Apache Log Tip: You could capture more from the HTTP header if you so desired. A full listing and definition of data in the header is found at the World Wide Web Consortium.

For a single Website, the default entry would suffice:

```
CustomLog logs/access_log combined
```

However, for logging multiple sites, you have a few options. The most common is to identify individual log files for each domain. This is seen in the example below, again using the log directive within the container for each domain.

```
<VirtualHost 10.0.0.2>
DocumentRoot "/home/sites/domain1/html/"
ServerName domain13.com
ErrorLog /home/sites/domain1/logs/error.log
CustomLog /home/sites/domain1/logs/web.log
</VirtualHost

<VirtualHost 10.0.0.3>
DocumentRoot "/home/sites/domain2/html/"
ServerName domain14.com
ErrorLog /home/sites/domain2/logs/error.log
CustomLog /home/sites/domain2/logs/web.log
</VirtualHost>

<VirtualHost 10.0.0.4>
DocumentRoot "/home/sites/domain3/html/"
ServerName domain3.com
ErrorLog /home/sites/domain3/logs/error.log
CustomLog /home/sites/domain3/logs/web.log
</VirtualHost>
```

In the above example, we have three domains with three unique Web logs (using the combined format we defined earlier). A traffic analysis package could then be scheduled to process these logs and generate reports for each domain independently.

13.7 Summary

In this unit, you have learned the importance of Apache Web Server in java. How Starting-Stopping-Restarting Apache works, how to Configuration, Security in Apache are discussed in this unit. Also how to create the Web Site and Apache Log Files are also discussed.

13.8 Key words

- prefix : You can specify the installation directory
- Enable-mods-shared: Setting this to 'all' will enable to install all the shared modules.
- Enable-ssl: Since we are going to configure Apache HTTP server with mod_ssl, this has been set to 'yes' to compile Apache with mod_ssl. By default this option is disabled

13.9 Unit end exercise and answers

1. What do u ment by Starting-Stopping-Restarting Apache?
2. How to Configuration Apache?
3. Security in Apache?
4. How to Create the Web Site?
5. Explain about Apache Log Files?

Answers :see 1. 13.2
2. 13.3
3. 13.4
4. 13.5
5. 13.6

13.10 Reference

Teach yourself Java - Joseph O'Neil
Java Programming – Herb Schildt
Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 20014. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 14: Perl and MySQL

Structure:

14.0 OBJECTIVES

At the end of this unit you will be able to know:

- Perl Documentation
- Perl Syntax Rules
- Introduction to MySQL
- Database Independent Interface
- Table Joins
- Loading and Dumping a Database

14.1 INTRODUCTION

The Perl programming language has grown from a scruffy tool exploited by Unix systems administrators for informal scripting to the most widely used development platform for the

World Wide Web. Perl was not designed for the Web or for databases, but its ease of use and powerful text-handling abilities have made it a natural for application development in these areas. A few of the libraries developed for Perl—called modules—have made it even more attractive for web and database applications.

Perl accesses databases through the DataBase Driver/DataBase Interface (DBD/DBI). The name arises from its two-layer implementation. At the bottom is the database driver layer. There, modules exist for each type of database accessible from Perl: MySQL.On top of these database-dependent driver modules lies a database-independent interface layer, which is the interface you use.

14.2 Perl Documentation

Perl comes with a lot of documentation, The most convenient way to access the documentation of core perl is to visit the perldoc website.

It contains an HTML version of the documentation for Perl, the language, and for the modules that come with core Perl as released by the Perl 5 Porters. It does not contain documentation for CPAN modules. There is an overlap though, as there are some modules that are available from CPAN but that are also included in the standard Perl distribution. (These are often referred to as dual-lifed.)

You can use the search box at the top right corner. For example you can type in split and you'll get the documentation of split.

Unfortunately it does not know what to do with while, nor with $_ or @_. In order to get explanation of those you'll have to flip through the documentation.

The most important page might be perlvar, where you can find information about variables such as $_ and @_.

perlsyn explains the syntax of Perl including that of the while loop.

perldoc on the command line

The same documentation comes with the source code of Perl, but not every Linux distribution installs it by default. In some cases there is a separate package. For example in Debian and

Ubuntu it is the perl-doc package. You need to install it using sudo aptitude install perl-doc before you can use perldoc.

Once you have it installed, you can type perldoc perl on the command line and you will get some explanation and a list of the chapters in the Perl documentation. You can quit this using the q key, and then type the name of one of the chapters. For example: perldoc perlsyn.

This works both on Linux and on Windows though the pager on Windows is really weak, so I cannot recommend it. On Linux it is the regular man reader so you should be familiar with it already.

Documentation of CPAN modules

Every module on CPAN comes with documentation and examples. The amount and quality of this documentation varies greatly among the authors, and even a single author can have very well documented and very under-documented modules.

After you installed a module called Module::Name, you can access its documentation by typing perldoc Module::Name.

There is a more convenient way though, that does not even require the module to be installed. There are several web interfaces to CPAN. The main ones are Meta CPAN and search CPAN. They both are based on the same documentation, but they provide a slightly different experience.

Perlpod - the Plain Old Documentation format. Pod is a simple-to-use markup language used for writing documentation for Perl, Perl programs, and Perl modules. Translators are available for converting Pod to various formats like plain text, HTML, man pages, and more. Pod markup consists of three basic kinds of paragraphs: ordinary, verbatim, and command.

Ordinary Paragraph

Most paragraphs in your documentation will be ordinary blocks of text, like this one. You can simply type in your text without any markup whatsoever, and with just a blank line before and after. When it gets formatted, it will undergo minimal formatting, like being rewrapped, probably put into a proportionally spaced font, and maybe even justified.

You can use formatting codes in ordinary paragraphs, for bold, italic, code-style , hyperlinks, and more. Such codes are explained in the "Formatting Codes" section, below.

Verbatim Paragraph

Verbatim paragraphs are usually used for presenting a codeblock or other text which does not require any special parsing or formatting, and which shouldn't be wrapped.

A verbatim paragraph is distinguished by having its first character be a space or a tab. (And commonly, all its lines begin with spaces and/or tabs.) It should be reproduced exactly, with tabs assumed to be on 8-column boundaries. There are no special formatting codes, so you can't italicize or anything like that. A \ means \, and nothing else.

Command Paragraph

A command paragraph is used for special treatment of whole chunks of text, usually as headings or parts of lists.

All command paragraphs (which are typically only one line long) start with "=", followed by an identifier, followed by arbitrary text that the command can use however it pleases. Currently recognized commands are

```
 =pod
=head1 Heading Text
=head2 Heading Text
=head3 Heading Text
=head4 Heading Text
=over indentlevel
=item stuff
=back
=begin format
=end format
=for format text...
=encoding type
=cut
```

14.3 Perl Syntax Rules

A Perl program consists of a sequence of declarations and statements which run from the top to the bottom. Loops, subroutines, and other control structures allow you to jump around within the code.

Perl is a free-form language: you can format and indent it however you like. Whitespace serves mostly to separate tokens, unlike languages like Python where it is an important part of the syntax, or Fortran where it is immaterial.

Many of Perl's syntactic elements are optional. Rather than requiring you to put parentheses around every function call and declare every variable, you can often leave such explicit elements off and Perl will figure out what you meant. This is known as Do What I Mean, abbreviated DWIM. It allows programmers to be lazy and to code in a style with which they are comfortable.

Perl borrows syntax and concepts from many languages: awk, sed, C, Bourne Shell, Smalltalk, Lisp and even English. Other languages have borrowed syntax from Perl, particularly its regular expression extensions. So if you have programmed in another language you will see familiar pieces in Perl. They often work the same, but see perltrap for information about how they differ.

Declarations

The only things you need to declare in Perl are report formats and subroutines (and sometimes not even subroutines). A scalar variable holds the undefined value (undef) until it has been assigned a defined value, which is anything other than undef. When used as a number, undef is treated as 0 ; when used as a string, it is treated as the empty string, "" ; and when used as a reference that isn't being assigned to, it is treated as an error. If you enable warnings, you'll be notified of an uninitialized value whenever you treat undef as a string or a number. Well, usually. Boolean contexts, such as:

```
if ($a) {}
```

are exempt from warnings (because they care about truth rather than definedness). Operators such as ++ , -- , += , -= , and .= , that operate on undefined variables such as:

```
undef $a;
$a++;
```

are also always exempt from such warnings.

A declaration can be put anywhere a statement can, but has no effect on the execution of the primary sequence of statements: declarations all take effect at compile time. All declarations are typically put at the beginning or the end of the script. However, if you're using lexically-scoped private variables created with my(), state(), or our(), you'll have to make sure your format or subroutine definition is within the same block scope as the my if you expect to be able to access those private variables.

Declaring a subroutine allows a subroutine name to be used as if it were a list operator from that point forward in the program. You can declare a subroutine without defining it by saying sub name , thus:

```
sub myname ($);
$me = myname $0          || die "can't get myname";
```

That now parses as you'd expect, but you still ought to get in the habit of using parentheses in that situation. For more on prototypes, see perlsub Subroutines declarations can also be loaded up with the require statement or both loaded and imported into your namespace with a use statement. See perlmod for details on this.

A statement sequence may contain declarations of lexically-scoped variables, but apart from declaring a variable name, the declaration acts like an ordinary statement, and is elaborated within the sequence of statements as if it were an ordinary statement. That means it actually has both compile-time and run-time effects.

Basic Rules and Syntax

The most basic things about perl are...

1) The #!/usr/local/bin/perl line. Put this in the top of the file. Line No 13. What does it do? This tells the system where the perl interpreter is located. Now there is a problem here. Different systems have perl at different locations. So you have to find where the perl executable is located in your system. Sometimes it will be #!/usr/bin/perl. Now if you are using perl in Windows, the script will run without this line. But put it in anyway. Make it a habit.

2) The second thing is the semi-colon or ;. Put it at the end of every statement. For example...
print "Hello World";
If you have any experience with C++, C etc, this will be familiar to you.

3) Include as many comments as possible. One of the most noted disadvantages of perl is its bad readability. So if you want to understand what you wrote today when you read it tomorrow, you'd better put comments wherever you can. Comments can be included with the '#' character.

Rather harsh, don't you think? Sorry about the pun - couldn't resist. This is the example of a comment...

```
#This is a comment
print "Hell World"; # Everything after the '#' is a comment
```

4) Double quotes or single quotes may be used around strings:

```
print "Hello, word";
print 'Helo world';
```

However, only double quotes will substitute variables and special characters such as new lines (\n):

```
print "Hello, $name\n"; # works fine
print 'Hello, $name\n'; # prints $name\n literally
```

5) You can use parentheses(brackets - ')') for functions' arguments or omit them according to your personal taste. They are only required occasionally to clarify issues of precedence.

```
print("Hello, world\n");
print "Hello, world\n";
```

6) Variables are words which has the '$', '@' or '%' symbol at the start. They need not be declared. But if you want a local scope variable use the keyword 'my'. You can learn more about variables later.

14.4 Introduction to MySQL

MySQL is a very popular, open source database. officially pronounced “my Ess Que Ell” (not my sequel). Handles very large databases; very fast performance.

MySQL is used more because it is Free, much cheaper than Oracle, Each student can install MySQL locally, and it is Easy to use Shell for creating tables, querying tables, etc. Also Easy to use with Java JDBC

MySQL provides an interactive shell for creating tables, inserting data, etc.

On igor, just go to the terminal app and type:

```
mysql -h igor -u ma007xyz –p
```

MySQL allows users to create multiple databases, so that a single MySQL server can host databases for many independent applications. Before you start issuing SQL commands to mysql, you first have to select the database that you will be using. In order to see what databases currently exist, run

```
SHOW DATABASES;
```

Information schema is a database that MySQL creates automatically and uses to maintain some internal statistics on datbases and tables. The other two databases, CS144 and TEST, are what we created for the project (note database names are case-sensitive in MySQL). This two database setup mimics common development environments in the real world. The CS144 database is your "production" database, meant for use in the final versions of your code. The TEST database is for any experimentation and for use during development and debugging. Select the TEST database for the rest of this tutorial by issuing the command

```
USE TEST;
```

It is also possible to specify a database as a command line parameter to the mysql command:

```
cs144@cs144:~$ mysql TEST
;l
```

14.5 Database Independent Interface

DBI is a database-independent interface for the Perl programming language. DBD::mysql is the driver for connecting to MySQL database servers with DBI.

DBI is the basic abstraction layer for working with databases in Perl.
DBD::mysql is the driver for using MySQL with DBI.
Net::MySQL is a pure-Perl implementation of the MySQL client-server protocol. (It is not necessary when using DBI with DBD::mysql, but may be useful in environments where you are not able to compile the MySQL client library required by DBD::mysql.)
The following are the main DBI classes. They need to be extended by individual database back-ends (Sybase, Oracle, etc.) Individual drivers need to provide methods for the generic functions listed here (those methods that are optional are so indicated).
The DBI classes are DBIObject, DBIDriver, DBIConnection and DBIResult.

All these are virtual classes. Drivers define new classes that extend these, e.g., PgSQLDriver, PgSQLConnection, and so on.

DBI is a database access Application Programming Interface (API) for the Perl Language. The DBI API Specification defines a set of functions, variables and conventions that provide a consistent database interface independent of the actual database being used.

The DBI defines three main types of objects that you may use to interact with databases. These objects are known as handles. There are handles for drivers, which the DBI uses to create handles for database connections, which, in turn, can be used to create handles for individual database commands, known as statements. Figure 4-3 illustrates the overall structure of the way in which handles are related, and their meanings are described in the following sections.

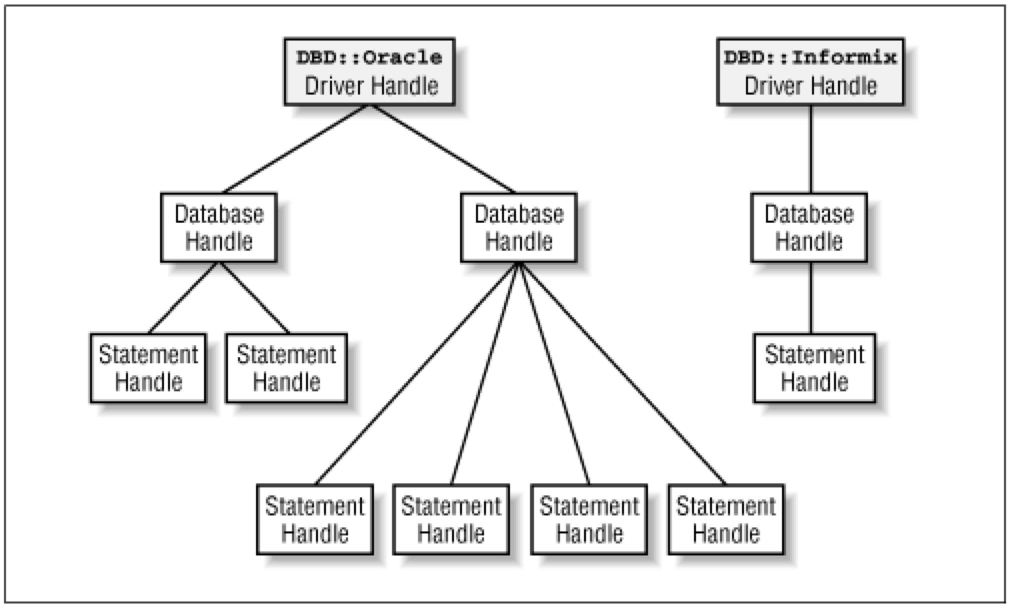


Fig-14.1

Driver Handles

Driver handles represent loaded drivers and are created when the driver is loaded and initialized by the DBI. There is exactly one driver handle per loaded driver. Initially, the driver handle is the only contact the DBI has with the driver, and at this stage, no contact has been made with any database through that driver.

The only two significant methods available through the driver handle are data_sources(), to enumerate what can be connected to, and connect(), to actually make a connection. These

methods are more commonly invoked as DBI class methods, however, which we will discuss in more detail later in this chapter.
Since a driver handle completely encapsulates a driver, there's no reason why multiple drivers can't be simultaneously loaded. This is part of what makes the DBI such a powerful interface.
For example, if a programmer is tasked with the job of transferring data from an Oracle database to an Informix database, it is possible to write a single DBI program that connects simultaneously to both databases and simply passes the data backwards and forwards as needed. In this case, two driver handles would be created, one for Oracle and one for Informix. No problems arise from this situation, since each driver handle is a completely separate Perl object.

Within the DBI specification, a driver handle is usually referred to as $drh.

Driver handles should not normally be referenced within your programs. The actual instantiation of driver handles happens ``under the hood'' of DBI, typically when DBI->connect() is called.

Database Handles
Database handles are the first step towards actually doing work with the database, in that they encapsulate a single connection to a particular database. Prior to executing SQL statements within a database, we must actually connect to the database. This is usually achieved through the DBI's connect() method:

```
$dbh = DBI->connect( $data_source, ... );
```

The majority of databases nowadays tend to operate in a multiuser mode, allowing many simultaneous connections, and database handles are designed accordingly. An example might be if you wanted to write a stock-monitoring program that simultaneously monitored data in tables within different user accounts in the database. A DBI script could make multiple connections to the database, one for each user account, and execute SQL statements on each. Database handles are completely encapsulated objects, meaning that transactions from one database handle cannot ``cross-over'' or ``leak'' into another.

Database handles are children of their corresponding driver handle, which supports the notion that we could also make multiple simultaneous connections to multiple database types, as well as multiple simultaneous connections to databases of the same type. For example, a more complicated DBI script could make two connections to each of an Oracle and an Informix database to perform the above-mentioned monitoring. Figure 4-3, shown earlier, illustrates the capability of having multiple database handles connecting through a driver handle to an Oracle database.

Keep in mind that had the monitoring program been written in C, two copies of code would be required, one for Oracle's programming interface and one for Informix's. DBI levels the playing field.
Within the DBI specification and sample code, database handles are usually referred to as $dbh.

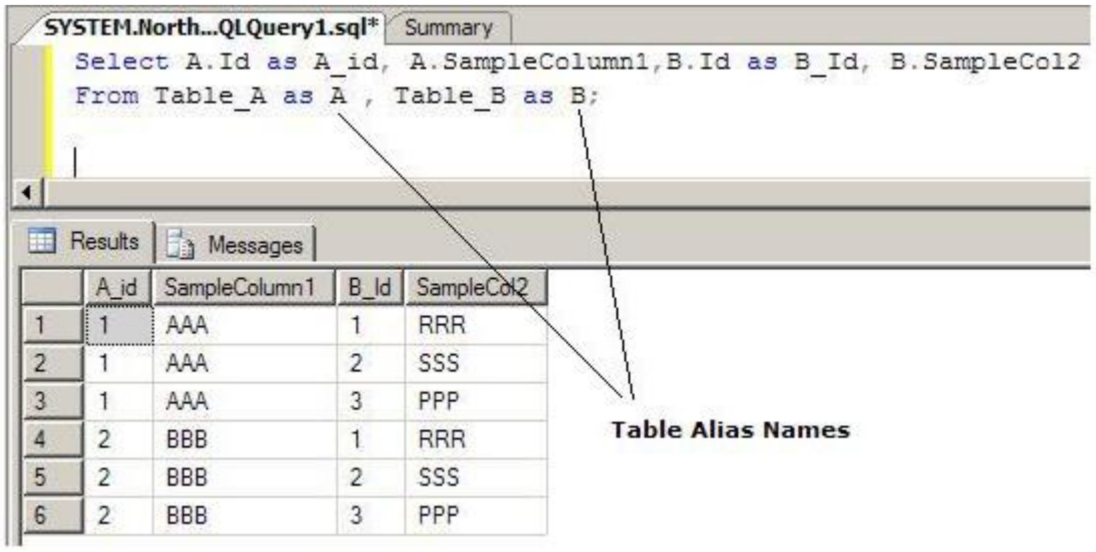
Statement Handles

Statement handles are the final type of object that DBI defines for database interaction and manipulation. These handles actually encapsulate individual SQL statements to be executed within the database.
Statement handles are children of their corresponding database handle. Since statement handles are objects in their own right, data within one statement is protected from tampering or modification by other statement handles.
For a given database handle, there is no practical limit to the number of statement handles that can be created and executed. Multiple statements can be created and executed within one script, and the data can be processed as it returns. A good example of this might be a data-mining robot that connects to a database, then executes a large number of queries that return all sorts of different types of information. Instead of attempting to write convoluted SQL to correlate the information within the database, the Perl script fetches all the data being returned from the many statements and performs analysis there, using the fully featured text and data manipulation routines that Perl has to offer.
Within the DBI specification and sample code, statement handles are generally referred to as $sth.

14.6 Table Joins

Joins are useful for bringing data together from different tables based on their database relations. Usually join will be performed between two tables based on the key columns between two tables those together constitutes the database table relationship. For Example DeptId in the employee table and DeptId in the Department table make the relationship between these two tables.

The below one is the example Joining the tables without using the key columns. Here, TableA and TableB are clubbed together to form the whole result set based on Cartesian Product. The Cartesian product will take a single record in the first table and attaches with all the records in the second table. Then takes the Second records in the first table and attaches with all the records the second table and this continue till the end of the records on the first table.

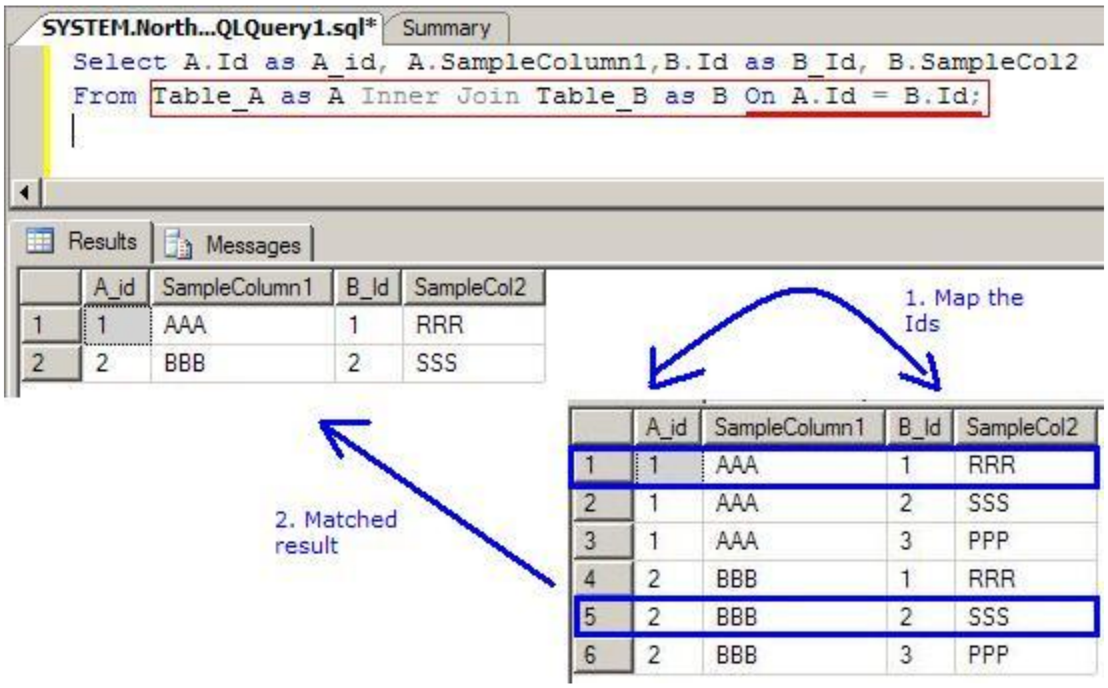


```
Select A.Id as A_id, A.SampleColumn1,B.Id as B_Id, B.SampleCol2
From Table_A as A , Table_B as B;
```

	A_id	SampleColumn1	B_Id	SampleCol2
1	1	AAA	1	RRR
2	1	AAA	2	SSS
3	1	AAA	3	PPP
4	2	BBB	1	RRR
5	2	BBB	2	SSS
6	2	BBB	3	PPP

Joining Two tables

When joining the two tables to avoid the bulk number of records that results as shown in the previous example, we should chose a join column from both the tables. The example given below joins ***Table_A*** and ***Table_B*** based on the Column called ID. Since the column in both the tables mapped based on the ID column, we will reduce huge records that are logically not useful and not related the ID map. Below is the Result of the Join:



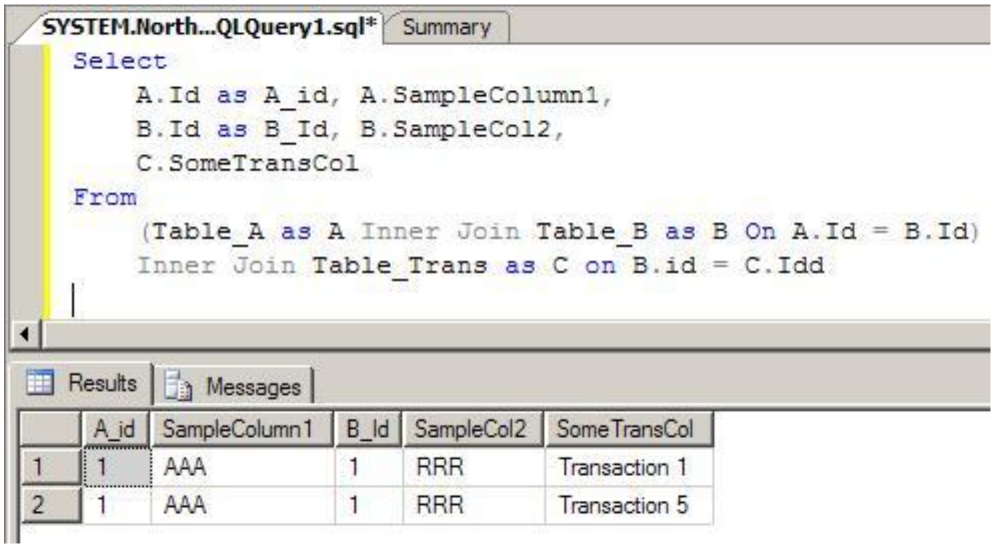
```
Select A.Id as A_id, A.SampleColumn1,B.Id as B_Id, B.SampleCol2
From Table_A as A Inner Join Table_B as B On A.Id = B.Id;
```

	A_id	SampleColumn1	B_Id	SampleCol2
1	1	AAA	1	RRR
2	2	BBB	2	SSS

	A_id	SampleColumn1	B_Id	SampleCol2
1	1	AAA	1	RRR
2	1	AAA	2	SSS
3	1	AAA	3	PPP
4	2	BBB	1	RRR
5	2	BBB	2	SSS
6	2	BBB	3	PPP

Joining multiple tables

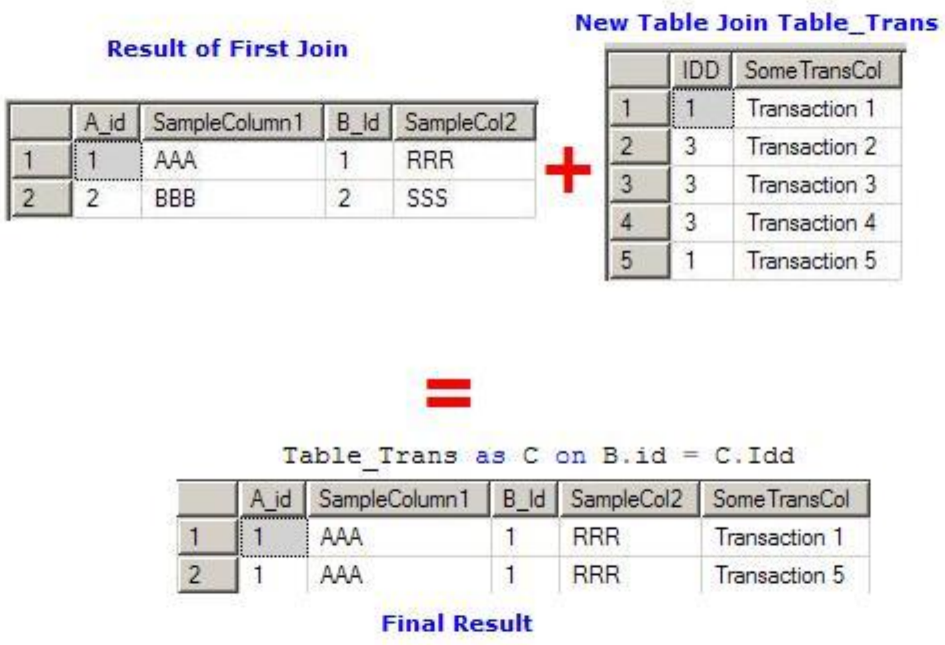
The above example joint two tables. To join multiple tables, we should use the result of the previous join and pick a column from it, then pick a column in the new table then specify the join condition as the previous example. This way we can join multiple numbers of tables. Consider whatever joint so far as the single table and join it with the new one.



```
Select
    A.Id as A_id, A.SampleColumn1,
    B.Id as B_Id, B.SampleCol2,
    C.SomeTransCol
From
    (Table_A as A Inner Join Table_B as B On A.Id = B.Id)
    Inner Join Table_Trans as C on B.id = C.Idd
```

	A_id	SampleColumn1	B_Id	SampleCol2	SomeTransCol
1	1	AAA	1	RRR	Transaction 1
2	1	AAA	1	RRR	Transaction 5

First ***Table_A*** is joins with ***Table_B***, which is nothing but the previous example. Then the joint result of A and B is considered as single table say AB. Then the AB is joint with the ***Table_C***. This is shown in the below picture:


Result of First Join
New Table Join Table_Trans
+
=
Table_Trans as C on B.id = C.Idd
Final Result

	A_id	SampleColumn1	B_Id	SampleCol2
1	1	AAA	1	RRR
2	2	BBB	2	SSS

	IDD	SomeTransCol
1	1	Transaction 1
2	3	Transaction 2
3	3	Transaction 3
4	3	Transaction 4
5	1	Transaction 5

	A_id	SampleColumn1	B_Id	SampleCol2	SomeTransCol
1	1	AAA	1	RRR	Transaction 1
2	1	AAA	1	RRR	Transaction 5

14.7 Loading and Dumping a Database.

We can load a database or otherwise execute SQL commands from a file. We simply put the commands or database into a file—let's call it mystuff.sql—and load it in with this command:

```
$ mysql people < mystuff.sql
```

We can also dump out a database into a file with this command:

```
$ mysqldump people > entiredb.sql
```

For fun, try the mysqldump command with the people database (a gentle reminder: the password is LampIsCool):

```
$ mysqldump -uapache -p people
```

Enter password:

Notice that this outputs all the SQL needed to create the table and insert all the current records. For more information, see man mysqldump.If you're storing anything in MySQL databases that you do not want to lose, it is very important to make regular backups of your data to protect it from loss. This tutorial will show you two easy ways to backup and restore the data in your MySQL database. You can also use this process to move your data to a new web server.

- Back up From the Command Line (using mysqldump)
- Back up your MySQL Database with Compress
- Restoring your MySQL Database
- Backing Up and Restoring using PHPMyAdmin

Back up From the Command Line (using mysqldump)

If you have shell or telnet access to your web server, you can backup your MySQL data by using the mysqldump command. This command connects to the MySQL server and creates an SQL dump file. The dump file contains the SQL statements necessary to re-create the database. Here is the proper syntax:

```
$ mysqldump --opt -u [uname] -p[pass] [dbname] > [backupfile.sql]
```

[uname] Your database username

[pass] The password for your database (note there is no space between -p and the password)

[dbname] The name of your database

[backupfile.sql] The filename for your database backup

[--opt] The mysqldump option

For example, to backup a database named 'Tutorials' with the username 'root' and with no password to a file tut_backup.sql, you should accomplish this command:

```
$ mysqldump -u root -p Tutorials > tut_backup.sql
```

This command will backup the 'Tutorials' database into a file called tut_backup.sql which will contain all the SQL statements needed to re-create the database.

With mysqldump command you can specify certain tables of your database you want to backup. For example, to back up only php_tutorials and asp_tutorials tables from the 'Tutorials' database accomplish the command below. Each table name has to be separated by space.

```
$ mysqldump -u root -p Tutorials php_tutorials asp_tutorials > tut_backup.sql
```

Sometimes it is necessary to back up more that one database at once. In this case you can use the --database option followed by the list of databases you would like to backup. Each database name has to be separated by space.

```
$ mysqldump -u root -p --databases Tutorials Articles Comments > content_backup.sql
```

If you want to back up all the databases in the server at one time you should use the --all-databases option. It tells MySQL to dump all the databases it has in storage.

```
$ mysqldump -u root -p --all-databases > alldb_backup.sql
```

The mysqldump command has also some other useful options:

--add-drop-table: Tells MySQL to add a DROP TABLE statement before each CREATE TABLE in the dump.

--no-data: Dumps only the database structure, not the contents.

--add-locks: Adds the LOCK TABLES and UNLOCK TABLES statements you can see in the dump file.

The mysqldump command has advantages and disadvantages. The advantages of using mysqldump are that it is simple to use and it takes care of table locking issues for you. The disadvantage is that the command locks tables. If the size of your tables is very big mysqldump can lock out users for a long period of time.

Back up your MySQL Database with Compress

If your mysql database is very big, you might want to compress the output of mysqldump. Just use the mysql backup command below and pipe the output to gzip, then you will get the output as gzip file.

```
$ mysqldump -u [uname] -p[pass] [dbname] | gzip -9 > [backupfile.sql.gz]
```

If you want to extract the .gz file, use the command below:

```
$ gunzip [backupfile.sql.gz]
```

Restoring your MySQL Database

Above we backup the Tutorials database into tut_backup.sql file. To re-create the Tutorials database you should follow two steps:

Create an appropriately named database on the target machine

Load the file using the mysql command:

```
$ mysql -u [uname] -p[pass] [db_to_restore] < [backupfile.sql]
```

Have a look how you can restore your tut_backup.sql file to the Tutorials database.

```
$ mysql -u root -p Tutorials < tut_backup.sql
```

To restore compressed backup files you can do the following:

```
gunzip < [backupfile.sql.gz] | mysql -u [uname] -p[pass] [dbname]
```

If you need to restore a database that already exists, you'll need to use mysqlimport command. The syntax for mysqlimport is as follows:

```
mysqlimport -u [uname] -p[pass] [dbname] [backupfile.sql]
```

Backing Up and Restoring using PHPMyAdmin

It is assumed that you have phpMyAdmin installed since a lot of web service providers use it. To backup your MySQL database using PHPMyAdmin just follow a couple of steps:

Open phpMyAdmin.

Select your database by clicking the database name in the list on the left of the screen.

Click the Export link. This should bring up a new screen that says View dump of database (or something similar).

In the Export area, click the Select All link to choose all of the tables in your database.

In the SQL options area, click the right options.

Click on the Save as file option and the corresponding compression option and then click the 'Go' button. A dialog box should appear prompting you to save the file locally.

Restoring your database is easy as well as backing it up. Make the following:

Open phpMyAdmin.

Create an appropriately named database and select it by clicking the database name in the list on the left of the screen. If you would like to rewrite the backup over an existing database then click on the database name, select all the check boxes next to the table names and select Drop to delete all existing tables in the database.

Click the SQL link. This should bring up a new screen where you can either type in SQL commands, or upload your SQL file.

Use the browse button to find the database file.

Click Go button. This will upload the backup, execute the SQL commands and re-create your database.

14.8 Summary

In this unit, you have learned the importance of perl and MySQL. Perl Documentation and perl syntax rules are discussed in detail .A brief introduction about MySQL and Database Independent Interface, Table Joins, Loading and Dumping a Database are also discussed in this unit.

14.9 Key words

- Statement handles are the final type of object that DBI defines for database interaction and manipulation.
- Verbatim paragraphs are usually used for presenting a codeblock or other text which does not require any special parsing or formatting, and which shouldn't be wrapped.

14.10 Unit end exercise and answers

1. What Perl Documentation? Explain.
2. Explain about Perl Syntax Rules?
3. Give introduction to MySQL.
4. What is Database Independent Interface? Explain.
5. Explain Table Joins in detail.
6. Explain Loading and Dumping a Database?

Answers :see 1. 14.2
2. 14.3
3. 14.4
4. 14.5
5. 14.6
6. 14.7

14.11 Reference

Teach yourself Java - Joseph O'Neil
Java Programming – Herb Schildt
Java: The Complete Reference, J2SE 5 Edition Herbert Schildt
Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 20014. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 15: The Common Gateway Interface

Structure:

15.0 OBJECTIVES

At the end of this unit you will be able to know:

- Apache Configuration
- A First CGI Program
- CGI.pm Introduced
- CGI.pm HTML Shortcuts
- Information Received by the CGI Program
- Form Widget Methods
- CGI Security

- Considerations A Note About die()

15.1 INTRODUCTION

Provides an interface for arbitrary programs to provide content for web pages

• CGI spec: http://hoohoo.ncsa.uiuc.edu/cgi/interface.html

• CGI 2: http://cgi-spec.golux.com/

• Advantages of CGI

– Easy to write

– Easy to maintain

– Readily available examples for download

• Disadvantages

– Slow startup ! slow runtime

– No maintenance of state

– Most of the ’readily available examples’ are very badly written

• Alternatives to CGI

– mod perl

– PHP

– FastCGI

– ASP

– JSP

– etc, etc, etc

• Each addresses the above problems in different ways, but with many of the same general concepts:

– Persistent database connections

– In-memory interpreter

– Direct interface to the web server API

15.2 Apache Configuration

Apache is configured by placing directives in plain text configuration files. The main configuration file is usually called httpd.conf. The location of this file is set at compile-time, but may be overridden with the -f command line flag. In addition, other configuration files may be added using the Include directive, and wildcards can be used to include many configuration files. Any directive may be placed in any of these configuration files. Changes to the main configuration files are only recognized by Apache when it is started or restarted.

The server also reads a file containing mime document types; the filename is set by the TypesConfig directive, and is mime.types by default.

Syntax of the Configuration Files

Apache configuration files contain one directive per line. The backslash "\" may be used as the last character on a line to indicate that the directive continues onto the next line. There must be no other characters or white space between the backslash and the end of the line.

Directives in the configuration files are case-insensitive, but arguments to directives are often case sensitive. Lines that begin with the hash character "#" are considered comments, and are ignored. Comments may not be included on a line after a configuration directive. Blank lines and white space occurring before a directive are ignored, so you may indent directives for clarity.

The values of shell environment variables can be used in configuration file lines using the syntax ${ENVVAR}. If "ENVVAR" is the name of a valid environment variable, the value of that variable is substituted into that spot in the configuration file line, and processing continues as if that text were found directly in the configuration file. (If the ENVVAR variable is not found, the characters "${ENVVAR}" are left unchanged for use by later stages in the config file processing.)

The maximum length of a line in the configuration file, after environment-variable substitution, joining any continued lines and removing leading and trailing white space, is 8192 characters.

You can check your configuration files for syntax errors without starting the server by using apachectl configtest or the -t command line option.

Modules

Apache is a modular server. This implies that only the most basic functionality is included in the core server. Extended features are available through modules which can be loaded into Apache. By default, a base set of modules is included in the server at compile-time. If the server is compiled to use dynamically loaded modules, then modules can be compiled separately and added at any time using the LoadModule directive. Otherwise, Apache must be recompiled to add or remove modules. Configuration directives may be included conditional on a presence of a particular module by enclosing them in an <IfModule> block.

To see which modules are currently compiled into the server, you can use the -l command line option.

Scope of Directives

Directives placed in the main configuration files apply to the entire server. If you wish to change the configuration for only a part of the server, you can scope your directives by placing them in <Directory>, <DirectoryMatch>, <Files>, <FilesMatch>, <Location>, and <LocationMatch> sections. These sections limit the application of the directives which they enclose to particular filesystem locations or URLs. They can also be nested, allowing for very fine grained configuration.

Apache has the capability to serve many different websites simultaneously. This is called Virtual Hosting. Directives can also be scoped by placing them inside <VirtualHost> sections, so that they will only apply to requests for a particular website.

Although most directives can be placed in any of these sections, some directives do not make sense in some contexts. For example, directives controlling process creation can only be placed in the main server context. To find which directives can be placed in which sections, check the Context of the directive. For further information, we provide details on How Directory, Location and Files sections work.

15.3 A First CGI Program

A CGI script can be as simple or complex as you need it to be. It could be in Perl, Java, Python or any programming language. At it's core, a CGI application simply takes a request via HTTP (typically a web browser) and returns HTML. Let's look at a simple Hello World CGI script and break it down into it's simplest forms.

```
#!/usr/bin/perl

 print "Content-type: text/html\n\n";
 print <<HTML;
 <html>
 <head>
 <title>A Simple Perl CGI</title>
 </head>
 <body>
 <h1>A Simple Perl CGI</h1>
 <p>Hello World</p>
 </body>
 HTML
 exit;
```

If you run the program on the command line, you'll see that it does exactly what you'd expect. First it prints the Content-type line, then it prints the raw HTML. In order to see it in action in a web browser, you'll need to copy or upload the script to your web server and make sure the permissions are set correctly (chmod 755 on *nix systems). Once you've set it correctly, you should be able to browse to it and see the page displayed live on your server.

The key line is the first print statement:

```
 print "Content-type: text/html\n\n";
```

This tells the browser that the document coming after the two newlines is going to be HTML. You must send a header so the browser knows what type of document is coming next, and you must include a blank line between the header and the actual document.

Once the header is sent, it's just a matter of sending the HTML document itself. In the above example, we're using a here-doc to simplify printing a large chunk of plain text. Of course, this is really no different than having a plain HTML document sitting on your server. The real power of using a programming language like Perl to create your HTML comes when you add in some fancy Perl programming. In the next example, let's take part of our time and date script and put it in our web page.

```
 #!/usr/bin/perl

 @months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
 @weekDays = qw(Sun Mon Tue Wed Thu Fri Sat Sun);
 ($second, $minute, $hour, $dayOfMonth, $month, $yearOffset, $dayOfWeek, $dayOfYear,
$daylightSavings) = localtime();
 $year = 1900 + $yearOffset;
 $theTime = "$weekDays[$dayOfWeek] $months[$month] $dayOfMonth, $year";

 print "Content-type: text/html\n\n";
 print <<HTML;
 <html>
 <head>
 <title>A Simple Perl CGI</title>
 </head>
 <body>
 <h1>A Simple Perl CGI</h1>
 <p>$theTime</p>
 </body>
 HTML
 exit;
```

This new CGI script will insert the current date into the page each time the script is called. In other words, it becomes a dynamic document that changes as the date changes, rather than a static document.

15.4 CGI.pm Introduced

CGI.pm is a large and widely used Perl module for programming Common Gateway Interface (CGI) web applications, providing a consistent API for receiving user input and producing HTML or XHTML output.

Here is a simple CGI page, written in Perl using CGI.pm (in object-oriented style):

```
use CGI;
my $cgi = CGI->new();
print
    $cgi->header('text/html'),
    $cgi->start_html('A Simple CGI Page'),
    $cgi->h1('A Simple CGI Page'),
    $cgi->start_form,
    'Name: ',
    $cgi->textfield('name'), $cgi->br,
    'Age: ',
    $cgi->textfield('age'), $cgi->p,
    $cgi->submit('Submit!'),
    $cgi->end_form, $cgi->p,
    $cgi->hr;

if ( $cgi->param('name') ) {
    print 'Your name is ', $cgi->param('name'), $cgi->br;
}

if ( $cgi->param('age') ) {
    print 'You are ', $cgi->param('age'), ' years old.';
}

print $cgi->end_html;
```

The following table lists some basic CGI.pm functions for generating form fields and related elements

startform(*$method*,*$action*,*$encoding*)	starts a form
start_multipart_form(*$method*,*$action*)	starts a form that may contain a file input field
endform()	ends a form
textfield(-name=>'*field_name*', -default=>*initial value*, -size=>*number*,-maxlength=>*number*)	single-line text input field
textarea(-name=>'*field_name*', -default=>*initial value*,-rows=>*number*, -cols=>*number*)	multi-line text input field
password_field(-name=>'*field_name*', -size=>*number*,-maxlength=>*number*)	"password" input field
filefield(-name=>'*field_name*', -default=>*initial filename*)	file input field
popup_menu(-name=>'*menu_name*', -values=>*array or hash reference*, -default=>*initial selection*)	select element
scrolling_list(-name=>'*menu_name*', -values=>*array or hash reference*, -default=>*initial selection(s)*, -size=>*number*,-multiple=>'true')	select element withsize larger than 1
checkbox_group(-name=>'*group_name*', -values=>*array or hash reference*, -default=>*initial selection(s), -linebreak=>'true')*	group of checkboxes with the same name
checkbox(-name=>'*checkbox_name*', -checked=>'checked',-value=>*internal value*, -label=>*visible label*)	a standalone checkbox
radio_group(-name=>'*group_name*', -values=>*array or hash reference*, -default=>*initial selection(s), -linebreak=>'true')*	group of radio buttons that work together
submit(-name=>'*button_name*', -value=>*button text*)	submit button
reset(-value=>*button text*)	reset button
defaults(-value=>*button text*)	reset to original defaults
hidden(-name=>'*field_name*', -default=>*array reference*)	hidden field

image_button(-name=>'*button_name*', -src=>*image URL*,-align=>*alignment*, -alt=>*text*, -value=>*text*)	image submit button
button(-name=>'*button_name*', -value=>*button text*, -onClick=>*script*)	button element, for client-side scripting

Note that the statement use CGI qw(:standard); is needed to make thethe invocations just generate HTML constructs as strings; you need to use the print function to make them actually appear in a document generated by your CGI script.

15.5 CGI.pm HTML Shortcuts

In addition to the methods discussed previously, CGI.pm has functions for most of the commonly used HTML tags, including headings, paragraph breaks, , and text-formatting commands (i() , em() , blockquote())

Using these shortcuts is as straightforward as the previous example. Printing a heading:

<H1>Welcome to www.opensourcewebbook.com</H1>

becomes:

$server_name = `/bin/hostname`; print h1("Welcome to $server_name")

An HTML paragraph, <P> , is formatted as print p(); (no arguments), while

print p(This is a new paragraph);

is equivalent to this:

<P>This is a new paragraph</P>

A list of arguments gets concatenated . This CGI.pm code:

$name = John Doe; print p(hello , $name, , how are you?);

results in this:

<P>hello John Doe, how are you?</P>

Like HTML commands, the shortcuts can be nested inside one another:

<P>Here is some <B>bold</B> text</P>

The previous statement can be programmed in CGI.pm as

print p(Here is some , b(bold), text);

We can use one example to show many of these methods. The following program uses these shortcuts and others as well as a Perl built-in function (localtime()) and a CGI.pm method (server_name()). In the end, with all this fancy Perl programming, this program creates the same web page created in the earlier example (see Figure 7.5) but in a much more flexible way:

```
#! /usr/bin/perl  # info15.cgi   use strict;   use CGI:standard;   my $host = server_name();  my $date = localtime();    print  header(),  start_html(-title => System Information,  -bgcolor => #520063,  -text => #ffffff),  h1("Hello from $host!"),  "The current time is now: $date", end_html();
```

After CGI.pm is use d, the program gets the local time and the hostname, as in the earlier example. The server_name() function, provided by CGI.pm, returns the name of the webserver . The localtime() function is a Perl built-in function that determines the local time on the machine; when assigned to a scalar, here $date , it returns the time in a nice, readable format. After these dynamic values are obtained, the HTML is printed, using the just-obtained values of $host and $date .

15.6 Form Widget Methods

The CGI::Test::Form class provides an interface to the content of the CGI forms. Instances are automatically created by CGI::Test when it analyzes an HTML output from a GET/POST request and encounters such beasts.

This class is really the basis of the CGI::Test testing abilities: it provides the necessary routines to query the CGI widgets present in the form: buttons, input areas, menus, etc... Queries can be made by type, and by name. There is also an interface to specifically access groupped widgets like checkboxes and radio buttons.

All widgets returned by the queries are polymorphic objects, heirs of CGI::Test::Form::Widget. If the querying interface can be compared to the human eye, enabling you to locate a particular graphical item on the browser screen, the widget interface can be compared to the mouse and keyboard, allowing you to interact with the located graphical components. Please refer to CGI::Test::Form::Widget for interaction details.

Apart from the widget querying interface, this class also offers a few services to other CGI::Test components, like handling of reset and submit actions, which need not be used directly in practice.

Finally, it provides inspection of the <FORM> tag attributes (encoding type, action, etc...) and, if you really need it, to the HTML tree of the all <FORM> content. This interface is based on the HTML::Element class, which represents a tree node. The tree is shared with other CGI::Test components, it is not a private copy. See HTML::Element if you are not already familiar with it.

If memory is a problem, you must be aware that circular references are used almost everywhere within CGI::Test. Because Perl's garbage collector cannot reclaim objects that are part of such a reference loop, you must explicitely call the delete method on CGI::Test::Form. Simply forgetting about the reference to that object is not enough. Don't bother with it if your regression test scripts die quickly. A form consists of two distinct parts: the HTML code and the CGI program. HTML tags create the visual representation of the form, while the CGI program decodes (or processes) the information contained within the form.

15.7 CGI Security Considerations

The Apache HTTP Server has a good record for security and a developer community highly concerned about security issues. But it is inevitable that some problems -- small or large -- will be discovered in software after it is released. For this reason, it is crucial to keep aware of updates to the software. If you have obtained your version of the HTTP Server directly from Apache, we highly recommend you subscribe to the Apache HTTP Server Announcements List where you can keep informed of new releases and security updates. Similar services are available from most third-party distributors of Apache software.

Of course, most times that a web server is compromised, it is not because of problems in the HTTP Server code. Rather, it comes from problems in add-on code, CGI scripts, or the

underlying Operating System. You must therefore stay aware of problems and updates with all the software on your system.
All network servers can be subject to denial of service attacks that attempt to prevent responses to clients by tying up the resources of the server. It is not possible to prevent such attacks entirely, but you can do certain things to mitigate the problems that they create.

Often the most effective anti-DoS tool will be a firewall or other operating-system configurations. For example, most firewalls can be configured to restrict the number of simultaneous connections from any individual IP address or network, thus preventing a range of simple attacks. Of course this is no help against Distributed Denial of Service attacks (DDoS).

There are also certain Apache HTTP Server configuration settings that can help mitigate problems:

The RequestReadTimeout directive allows to limit the time a client may take to send the request.
The TimeOut directive should be lowered on sites that are subject to DoS attacks. Setting this to as low as a few seconds may be appropriate. As TimeOut is currently used for several different operations, setting it to a low value introduces problems with long running CGI scripts.
The KeepAliveTimeout directive may be also lowered on sites that are subject to DoS attacks. Some sites even turn off the keepalives completely via **KeepAlive**, which has of course other drawbacks on performance.
The values of various timeout-related directives provided by other modules should be checked.
The directives LimitRequestBody, LimitRequestFields, LimitRequestFieldSize, LimitRequestLine, and LimitXMLRequestBody should be carefully configured to limit resource consumption triggered by client input.
On operating systems that support it, make sure that you use the AcceptFilter directive to offload part of the request processing to the operating system. This is active by default in Apache httpd, but may require reconfiguration of your kernel.
Tune the MaxRequestWorkers directive to allow the server to handle the maximum number of simultaneous connections without running out of resources. See also the performance tuning documentation.

The use of a threaded mpm may allow you to handle more simultaneous connections, thereby mitigating DoS attacks. Further, the event mpm uses asynchronous processing to avoid devoting a thread to each connection. Due to the nature of the OpenSSL library the event mpm is currently incompatible with mod_ssl and other input filters. In these cases it falls back to the behaviour of the worker mpm.

There are a number of third-party modules available through http://modules.apache.org/ that can restrict certain client behaviors and thereby mitigate DoS problems.

All the CGI scripts will run as the same user, so they have potential to conflict (accidentally or deliberately) with other scripts e.g. User A hates User B, so he writes a script to trash User B's CGI database. One program which can be used to allow scripts to run as different users is suEXEC which is included with Apache as of 115.2 and is called from special hooks in the Apache server code. Another popular way of doing this is with CGIWrap.

Allowing users to execute CGI scripts in any directory should only be considered if:

You trust your users not to write scripts which will deliberately or accidentally expose your system to an attack.

You consider security at your site to be so feeble in other areas, as to make one more potential hole irrelevant.

You have no users, and nobody ever visits your server.

Limiting CGI to special directories gives the admin control over what goes into those directories. This is inevitably more secure than non script aliased CGI, but only if users with write access to the directories are trusted or the admin is willing to test each new CGI script/program for potential security holes. Most sites choose this option over the non script aliased CGI approach.

Embedded scripting options which run as part of the server itself, such as mod_php, mod_perl, mod_tcl, and mod_python, run under the identity of the server itself (see the User directive), and therefore scripts executed by these engines potentially can access anything the server user can. Some scripting engines may provide restrictions, but it is better to be safe and assume not.

When setting up dynamic content, such as mod_php, mod_perl or mod_python, many security considerations get out of the scope of httpd itself, and you need to consult documentation from those modules. For example, PHP lets you setup Safe Mode, which is most usually disabled by default. Another example is Suhosin, a PHP addon for more security. For more information about those, consult each project documentation.

At the Apache level, a module named mod_security can be seen as a HTTP firewall and, provided you configure it finely enough, can help you enhance your dynamic content security.

15.8 A Note About die()

The most common method that Perl developers use for handling errors is Perl's built-in die function. Here is an example:

```
open FILE, $filename or die "Cannot open $filename: $!";
```

If Perl is unable to open the file specified by $filename, die will print an error message to STDERR and terminate the script. The open function, like most Perl commands that interact with the system, sets $! to the reason for the error if it fails.

Unfortunately, die is not always the best solution for handling errors in your CGI scripts.

utput to STDERR is typically sent to the web server's error log, triggering the web server to return a 500 Internal Server Error . This is certainly not a very user-friendly response.

You should determine a policy for handling errors on your site. You may decide that 500 Internal Server Error pages are acceptable for very uncommon system errors like the inability to read or write to files. However, you may decide that you wish to display a formatted HTML page instead with information for users such as alternative actions they can take or who to notify about the problem.

It is possible to trap die so that it does not generate a 500 Internal Server Error automatically. This is especially useful because many common third-party modules use die (and variants such as croak) as their manner for responding to errors. If you know that a particular subroutine may call die, you can catch this with an eval block in Perl:

```
eval {
   dangerous_routine(  );
   1;
} or do {
   error( $q, $@ || "Unknown error" );
};
```

If dangerous_routine does call die, then eval will catch it, set the special variable $@ to the value of the die message, pass control to the end of the block, and return undef. This allows us to call another subroutine to display our error more gracefully. Note that an eval block will not trap exit. This works, but it certainly makes your code a lot more complex, and if your CGI script interacts with a lot of subroutines that might die, then you must either place your entire script within an eval block or include lots of these blocks throughout your script.

Fortunately, there is a better way. You may already know that it is possible to create a global signal handler to trap Perl's die and warn functions. This involves some rather advanced Perl; you can find specific information in Programming Perl. Fortunately, we don't have to worry about the specifics, because there is a module that not only does this, but is written specifically for CGI scripts: CGI::Carp.

15.9 Summary

In this unit, you have learned about the The Common Gateway Interface, importance of also Apache configuration. First a CGI program is explained and what things can go worong with a program is explained. CGI.pm is Introduced, CGI.pm HTML Shortcuts, how Information is Received by the CGI Program, which are the Form Widget Methods, what are the CGI Security Considerations and a Note About die() function is discussed in detail.

15.10 Key words

Directives: placed in the main configuration files apply to the entire server.

CGI.pm: is a large and widely used Perl module for programming Common Gateway Interface (CGI) web applications, providing a consistent API for receiving user input and producing HTML or XHTML output.

15.11 Unit end exercise and answers

1. How can we do is Apache Configuration?
2. Write a CGI Program?
3. Give introduction about CGI.pm?

4. Which are the CGI.pm HTML Shortcuts?
5. How to receive Information by the CGI Program?
6. What is Form Widget Methods?
7. What are CGI Security Considerations?
8. Write a Note about die ()?

Answers :see 1. 15.2

2. 15.3

3. 15.4

4. 15.5

5. 15.6

6. 15.7

7. 15.8

8. 15.9

15.12 Reference

Teach yourself Java - Joseph O'Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 20014. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks

UNIT 16: PHP and Server Side Includes

Structure:

16.0 OBJECTIVES

At the end of this unit you will be able to know:

- Security Considerations.
- Introduction on PHP
- Embedding PHP into HTML
- Configuration
- Quick Examples
- Language Syntax
- Built-In PHP Functions
- PHP and MySQL

16.1 Introduction

In the general programming world, is it often expected for a good programmer to separate his program into many sections or blocks to enable code re-use and easy maintenance of the scripts. Using server side includes we can produce more modular scripts, as we are able to include pieces of code that have been previously written.

Using a feature that PHP offers, use we are allowed to do these server side includes to include both HTML and PHP pages, and actually any pages we really want to. Including HTML pages becomes very useful, in creating the same look and feel over your entire site, as we can include a standard header and footer on every page.

There are in fact, four different types of includes, all surprisingly similar. They are; include(), require(), include_once() and require_once().
Include
The function include does exactly as it suggests, it simply includes the requested file into your PHP, and if it doesn't find it, it gives a minor error.
SSI stands for Server Side Include and means that you can include a file as part of your document while still on the server before it is executed and before the data reaches the client. One reason for doing this is that often data such as a header, footer, or variables are repeated in many places on the site. Using SSI allows these to each be stored in only one place, and thus only updated in one place when changes are made.

16.2 Security Considerations.

A completely secure system is a virtual impossibility, so an approach often used in the security profession is one of balancing risk and usability. If every variable submitted by a user required two forms of biometric validation (such as a retinal scan and a fingerprint), you would have an extremely high level of accountability. It would also take half an hour to fill out a fairly complex form, which would tend to encourage users to find ways of bypassing the security.

The best security is often unobtrusive enough to suit the requirements without the user being prevented from accomplishing their work, or over-burdening the code author with excessive complexity. Indeed, some security attacks are merely exploits of this kind of overly built security, which tends to erode over time.

A phrase worth remembering: A system is only as good as the weakest link in a chain. If all transactions are heavily logged based on time, location, transaction type, etc. but the user is only verified based on a single cookie, the validity of tying the users to the transaction log is severely weakened.

When testing, keep in mind that you will not be able to test all possibilities for even the simplest of pages. The input you may expect will be completely unrelated to the input given by a disgruntled employee, a cracker with months of time on their hands, or a housecat walking across the keyboard. This is why it's best to look at the code from a logical perspective, to discern where unexpected data can be introduced, and then follow how it is modified, reduced, or amplified.

The Internet is filled with people trying to make a name for themselves by breaking your code, crashing your site, posting inappropriate content, and otherwise making your day interesting. It doesn't matter if you have a small or large site, you are a target by simply being online, by having a server that can be connected to. Many cracking programs do not discern by size, they simply trawl massive IP blocks looking for victims.
Even after being a programmer-friendly language, a number of issues are there in php which can cause serious security related threats in a web application. The effect of such a security hole can cause blunders to a business, if it is heavily dependent on a web application. Since we all know the advantages of PHP, some of the potential void areas in php development are:

Access Control: If certain confidential or crucial sections of an application are not secured properly, the access control issues arise in the web application. Taking an example, if normal user with non-administrative privilege logs in to the application and re-writes the URL of the web app to the address location of administrator, it's quite possible that the user can by-pass the administrator's log in section and gain the authority of the administrator without authenticating himself. Preventing the sensitive pages from hijacking can be practiced by placing them in a separate directory protected by the .htaccess file.

Session ID protection: Each time a user logs in his account, a session ID is assigned which identifies the user during the session. In a php based web application, hijacking of the session ID is a common problem where if session ID is known to an intruder, he can easily gain access to the user's session. This can be a problem in events like password reset, credit card authentication etc. In such circumstances, re-validating the user's credentials for highly sensitive actions is advised to protect the session from any hacking attempt.

Cross Site Scripting or XSS attempts: Another type of security issue that arises in a php application is the cross site scripting, malicious scripting codes which can modify the behavior of the web page. Crucial session information like session IDs, cookies information are passed to the hijacker, who can use it for unauthorized and unintended purposes and can cause harm to user's confidentiality. To prevent a web app from such type of attacks, the submission of HTML tags like "<" and ">" should be prohibited.

Problem in error reporting: If the value of the display_error in the php.ini file is not set to "0" results of all the errors conditions, during the execution of the code, will be displayed in the user's browser. A technically sound user can gain advantage of knowing the internal functionality of the application by providing error causing or bad inputs in the application. Further, by knowing the execution procedure of the application, the attacker can find out the entry point in the system and can modify the behavior of the system according to his own choice.

16.3 PHP: Introduction

PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language.

PHP code is interpreted by a web server with a PHP processor module, which generates the resulting web page: PHP commands can be embedded directly into an HTML source document rather than calling an external file to process data. It has also evolved to include a command-line interface capability and can be used in standalone graphical applications.PHP is free software

released under the PHP License. PHP can be deployed on most web servers and also as a standalone shell on almost every operating system and platform, free of charge.
PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 19916.

PHP is a recursive acronym for "PHP: Hypertext Preprocessor".

PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.

It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.

PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.

PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.

PHP is forgiving: PHP language tries to be as forgiving as possible.

PHP Syntax is C-Like.

Common uses of PHP:

PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.

PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.

You add, delete, modify elements within your database thru PHP.

Access cookies variables and set cookies.

Using PHP, you can restrict users to access some pages of your website.

It can encrypt data.

Characteristics of PHP

Five important characteristics make PHP's practical nature possible:

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

16.4 Embedding PHP into HTML

When building a complex page, at some point you will be faced with the need to combine PHP and HTML to achieve your needed results. At first point, this can seem complicated, since PHP and HTML are two separate languages, but this is not the case. PHP is designed to interact with HTML and PHP scripts can be included in an HTML page without a problem.

There are two ways to use HTML on your PHP page. The first way is to put the HTML outside of your PHP tags. You can even put it in the middle if you close and reopen the <?php -and- ?> tags. Here is an example of putting the HTML outside of the tags:

```
<html>
<title>HTML with PHP</title>
<body>
<h1>My Example</h1>

<?php
//your php code here
?>

<b>Here is some more HTML</b>

<?php
//more php code
?>



</body>
</html>
```

As you can see you can use any HTML you want without doing anything special or extra in your .php file, as long as it is outside of the PHP tags.

The second way to use HTML with PHP is by using PRINT or ECHO. By using this method you can include the HTML inside of the PHP tags. This is a nice quick method if you only have a line or so to do. Here is an example:

```
<?php
Echo "<html>";
Echo "<title>HTML with PHP</title>";
Echo "<b>My Example</b>";

//your php code here
```

```
 Print "<i>Print works too!</i>";
 ?>
```

Using one or both of these methods you can easily embed HTML code in your PHP pages, to give them a nicer more formatted look, and make them more user friendly.

In an HTML page, PHP code is enclosed within special PHP tags. When a visitor opens the page, the server processes the PHP code and then sends the output (not the PHP code itself) to the visitor's browser. Actually it is quite simple to integrate HTML and PHP. A PHP script can be treated as an HTML page, with bits of PHP inserted here and there. Anything in a PHP script that is not contained within **<?php ?>** tags is ignored by the PHP compiler and passed directly to the web browser.

```
<html>
<head></head>
<body class="page_bg">
Hello, today is <?php echo date('l, F jS, Y'); ?>.
</body>
</html>
```

16.5 Configuration

The configuration file (php.ini) is read when PHP starts up. For the server module versions of PHP, this happens only once when the web server is started. For the CGI and CLI versions, it happens on every invocation.

php.ini is searched for in these locations (in order):

SAPI module specific location (PHPIniDir directive in Apache 2, -c command line option in CGI and CLI, php_ini parameter in NSAPI, PHP_INI_PATH environment variable in THTTPD)

The PHPRC environment variable. Before PHP 5.116.0, this was checked after the registry key mentioned below.

As of PHP 5.116.0, the location of the php.ini file can be set for different versions of PHP. The following registry keys are examined in order: [HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x.y.z], [HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x.y] and [HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x], where x, y and z mean the PHP major, minor and release versions. If there is a value for IniFilePath in any of these keys, the first one found will be used as the location of the php.ini (Windows only).
[HKEY_LOCAL_MACHINE\SOFTWARE\PHP], value of IniFilePath (Windows only).
Current working directory (except CLI).
The web server's directory (for SAPI modules), or directory of PHP (otherwise in Windows).
Windows directory (C:\windows or C:\winnt) (for Windows), or --with-config-file-path compile time option.
If php-SAPI.ini exists (where SAPI is the SAPI in use, so, for example, php-cli.ini or php-apache.ini), it is used instead of php.ini. The SAPI name can be determined with php_sapi_name().

16.6 Quick Examples

To get a feel for PHP, start with simple PHP scripts. Since "Hello, World!" is an essential example, will create a friendly little "Hello, World!" script.

As mentioned, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this:

```
<html>
<head>
<title>Hello World</title>
<body>
   <?php echo "Hello, World!";?>
</body>
</html>
```

It will produce following result:

```
Hello, World!
```

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.

All PHP code must be included inside one of the three special markup tags ate are recognised by the PHP Parser.

```
<?php PHP code goes here ?>

<?    PHP code goes here ?>

<script language="php"> PHP code goes here </script>
```

16.7 Language Syntax

A PHP script can be placed anywhere in the document.A PHP script starts with <?php and ends with ?>:

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is ".php".

A PHP file normally contains HTML tags, and some PHP scripting code.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

```
<!DOCTYPE html>
<html>
```

```
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

16.8 Built-In PHP Functions

PHP comes standard with many functions and constructs. There are also functions that require specific PHP extensions compiled in, otherwise fatal "undefined function" errors will appear. For example, to use image functions such as imagecreatetruecolor(), PHP must be compiled with GD support. Or, to use mysql_connect(), PHP must be compiled with MySQL support. There are many core functions that are included in every version of PHP, such as the string and variable functions. A call to phpinfo() or get_loaded_extensions() will show which extensions are loaded into PHP. Also note that many extensions are enabled by default and that the PHP manual is split up by extension. See the configuration, installation, and individual extension chapters, for information on how to set up PHP.

Reading and understanding a function's prototype is explained within the manual section titled how to read a function definition. It's important to realize what a function returns or if a function works directly on a passed in value. For example, str_replace() will return the modified string while usort() works on the actual passed in variable itself. Each manual page also has specific information for each function like information on function parameters, behavior changes, return values for both success and failure, and availability information. Knowing these important (yet often subtle) differences is crucial for writing correct PHP code.
PHP is very rich in terms of Buil-in functions. Here is the list of various important function categories. There are various other function categories which are not covered here

- PHP Array Functions
- PHP Calender Functions
- PHP Class/Object Functions

- PHP Character Functions
- PHP Date & Time Functions
- PHP Directory Functions
- PHP Error Handling Functions
- PHP File System Functions
- PHP MySQL Functions
- PHP Network Functions
- PHP ODBC Functions
- PHP String Functions
- PHP SimpleXML Functions
- PHP XML Parsing Function

16.9 PHP and MySQL

When setting out to build an e-commerce site, there are many different products that you could use.
You will need to choose the following:

- Hardware for the Web server
- An operating system
- Web server software
- A database management system
- A programming or scripting language

Some of these choices will be dependent on the others. For example, not all operating systems will run on all hardware, not all scripting languages can connect to all databases, and so on.
One of the nice features of PHP is that it is avail-able for Microsoft Windows, for many versions of Unix, and with any fully functional Web server. MySQL is similarly versatile.
To demonstrate this, the examples in this book have been written and tested on two
popular setups:

- Linux using the Apache Web server

- Microsoft Windows 2000 using Microsoft Internet Information Server (IIS)

Whatever hardware, operating system, and Web server you choose, we believe you should seriously consider using PHP and MySQL
Some of PHP’s main competitors are Perl, Microsoft Active Server Pages (ASP), Java
Server Pages (JSP), and Allaire ColdFusion.
In comparison to these products, PHP has many strengths, including the following:

- High performance
- Interfaces to many different database systems
- Built-in libraries for many common Web tasks
- Low cost
- Ease of learning and use
- Portability
- Availability of source code

Some of MySQL’s Strengths
Some of MySQL’s main competitors are PostgreSQL, Microsoft SQL Server, and Oracle.
MySQL has many strengths, including the following:

- High performance
- Low cost
- Easy to configure and learn
- Portable
- The source code is available

PHP is a powerful programming language that lets you build dynamic Web sites. It works well on a variety of platforms, and it's reasonably easy to understand. MySQL is an impressive relational data management system used to build commercial quality databases. PHP and MySQL are such powerful and easy-to-use platforms that they make Web programming accessible even for beginners

16.10 Summary

In this unit, you have learned about Server Side Includes also the importance of PHP and MySQL in java. This unit begins with Security Considerations ,introduction to PHP, how Embedding PHP into HTML, how to Configuration, a Quick Examples of PHP, Language Syntax, which are the Built-In PHP Functions and last how PHP and MySQL are inter-related.

16.11 Key words

- **SSI:** stands for Server Side Include and means that you can include a file as part of your document while still on the server before it is executed and before the data reaches the client.
- Language Syntax: Visual Basic rules that define the correct way to use keywords, operators, and programmer-defined names.

16.12 Unit end exercise and answers

1. What are the Security Considerations?
2. Give introduction to PHP
3. How to Embedding PHP into HTML?
4. How to Configure?
5. Give an Examples?
6. Explain about Language Syntax
7. Which are the Built-In PHP Functions? Explain.
8. Explain about PHP and MySQL?

Answers :see 1. 16.2

2. 16.3

3. 16.4

4. 16.5

5. 16.6

6. 16.7
7. 16.8
8. 16.9

16.13 Reference

Teach yourself Java - Joseph O'Neil

Java Programming – Herb Schildt

Java: The Complete Reference, J2SE 5 Edition Herbert Schildt

Open Source Web Development with LAMP James Lee, Brent Ware

Java Hand Book, TMH, 200116. Patrick Naughton

Programming with Java, 2nd Edition.Balaguruswamy

CGI Programming with Perl, 2nd Edition. Scott Guelich, Shishir Gundavaram, Gunther Birznieks