UNIVERSITY OF DAR ES SALAAM
COMPUTING CENTRE

Diploma in Computing and Information Technology

CIT06104: OBJECT ORIENTED PROGRAMMING

University Road
P.O Box 35062
Dar es Salaam
Tanzania

Tel: +255 (022) 2410645
Fax: +255 (022) 2410690
Email: ***training@udsm.ac.tz***
Internet: ***http://www.ucc.co.tz***

Table of contents

Introduction

Course Description

The course introduces students to key features of object oriented programming languages and utilization of common programming structures and constructs of Object Oriented Programming Language in creating computer programs. The course also helps a student to design large-scale software systems based on requirements specification

Course Objectives

At the end of the course students should be able to

1. Describe characteristics of object oriented programming language
2. Describe object and its relation with class
3. Compare and contrast object oriented programming with procedural programming languages
4. Apply control structures and modular approach in creating computer programs
5. Apply classes and methods in program development
6. Apply overload and override in programs development
7. Create and use abstract data types
8. Design modular solutions to a given problem statement
9. Develop software by employing the principles of encapsulation, data abstraction and polymorphism
10. Implement abstract classes and interfaces in software development
11. Make appropriate use of advanced features such as inheritance, exception handling, I/O, references and GUIs in software implementation

Delivery Methodology

The course will be delivered in form of lectures in the classroom and in the Computer lab accordingly. Exercise with real life nature will be provided during and at the end of the class. The manual is also designed such that one can follow the course at own time and pace

Chapter 1: Introduction to java

1.1. History of java

Java, having been developed in 1991, is a relatively new programming language. At that time, James Gosling from Sun Microsystems and his team began designing the first version of Java aimed at programming home appliances which are controlled by a wide variety of computer processors.

Gosling's new language needed to be accessible by a variety of computer processors. In 1994, he realized that such a language would be ideal for use with web browsers and Java's connection to the internet began. In 1995, Netscape Incorporated released its latest version of the Netscape browser which was capable of running Java programs.

Why is it called Java? It is customary for the creator of a programming language to name the language anything he/she chooses. The original name of this language was Oak, until it was discovered that a programming language already existed that was named Oak. As the story goes, after many hours of trying to come up with a new name, the development team went out for coffee and the name Java was born.

While Java is viewed as a programming language to design applications for the Internet, it is in reality a general all purpose language which can be used independent of the Internet.

What is Java

Java is a high level, robust, secured and object-oriented programming language.

Where it is used?

According to Sun, 3 billion devices run java. There are many devices where java is currently used. Some of them are as follows:

- Desktop Applications such as acrobat reader, media player, antivirus etc.
- Web Applications such as irctc.co.in, javatpoint.com etc.
- Enterprise Applications such as banking applications.
- Mobile
- Embedded System
- Smart Card
- Robotics
- Games etc.

1.2. Types of Java Applications

There are mainly 4 types of applications that can be created using java programming:

1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. Abstract Window Toolkit (AWT) and Swing are used in java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) Enterprise Application

An application that is distributed in nature. Some of the more common types of enterprise applications include automated billing systems, payment processing, content management Enterprise Resource Planning (ERP), and Business Intelligence e.t.c.

It has the advantage of high level security, load balancing and clustering. In java, Enterprise Java Beans (EJB) is used for creating enterprise applications. Applications are designed to solve problems encountered by large enterprises. Enterprise applications are not only useful for large corporations, agencies, and governments, however. The benefits of an enterprise application are helpful, even essential, for individual developers and small organizations in an increasingly networked world.

The features that make enterprise applications powerful, like security and reliability, often make these applications complex. The Java EE platform is designed to reduce the complexity of enterprise application development by providing a development model, API, and runtime environment that allows developers to concentrate on functionality.

4) Mobile Application

An application that is created for mobile devices. Currently Android and Java Micro Edition(ME) are used for creating mobile applications.

1.3. Features of java

The following are features of Java:

- Object Oriented: In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- Platform independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.
- Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP Java would be easy to master.
- Secure: With Java's secure feature enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- Architectural-neutral: Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
- Portable: Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in

Java is written in ANSI C with a clean portability boundary which is a POSIX subset.

- Robust: Robust simply means strong. Java uses strong memory management. There is lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points make java robust.
- Multithreaded: With Java's multithreaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.
- Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light weight process.
- High Performance: With the use of Just-In-Time compilers, Java enables high performance.
- Distributed: Java is designed for the distributed environment of the internet.
- Dynamic: Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

1.4. Tools needed to create and run a java program

For creating a program in java, you will need a computer withProcessor: Minimum Pentium 2 with 266 MHz and 8 MB of RAM.

You will also need the following software:

- Operating system; Linux or Windows
- Java JDK 8
- Microsoft Notepad or any other text editor

1.5. Local Environment Setup

This section guides you on how to download and set up Java on your machine. Please follow the following steps to set up the environment.

Java SE is freely available from www.sun.com (this will redirect you to https://www.oracle.com/sun/index.html). So download a version based on your operating system.

Follow the instructions to download java and run the .exe to install Java on your machine. Once you installed Java on your machine, you would need to set environment variables to point to correct installation directories.

Setting up the path for windows:

Assuming you have installed Java in c:\Program Files\java\jdk directory:

- Right-click on 'My Computer' and select 'Properties'.

- Click on the 'Environment variables' button under the 'Advanced' tab.
- Now, alter the 'Path' system variable so that it also contains the path to the Java executable. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM', then change your path to read 'C:\WINDOWS\SYSTEM;c:\Program Files\java\jdk\bin'.

Setting up the path for Linux, UNIX, Solaris, FreeBSD:

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation if you have trouble doing this.

Example, if you use bash as your shell, then you would add the following line to the end of your '.bashrc: export PATH=/path/to/java:$PATH'

1.6. Popular Java Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market. But for now, you can consider one of the following:

- Notepad: On Windows machine you can use any simple text editor like Notepad, Notepad++ or TextPad.
- Netbeans: is a Java IDE that is open-source and free which can be downloaded from https://netbeans.org/downloads/index.html.
- Eclipse: is also a Java IDE developed by the eclipse open-source community and can be downloaded from http://www.eclipse.org/.

1.7. How to create and run a Java Program in the Computer

How to Input and Save a Java Program in the Computer

A java program is typed in a text editor and saved in a storage device preferable a hard disk. A java program is stored in text mode so that the programmer can read the program as it was written.

Procedure

->click start

->click All Programs

->click Accessories

->click Notepad

Type the following program in notepad:-

```
//program Hello world
class Hello
{
public static void main(String args[])
```

```
{

        System.out.println("Hello World");
}
}
```

Save the file as text file with the name Hello.java and save it in your home directory eg if your account name is Salima and you are using windows then save in c:\users\salima\Hello.java

How to Compile a Java Program

Before a program can be run on your computer, it must first be compiled. Compilation is a process that will transform your program into a form the computer can execute.

The computer cannot execute the Java statements as they currently appear in the program; the statements must be translated to an intermediate form for execution.

The output from the compiler is the same program, now represented by a set of Java byte codes. Java byte codes are a set of instructions written for a hypothetical computer, known as the Java virtual machine.

In addition to translation, a compiler reports on any grammatical errors made by the programmer in the language statements of the program. If errors are reported, it is necessary to return to the editor to correct the errors, resave the program, and then recompile the program.

If you are working on a PC using Microsoft's Windows environment, then open an MSDOS window and change your subdirectory to wherever you saved your Java program. The command to compile a Java program using the Java SDK is javac. To compile the first program listed, you would issue the following command:

javac Hello.java

The byte code produced by the compiler will be stored in a file called Hello.class. You should not try to edit or print a class file.

If you get errors listed in the MSDOS window, they could be caused by the following problems.

- You have not modified the path entry of your system environmental variable correctly and the computer cannot execute the command javac to compile your program.

- You have made a mistake when typing the program and the syntax of at least one statement might be incorrect.

You should carefully examine all three cases and make any necessary amendments before you recompile the program.

How to Execute (run) a Java Program

The program stored as Java byte codes is loaded into the memory of the computer, and is read and translated by an interpreter. An interpreter will read the byte code one "line" at a time, and translate each line, in turn, into a sequence of commands that can be directly executed by the computer. There exist different interpreters for different computers. The interpreter reads the respective byte codes and instructs the computer to execute the meanings of the instructions.

If the compilation is successful, you can execute (run) the program. The command to execute or run a Java program using the Java SDK is java. To execute the HELLO WORLD program you would issue the following command
in the same window where you compiled the program:

java Hello

If the program has executed correctly, the output should appear in your screen as Hello World

You can stop the program by clicking on the X in the upper-right corner of the window, by choosing Close from the pull-down menu or
by pressing the Alt-F key combination.

It is possible for a program to fail during the execution phase, in which case it must be stopped from any further execution. If modifications to the program are required, it is necessary to perform the amendments on the source file, recompile and rerun the program

1.8. SDK Tools

In order to build Java programs on your computer, the SDK contains a set of tools for compiling and executing your programs, plus a variety of other utilitarian features. The two tools that you used in this chapter and that you will use extensively throughout this manual are:

- **javac**—the Java Language Compiler that you use to compile programs written in the Java programming language into bytecodes.

- **java**—the Java Interpreter that you use to run programs written in the java programming language

EXCERCISES

1. What is Java?
2. Why are interpreted languages slower than compiled ones?
3. explain different features of java
4. explain how you will type and run a java program

Chapter 2. OOP CONCEPTS

2.1 Java OOPs Concepts

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which are data structures that contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.

Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.

Simula is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object is known as truly object-oriented programming language.

Smalltalk is considered as the first truly object-oriented programming language. OOP simplifies the software development and maintenance by providing some concepts:

- Encapsulation
- Inheritance
- Polymorphism
- Data abstraction

Encapsulation

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class, therefore it is also known as data hiding.

Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Data abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.
In java, we use abstract class and interface to achieve abstraction.

2.2 Procedural Programming Languages

Programs are made up of modules, which are parts of a program that can be coded and tested separately, and then assembled to form a complete program. In

procedural languages (i.e. C) these modules are procedures, where a procedure is a sequence of statements. In C for example, procedures are a sequence of imperative statements, such as assignments, tests, loops and invocations of sub procedures. These procedures are functions, which map arguments to return statements.

The design method used in procedural programming is called Top Down Design. This is where you start with a problem (procedure) and then systematically break the problem down into sub problems (sub procedures). This is called functional decomposition, which continues until a sub problem is straightforward enough to be solved by the corresponding sub procedure. The difficulties with this type of programming, is that software maintenance can be difficult and time consuming. When changes are made to the main procedure (top), those changes can cascade to the sub procedures of main, and the sub-sub procedures and so on, where the change may impact all procedures in the pyramid.

2.3 Object-Oriented Programming Languages

OBJECT-ORIENTED PROGRAMMING is a programming language that uses classes and objects to create models based on the real world environment. An OBJECT-ORIENTED PROGRAMMING application may use a collection of objects which will pass messages when called upon to request a specific service or information. Objects are able to pass, receive messages or process information in the form of data.

One reason to use OBJECT-ORIENTED PROGRAMMING is because it makes it easy to maintain and modify existing code as new objects are created inheriting characteristics from existing ones. This cuts down the development time considerably and makes adjusting the program much simpler.

Another reason to use OBJECT-ORIENTED PROGRAMMING is the ease of development and ability for other developers to understand the program after development. Well commented objects and classes can tell a developer the process that the developer of the program was trying to follow. It can also make additions to the program much easier for the new developer.

The last reason to use OBJECT-ORIENTED PROGRAMMING is the efficiency of the language. Many programming languages using OBJECT-ORIENTED PROGRAMMING will dump or destroy unused objects or classes freeing up system memory. By doing this the system can run the program faster and more effectively.

2.4 Advantage of OOPs over Procedure-oriented programming language

The following are advantages of OOP over procedure-oriented programming language: -

- OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows

- OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere
- OOPs provide ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

EXCERCISES

1. What is OOP?
2. What are the four pillars of OOP?
3. Differentiate procedure from OO programming

Chapter 3. Basic concepts

3.1 Statements and expressions

3.1.1 Expressions

An *expression* is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language that evaluates to a single value. Example of an expressions, is illustrated in bold below:

int **cadence = 0**;

3.1.2 Statements

Statements are roughly equivalent to sentences in natural languages. A *statement* forms a complete unit of execution. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;).

```
// assignment statement
aValue = 8933.234;
// increment statement
aValue++;
```

3.2 Comments

Comments are added in the code to provide information about the code you write and this makes the code easier to understand. All professionally written codes will have sufficient comments in it to make the code easily readable and easily maintainable.
Java support three kinds of comments and they are given below:-

- Single line comment:
  Single line comment begins with // and ends at the end of the line.
- Multi line comment
  Multi line comment begins with /* and ends with */ that spans multiple lines.
- Document comment
  Documentation style comments begin with /** and terminate with */ and that spans multiple lines. (Note that documentation comment starts with /** whereas multi line comment start with /*)
  Javadoc program uses this documentation comments to generate HTML pages of API documentation from Java source code files.

  Documentation comment must come immediately before a class or interface or method or field definition. A documentation comment has two parts – a description followed by block tags.

3.3 Data types and literals

Data
Data is defined as the quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.

Data can exist in a variety of forms -- as numbers or text on pieces of paper, as bits and bytes stored in electronic memory, or as facts stored in a person's mind. Strictly speaking, data is the plural of *datum*, a single piece of information. In practice, however, people use *data* as both the singular and plural form of the word.
All software is divided into two general categories: *data* and *programs*. Programs are collections of instructions for manipulating data.

Data types

A variable's data type indicates what sort of value the variable represents, such as whether it is an integer, a floating-point number, or a character.
There are two data types available in Java:

- Primitive Data Types
- Reference/Object Data Types

Primitive Data Types:

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword. Let us now look into detail about the eight primitive data types.

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 range than int is needed.

- Default value is 0L.
- Example: long a = 100000L, long b = -200000L

float:

- Float data type is a single-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

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 flags that track true/false conditions.
- Default value is false.
- Example: boolean one = true

char:

- char data type is a single 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'

Reference Data Types:

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy etc.
- Class objects and various types of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to any object of the declared type or any compatible type.
- Example: Animal animal = new Animal("giraffe");

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
\'	Single quote
\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

3.4 Operators

An operator is a symbol that represents a specific action. For example, a plus sign (+) is an operator that represents addition. The basic mathematic operators are + addition, - subtraction,* multiplication,/ division

They are used to manipulate primitive data types. Java operators can be classified as unary, binary, or ternary—meaning taking one, two, or three arguments,

respectively. A unary operator may appear before (prefix) its argument or after (postfix) its argument. A binary or ternary operator appears between its arguments.

Operators in java fall into 8 different categories:

Java operators fall into eight different categories: assignment, arithmetic, relational, logical, bitwise, compound assignment, conditional, and type.

```
Assignment Operators    =

Arithmetic Operators    -   +   *   /   %   ++   --

Relational Operators    >  <   >=     <=    ==   !=

Logical Operators       &&    ||    &  |  !   ^

Bit wise Operator       &   |   ^  >>   >>>

Compound Assignment Operators    +=     -=     *=     /=     %=

                          <<= >>= >>>=   Conditional Operator  ?:
```

Java has eight different operator types: assignment, arithmetic, relational, logical, bitwise, compound assignment, conditional, and type.

ASSIGNMENT OPERATORS

The java assignment operator statement has the following syntax:

<variable> = <expression>

Eg x = 10;

If the value already exists in the variable it is overwritten by the assignment operator (=).

ARITHMETIC OPERATORS

Java provides eight Arithmetic operators. They are for addition, subtraction, multiplication, division, modulo (or remainder), increment (or add 1), decrement (or subtract 1), and negation.

The binary operator + is overloaded in the sense that the operation performed is determined by the type of the operands. When one of the operands is a String object, the other operand is implicitly converted to its string representation and string concatenation is performed.

Eg x= y + z;

RELATIONAL OPERATORS

Relational operators in Java are used to compare 2 or more objects. Java provides six relational operators:

greater than (>), less than (<), greater than or equal (>=), less than or equal (<=), equal (==), and not equal (!=).

All relational operators are binary operators, and their operands are numeric expressions.

Binary numeric promotion is applied to the operands of these operators. The evaluation results in a boolean value.

LOGICAL OPERATORS

Logical operators return a true or false value based on the state of the Variables. There are six logical, or boolean, operators. They are AND, conditional AND, OR, conditional OR, exclusive OR, and NOT. Each argument to a logical operator must be a boolean data type, and the result is always a boolean data type.

BITWISE OPERATORS

Java provides Bit wise operators to manipulate the contents of variables at the bit level.
These variables must be of numeric data type (char, short, int, or long). Java provides seven bitwise operators. They are AND, OR, Exclusive-OR, Complement, Left-shift, Signed Right-shift, and Unsigned Right-shift.

COMPOUND OPERATORS

Compound operators perform shortcuts in common programming operations. Java has eleven compound assignment operators.

Syntax:

= argument2.

The above statement is the same as, argument1 = argument1 operator argument2.

CONDITIONAL OPERATORS

The Conditional operator is the only ternary (operator takes three arguments) operator in Java. The operator evaluates the first argument and, if true, evaluates the second argument. If the first argument evaluates to false, then the third argument is evaluated. The conditional operator is the expression equivalent of the if-else statement. The conditional expression can be nested and the conditional operator associates from right to left: (a?b?c?d:e:f:g) evaluates as (a?(b?(c?d:e):f):g)

Type conversion allows a value to be changed from one primitive data type to another. Conversion can occur explicitly, as specified in the program, or implicitly, by Java itself. Java allows both type widening and type narrowing conversions.

In java Conversions can occur by the following ways:

- Using a cast operator (explicit promotion)
- Using an arithmetic operator is used with arguments of different data types (arithmetic promotion)
- A value of one type is assigned to a variable of a different type (assignment promotion)

OPERATOR PRECEDENCE

The order in which operators are applied is known as precedence. Operators with a higher precedence are applied before operators with a lower precedence. The operator precedence order of Java is shown below. Operators at the top of the table are applied before operators lower down in the table. If two operators have the same precedence, they are applied in the order they appear in a statement.

That is, from left to right. You can use parentheses to override the default precedence.

Java Operator Precedence Table

Precedence	Operator	Type	Associability
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (type)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right

7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Larger number means higher precedence.

3.5 Type casting

Assigning a value of one type to a variable of another type is known as Type Casting.

```
 int x = 10;
byte y = (byte)x;
```

Widening or Automatic type conversion

Automatic Type casting take place when,
the two types are compatible
the target type is larger than the source type
Eg

```
int i = 100;
        long l = i;    //no explicit type casting required
```

Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.
Eg

```
double d = 100.04;
     long l = (long)d;  //explicit type casting required
```

3.6 Identifiers

Identifiers are the names of variables, methods, classes, packages and interfaces. Unlike literals they are not the things themselves, just ways of referring to them. E.g. num1, name, etc.

Rules for identifiers:

- They must begin with a letter of the alphabet, an underscore, or (_), or a dollar sign ($). The convention is to always use a letter of the alphabet. After the first initial letter, identifier may also contain letters and the digits 0

to 9. No spaces or special characters are allowed. The dollar sign and the underscore are discouraged.
- They can be of any length
- They are case-sensitive.
- You cannot use a java keyword (reserved word) for an identifier

3.7 Variables and constants

A variable is a symbol or name that stands for a value. For example, in the expression

x+y

x and *y* are variables. Variables can represent numeric values, characters, character strings, or memory addresses.

Variables play an important role in computer programming because they enable programmers to write flexible programs. Rather than entering data directly into a program, a programmer can use variables to represent the data. Then, when the program is executed, the variables are replaced with real data. This makes it possible for the same program to process different sets of data.

Every variable has a name, called the *variable name,* and a data type. The opposite of a *variable* is a constant. Constants are values that never change. Because of their inflexibility, constants are used less often than variables in programming.

Syntax for variable declaration

Data-type identifier; or

Data-type identifier-list;

Example: int x;

Int x,y,z;

Variables cab be initialized (set to an initial value) by the programmer at the point of declaration, using the following syntax:

Variable initialization: data-type identifier = literal;

Eg int height = 10;

The syntax for a constant declaration follows:

Final data-type identifier = literal;

Eg final float TAX = 0.05;

3.8 The anatomy of a simple program

Consider the following template of a java program

```
Heading giving details of the name and purpose of the program

import list

class name

{

main method

        {

                Declaration of constants

                Declaration of variables

                Program statements

        }

}
```

Heading giving details of the name and purpose of the program
It is a set of comments written on as many lines as necessary.
Documentation comment must come immediately before a class or interface or method or field definition. A documentation comment has two parts - a description followed by block tags.

Import list
The import statement makes Java classes available to the program. You can specify each class in the import statement, for example avi.DialogBox and avi.Window, but it is a lot simpler to use an asterisk as wildcard to make all classes of the package avi available; hence the statement avi.* that is used in many of the program in this manual.
A wildcard is a character that can represent a number of different characters. The wildcard * may represent any of the class names.
Note: the java.lang package is automatically imported; therefore, there is never any need to include it in the import list.

Class name
The name of the class containing the main method must be the same as the name given to the program file (omitting the .java suffix). The naming of the class must follow the same rules as for the naming of any other identifier. The use of braces {} indicates the beginning and ending of the class.
Note: the Java convention also dictates that the name of a class should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.

Main method

In the Java language, when you execute a class with the Java interpreter, the runtime system starts by calling the class's main() method. The main() method then calls all the other methods required to run your application

Exercises

1. If semicolons are needed at the end of each statement, why does the comment line // My first Java program goes here not end with a semicolon?
2. How important is it to put the right number of blank spaces on a line in a Java program?
3. I have several word processing programs on my system. Which should I use to write Java programs?
4. What is a character?
5. I couldn't find any errors in the line where the compiler noted an error. What can I do?

Chapter 4. Classes and Objects

4.1 Class declaration

A class declaration names the class and encloses the class body between braces.

A class declaration syntax:

```
class MyClass extends MySuperClass implements YourInterface {
   // field, constructor, and
   // method declarations
}
```

Example of class declaration

```
class Circle
{
        private double radius;
        public static double PI = 3.14;
        public Circle(double r)
        {
                radius = r;
        }
        public double getRadius()
        {
                return radius;
        }
        public void setRadius(double r)
        {
                radius = r;
        }
        public double circumference()
        {
                return 2 * PI * radius;
        }
        public double area()
        {
                return PI * radius* radius;
        }
        public static void main(String args[])
        {
                Circle c = new Circle(10);
                System.out.println(c.area());
        }
}
```

In general, class declarations can include these components, in order:

1. Modifiers such as public, private, and a number of others that you will encounter later.
2. The class name, with the initial letter capitalized by convention.

3. The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. The class body, surrounded by braces, {}.

The class body contains fields, methods, and constructors for the class. A class uses fields to contain state information and uses methods to implement behavior.

4.2 Member Variables Declaration

There are several kinds of variables:

- Member variables in a class—these are called fields.
- Variables in a method or block of code—these are called local variables.
- Variables in method declarations—these are called parameters.

The circle class uses the following line of code to define its field:
private double radius;

Field declarations are composed of three components, in order:

1. Zero or more modifiers, such as public or private.
2. The field's type.
3. The field's name.

The fields of circle is named radius and is of data type double. The public keyword identifies these fields as public members, accessible by any object that can access the class.

Types

All variables must have a type. You can use primitive types such as int, float, boolean, etc. Or you can use reference types, such as strings, arrays, or objects.

Variable Names

All variables, whether they are fields, local variables, or parameters, follow the same naming rules and conventions. That is; -

- the first letter of a class name should be capitalized, and
- the first (or only) word in a method name should be a verb.

4.3 Defining Methods

Here is an example of a typical method declaration:

```
public double calculateAnswer(double wingSpan, int numberOfEngines,
                              double length, double grossTons) {
   //do the calculation here
}
```

The only required elements of a method declaration are the method's return type, name, a pair of parentheses, (), and a body between braces, {}.
More generally, method declarations have six components, in order:

1. Modifiers—such as public, private, and others you will learn about later.
2. The return type—the data type of the value returned by the method, or void if the method does not return a value.
3. The method name—the rules for field names apply to method names as well, but the convention is a little different.
4. The parameter list in parenthesis—a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, (). If there are no parameters, you must use empty parentheses.
5. An exception list—to be discussed later.
6. The method body, enclosed between braces—the method's code, including the declaration of local variables, goes here.

Definition: Two of the components of a method declaration comprise the method signature—the method's name and the parameter types.

The signature of the method declared above is:

calculateAnswer(double, int, double, double)

Naming a Method

By convention, method names should be a verb in lowercase or a multi-word name that begins with a verb in lowercase, followed by adjectives, nouns, etc. In multi-word names, the first letter of each of the second and following words should be capitalized. Here are some examples:

run
runFast
getFinalData

You specify a class variable or a class method by using the static keyword in the member's declaration. A member that is not declared as static is implicitly an instance member. Class variables are shared by all instances of a class and can be accessed through the class name as well as an instance reference. Instances of a class get their own copy of each instance variable, which must be accessed through an instance reference.

Instance variables and methods that are accessible to code outside of the class that they are declared in can be referred to by using a qualified name. The qualified name of an instance variable looks like this:

objectReference.variableName

The qualified name of a method looks like this:

objectReference.methodName(argumentList)

or:

objectReference.methodName()

4.4 Constructors

Constructors that initialize a new instance of a class use the name of the class and look like methods without a return type.

For example, circle has one constructor:

```
public Circle(double r)
        {
                radius = r;
        }
```

To create a new circle object called myCircle, a constructor is called by the new operator:

```
Circle myCircle = new Circle(10);
```

new Circle(10) creates space in memory for the object and initializes its fields.

Although circle only has one constructor, it could have others, including a no-argument constructor.

Both constructors could have been declared in circle because they have different argument lists. As with methods, the Java platform differentiates constructors on the basis of the number of arguments in the list and their types.

You don't have to provide any constructors for your class, but you must be careful when doing this. The compiler automatically provides a no-argument, default constructor for any class without constructors. This default constructor will call the no-argument constructor of the superclass. In this situation, the compiler will complain if the superclass doesn't have a no-argument constructor so you must verify that it does. If your class has no explicit superclass, then it has an implicit superclass of Object, which does have a no-argument constructor.

You can use a superclass constructor yourself. The MountainBike class at the beginning of this lesson did just that. This will be discussed later, in the lesson on interfaces and inheritance.

You can use access modifiers in a constructor's declaration to control which other classes can call the constructor.

Note: If another class cannot call a Circle constructor, it cannot directly create Circle objects.

You create an object from a class by using the new operator and a constructor. The new operator returns a reference to the object that was created. You can assign the reference to a variable or use it directly.

4.5 Access Modifiers

You control access to classes and members in the same way: by using an access modifier such as public in their declaration.

The first (left-most) modifier used lets you control what other classes have access to a member field. For the moment, consider only public and private. Other access modifiers will be discussed later.

- public modifier—the field is accessible from all classes.
- private modifier—the field is accessible only within its own class.

In the spirit of encapsulation, it is common to make fields private. This means that they can only be directly accessed from the circle class. We still need access to these values, however. This can be done indirectly by using public methods that obtain the field values for us. For example, the getRadius() method.

4.6 The Garbage Collector

The garbage collector automatically cleans up unused objects. An object is unused if the program holds no more references to it. You can explicitly drop a reference by setting the variable holding the reference to null.

4.7 Packages and class library

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:

While creating a package, you should choose a name for the package and include a package statement along 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 placed in the current default package.

To compile the Java programs with package statements you have to do use -d option as shown below.

javac -d Destination_folder file_name.java

Then a folder with the given package name is created in the specified destination, and the compiled class files will be placed in that folder

Example

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

Below given package example contains interface named animals:

```
/* File name : Animal.java */
package animals;
interface Animal {
   public void eat();
   public void travel();
}
```

In object-oriented programming , a class library is a collection of prewritten classes or coded templates, any of which can be specified and used by a programmer when developing an application program. The programmer specifies which classes are being used and furnishes data that instantiates each class as an object that can be called when the program is executed. Access to and use of a class library greatly simplifies the job of the programmer since standard, pretested code is available that the programmer doesn't have to write.

A class library is analogous to a subroutine library in earlier, procedural programming.

Advantages of Library Classes in Java:

- The type constraints are used to control where the java library classes can be replaced with routine versions without affecting type perfection of programs.

- Static analysis is then used to control those applicants for which unused library functionality and synchronization can be removed safely from the allocated types.
- The profile data is collected about the usage features of the customization candidates to determine where the allocation of custom library classes is likely to be cost-effective.
- To base on the static analysis results and the profiling information the custom library classes are automatically generated from a template.
- The byte code of the client application is rewritten to use the generated custom classes. This byte code rewriting is completely see-through to the programmer.

List of Library Classes in Java:

Library classes	Purpose of the class
Java.io	Use for input and output functions.
Java.lang	Use for character and string operation.
Java.awt	Use for windows interface.
Java.util	Use for develop utility programming.
Java.applet	Use for applet.
Java.net	Used for network communication.
Java.math	Used for various mathematical calculations like power, square root etc.

Exercises

1. Can constructor methods send back a value like other methods?
2. What is the difference between a local variable and a field?
3. Explain the purpose of a method parameter. What is the difference between a parameter and an argument?
4. What's the purpose of keyword new? Explain what happens when you use it.
5. Create a class called Date that includes three instance variables—a month (type int), a day (type int) and a year (type int). Provide a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set and a get method for each instance variable. Provide a method displayDate that displays the month, day and year separated by forward slashes (/). Write a test application named DateTest that demonstrates class Date's capabilities

Chapter 5: Control structures

5.1. Introduction

We have mentioned that Java programs contain classes and that these classes contain methods which contain statements that are executed by the computer. Now we look at ways of putting statements together using control structures to organize the execution of the program. A control structure might cause a statement to be executed once, several times, or not at all. Control structures make up some of the statements of the Java language. Statements are said to be sequentially composed when they are written one after the other. Sequentially composed statements can be grouped together into a block by using the left brace and right brace symbols ("{" and "}") to bracket them.

5.2. Conditional statements

A conditional statement allows a choice from a selection of statements. It first evaluates an expression to decide between the possibilities. If there are only two then a Boolean- valued expression (also called a logical expression) will be enough to allow us to choose. These can be for med using the ==operator (equal to) or the != operator (not equal to). When comparing numeric values we could use the relational operators <(less than), >
(greater than), <=(less than or equal to) or >=(greater than or equal to). A conditional statement uses the keywords if and else to mark the beginning of the conditional statement and to separate the two sub-statements. The first sub-statement is to be executed if the expression in the condition evaluates to true. The second sub-statement is to be executed if the expression in the condition evaluates to false. These two sub-statements are referred to as the then-statement and the else- statement respectively. It is possible for a conditional statement not to have an else-clause. Java has both an if-then statement and an if-then-else statement. We will consider some examples.

Statement	**effect**
if(x == 0) System.out.println("zero");	Prints "zero" if x has the value 0.
if(x == 0) System.out.println("zero"); if(x != 0) System.out.println("non-zero");	Prints "zero" if x has the value 0, Prints "non-zero" otherwise.
if(x == 0) System.out.println("zero"); else System.out.println("non-zero");	Prints "zero" if x has the value 0, prints "non-zero" otherwise

if(x == 0)
System.out.println("zero");
else
if(x > 0)
System.out.println("positive");
else
System.out.println("negative");
	Prints "zero" if x has the value 0,
	prints "positive" if x is positive,
	prints "negative" if x is negative.

if(x != 0)
if(x > 0)
System.out.println("positive");
	Prints "positive" if x is positive,
	and non-zero. Prints " negative"
	if x is negative

else. System.out.println("negative");	

All of the conditional statements which we have seen have just a single statement as the then-statement or the else-statement. When we need to perform two actions in some case then we need to bracket them together with left and right braces. Without these the effect is quite different.

Statement	**effect**

if(x < 0){
System.out.println("negative");
}
x *= -1;
	Changes the sign of x and prints Negative if x is negative.

if(x < 0)
System.out.println("negative");
x *= -1;
	Prints negative if x is negative.
	Changes the sign of x whether it was negative or not.

When for matting computer programs we will normally use blank-space indentation to convey hints to the reader about statement structure. However, as far as the compiler for the Java language is concerned, one blank space is as good as ten so the different effect of the two statements is achieved by the use of the left and right braces, and not by the blank-space indentation at the start of the line.

5.3. The switch statement

A conditional statement contains a Boolean valued expression. Sometimes the expression which we need to examine is an integer or a character. In this circumstance we can use a switch statement. The switch statement introduces four new keywords, switch, case, default and break. The following example is very similar to the second and third examples of conditional statements which we saw.

Statement	**Effect**
switch(x){ case0: System.out.println ("zero"); break; default: System.out.println ("non-zero"); break; }	Prints “zero” if x has the value 0, prints “non-zero” otherwise.

Without the break statements the flow of control falls through the statement so that the first matching statement is executed and then all of the statements which follow it. Since this is rarely useful, a switch statement almost always has a break statement at the end of each case.
In the example shown above we have only one case treated specially in the switch. We could have more. The if-then-else statement allows to choose between two possible sub-statements but the switch allows us to choose between any number of them. The other cases also appear in the body of the switch statement with the default case coming at the end.

5.4. Contrasting “switch” with “if-then-else”

The switch and conditional statements perform related tasks, but they are not interchangeable. One reason for this is that Java does not allow the programmer to write boolean literals as expressions on the limbs of the switch. This means that although we can think of a conditional statement as being like a switch on the value of a boolean expression, we cannot express this in Java.
Supporting this separation between “switch” and “if-then-else” is the fact that Java treats boolean values as different from integer values. A boolean variable in Java can hold only the value true or the value false. An integer variable can hold negative or positive integers.

5.5. Conditional ? : Operator

The conditional operator ? : can be used to replace if...else statements. It has the following general form:
Exp1 ? Exp2 : Exp3;
Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.
To determine the value of whole expression, initially exp1 is evaluated

- If the value of exp1 is true, then the value of Exp2 will be the value of the whole expression.
- If the value of exp1 is false, then Exp3 is evaluated and its value becomes the value of the entire expression.

The ?: is called a ternary operator because it requires three operands and can be used to replace if-else statements, which have the following form:

```
if(condition){
   var = X;
}else{
   var = Y;
}
```

For example, consider the following code:

```
if(y < 10){
   var = 30;
}else{
   var = 40;
}
```

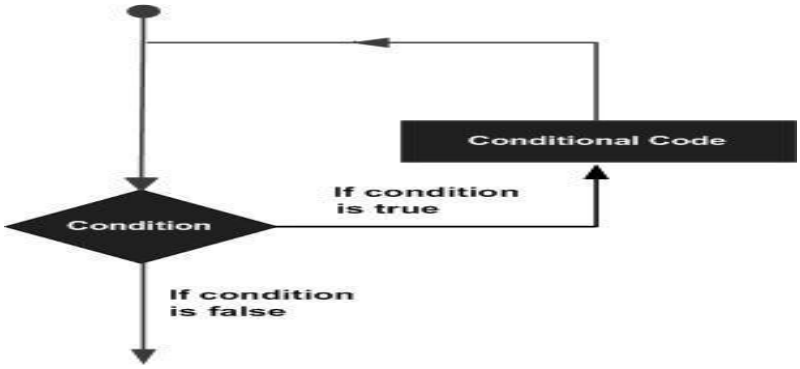
Above code can be rewritten like this:

```
var = (y < 10) ? 30 : 40;
```

Here, var is assigned the value of 30 if y is less than 10 and 40 if it is not.

5.6. Iteration

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:



Java programming language provides the following types of loop to handle looping requirements.

5.7. While loop

A while loop statement in java programming language repeatedly executes a target statement as long as a given condition is true. It tests the condition before executing the loop body.

Syntax:

The syntax of a while loop is:

```
while(Boolean_expression)
{
  //Statements
}
```

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non zero value.

When executing, if the boolean_expression result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true. When the condition becomes false, program control passes to the line immediately following the loop

Here, key point of the while loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example:

```
public class Test {

  public static void main(String args[]) {
    int x = 10;

    while( x < 20 ) {
      System.out.print("value of x : " + x );
```

```
        x++;
        System.out.print("\n");
      }
    }
}
```

This would produce the following result:

value of x : 10

value of x : 11

value of x : 12

value of x : 13

value of x : 14

value of x : 15

value of x : 16

value of x : 17

value of x : 18

value of x : 19

5.8. For loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. A for loop is useful when you know how many times a task is to be repeated.

Syntax:

The syntax of a for loop is:

```
for(initialization; Boolean_expression; update)
{
  //Statements
}
```

Here is the flow of control in a for loop:

- The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. and this step ends with a semi colon (;)
- Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop will not be executed and control jumps to the next statement past the for loop.

- After the body of the for loop gets executed, the control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank with a semicolon at the end.
- The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

Example:

Below given is the example code of the for loop in java

```
public class Test {

   public static void main(String args[]) {

      for(int x = 10; x < 20; x = x+1) {
         System.out.print("value of x : " + x );
         System.out.print("\n");
      }
   }
}
```

This would produce the following result:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

Enhanced for loop in Java

As of Java 5, the enhanced for loop was introduced. This is mainly used to traverse collection of elements including arrays.

Syntax:

The syntax of enhanced for loop is:

```
for(declaration : expression)
{
   //Statements
}
```

- **Declaration**: The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.

- **Expression**: This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Example:

```
public class Test {

   public static void main(String args[]){
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ){
         System.out.print( x );
         System.out.print(",");
      }
      System.out.print("\n");
      String [] names ={"James", "Larry", "Tom", "Lacy"};
      for( String name : names ) {
         System.out.print( name );
         System.out.print(",");
      }
   }
}
```

This would produce the following result:

```
10,20,30,40,50,
James,Larry,Tom,Lacy,
```

5.9. Do while loop

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time. Like a while statement, except that it tests the condition at the end of the loop body

Syntax:

The syntax of a do...while loop is:

```
do
{
   //Statements
}while(Boolean_expression);
```

Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the Boolean expression is true, the control jumps back up to do statement, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

Example:

```
public class Test {
```

```
   public static void main(String args[]){
      int x = 10;

      do{
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }while( x < 20 );
   }
}
```

This would produce the following result:

value of x : 10

value of x : 11

value of x : 12

value of x : 13

value of x : 14

value of x : 15

value of x : 16

value of x : 17

value of x : 18

value of x : 19

5.10. Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Java supports the following control statements.

Control Statement	Description
break statement	Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

5.11. Arrays

Arrays are a way to store a list of items. Arrays can contain any type of value (base types or objects), but you can't store different types in a single array.To create an array in Java, you use three steps:

- Declare a variable to hold the array.
- Create a new array object and assign it to the array variable.
- Store things in that array.

Declaring Array Variables

Array variables indicate the type of object the array will hold (just as they do for any variable) and the name of the array, followed by empty brackets ([])

```
int temps[]; OR
int[] temps;
```

Creating Array Objects

The second step is to create an array object and assign it to that variable. There are two ways to do this:

- The first way is to use the new operator to create a new instance of an array:

  ```
  String[] names = new String[10];
  ```

  When you create an array object using new, all its elements are initialized for you (0 for numeric arrays, false for boolean, '\0' for character arrays, and null for everything else)
- You can also create and initialize an array at the same time. Instead of using new to create the new array object, enclose the elements of the array inside braces, separated by commas:

  ```
  String[] chiles = { "jalapeno", "anaheim", "serrano," "habanero," "thai" }
  ```

  An array the size of the number of elements you've included will be automatically created for you.

Accessing Array Elements

To get at a value stored within an array, use the array subscript expression:

```
myArray[subscript];
```

The myArray part of this expression is a variable holding an array object. The subscript is the slot within the array to access. Array subscripts start with 0.
You can test for the length of the array in your programs using the length instance variable - it's available for all array objects, regardless of type

Changing Array Elements

To assign a value to a particular array slot, merely put an assignment statement after the array access expression:

```
myarray[1] = 15;
```

Multidimensional Arrays

Java does not support multidimensional arrays. However, you can declare and create an array of arrays, and access them as you would C-style multidimensional arrays:

```
int coords[][] = new int[12][12];
coords[0][0] = 1;
coords[0][1] = 2;
```

the following example demonstrate arrays of arrays implementation

```
class MultiDimArrayDemo {
   public static void main(String[] args) {
      String[][] names = {
         {"Mr. ", "Mrs. ", "Ms. "},
         {"Smith", "Jones"}
      };
```

```
        // Mr. Smith
        System.out.println(names[0][0] + names[1][0]);
        // Ms. Jones
        System.out.println(names[0][2] + names[1][1]);
    }
}
```

The output from this program is:
Mr. Smith
Ms. Jones

Exercises

1. Write a program that computes the product of two square matrices of boolean values
2. Write a code fragment that prints the contents of a two-dimensional boolean array, using * to represent true and a space to represent false. Include row and column numbers.
3. What pitfalls should I watch out for when using arrays?
4. Factorials are used frequently in probability problems. The factorial of a positive integer n (written n! and pronounced "n factorial") is equal to the product of the positive integers from 1 to n. Write an application that calculates the factorials of 1 through 20. Use type long. Display the results in tabular format. What difficulty might prevent you from calculating the factorial of 100?
5. If a loop never ends, how does the program stop running?

Chapter 6. Polymorphism

6.1 What is polymorphism in programming?

Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations. I know it sounds confusing. Don't worry we will discuss this in detail.

- It is a feature that allows one interface to be used for a general class of actions.
- An operation may exhibit different behavior in different instances.
- The behavior depends on the types of data used in the operation.
- It plays an important role in allowing objects having different internal structures to share the same external interface.
- Polymorphism is extensively used in implementing inheritance.

6.2 Types of polymorphism

Following concepts demonstrate different types of polymorphism in java.
1) Method Overloading
2) Method Overriding

6.2.1 Method Overloading

In Java, it is possible to define two or more methods of same name in a class, provided that their argument list or parameters are different. This concept is known as Method Overloading.

1. To call an overloaded method in Java, it is must to use the type and/or number of arguments to determine which version of the overloaded method to actually call.
2. Overloaded methods may have different return types; the return type alone is insufficient to distinguish two versions of a method. .
3. When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call.
4. It allows the user to achieve compile time polymorphism.
5. An overloaded method can throw different exceptions.
6. It can have different access modifiers.

Example:

```
class Overload
{
   void demo (int a)
   {
     System.out.println ("a: " + a);
   }
   void demo (int a, int b)
   {
     System.out.println ("a and b: " + a + "," + b);
```

```
    }
    double demo(double a) {
      System.out.println("double a: " + a);
      return a*a;
    }
}
class MethodOverloading
{
    public static void main (String args [])
    {
        Overload Obj = new Overload();
        double result;
        Obj .demo(10);
        Obj .demo(10, 20);
        result = Obj .demo(5.5);
        System.out.println("O/P : " + result);
    }
}
```

Here the method demo() is overloaded 3 times: first having 1 int parameter, second one has 2 int parameters and third one is having double arg. The methods are invoked or called with the same type and number of parameters used.

Output:

```
a: 10
a and b: 10,20
double a: 5.5
O/P : 30.25
```

Rules for Method Overloading

1. Overloading can take place in the same class or in its sub-class.
2. Constructor in Java can be overloaded
3. Overloaded methods must have a different argument list.
4. Overloaded method (can also take place in sub class), with same name but different parameters.
5. The parameters may differ in their type or number, or in both.
6. They may have the same or different return types.
7. It is also known as compile time polymorphism.

6.2.2 Method Overriding

Child class has the same method as of base class. In such cases child class overrides the parent class method without even touching the source code of the base class. This feature is known as method overriding.

Example:

```
public class BaseClass
{
    public void methodToOverride() //Base class method
    {
        System.out.println ("I'm the method of BaseClass");
```

```
    }
}
public class DerivedClass extends BaseClass
{
    public void methodToOverride() //Derived Class method
    {
        System.out.println ("I'm the method of DerivedClass");
    }
}

public class TestMethod
{
    public static void main (String args []) {
      // BaseClass reference and object
      BaseClass obj1 = new BaseClass();
      // BaseClass reference but DerivedClass object
      BaseClass obj2 = new DerivedClass();
      // Calls the method from BaseClass class
      obj1.methodToOverride();
      //Calls the method from DerivedClass class
      obj2.methodToOverride();
    }
}
```

Output:

```
I'm the method of BaseClass
I'm the method of DerivedClass
```

Rules for Method Overriding:

1. applies only to inherited methods
2. object type (NOT reference variable type) determines which overridden method will be used at runtime
3. Overriding method can have different return type
4. Overriding method must not have more restrictive access modifier
5. Abstract methods must be overridden
6. Static and final methods cannot be overridden
7. Constructors cannot be overridden
8. It is also known as Runtime polymorphism.

Super Keyword in Overriding:

When invoking a superclass version of an overridden method the super keyword is used.
Example:

```
class Vehicle {
    public void move () {
        System.out.println ("Vehicles are used for moving from one place to another
");
    }
}
```

```
class Car extends Vehicle {
   public void move () {
     super. move (); // invokes the super class method
     System.out.println ("Car is a good medium of transport ");
   }
}

public class TestCar {
   public static void main (String args []){
      Vehicle b = new Car (); // Vehicle reference but Car object
      b.move (); //Calls the method in Car class
   }
}
```

Output:

Vehicles are used for moving from one place to another
Car is a good medium of transport

6.3 Overloading vs Overriding in Java

1. Overloading happens at compile-time while Overriding happens at runtime: The binding of overloaded method call to its definition has happens at compile-time however binding of overridden method call to its definition happens at runtime.
2. Static methods can be overloaded which means a class can have more than one static method of same name. Static methods cannot be overridden, even if you declare a same static method in child class it has nothing to do with the same method of parent class.
3. The most basic difference is that overloading is being done in the same class while for overriding base and child classes are required. Overriding is all about giving a specific implementation to the inherited method of parent class.
4. Static binding is being used for overloaded methods and dynamic binding is being used for overridden/overriding methods.
5. Performance: Overloading gives better performance compared to overriding. The reason is that the binding of overridden methods is being done at runtime.
6. private and final methods can be overloaded but they cannot be overridden. It means a class can have more than one private/final methods of same name but a child class cannot override the private/final methods of their base class.
7. Return type of method does not matter in case of method overloading; it can be same or different. However, in case of method overriding the overriding method can have more specific return type.
8. Argument list should be different while doing method overloading. Argument list should be same in method Overriding.

Exercises

1. What is polymorphism?
2. Name types of polymorphism in java
3. What is method overloading
4. Differentiate method overloading from method overriding

Chapter 7. Abstraction

7.1 Abstraction in Java

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
2. Interface

7.2 Abstract class in Java

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.

Example of abstract class

```
abstract class A {}
```

abstract method

Example abstract method

```
abstract void printStatus();//no body and followed by a semicolon
```

Example of abstract class that has abstract method
In this example, Teacher the abstract class that contains only one abstract method run. Its implementation is provided by the class University.

```
abstract class Teacher{
  abstract void run();
 }

 class University extends Teacher{
 void run(){System.out.println("I am a Tutor!!..");}

 public static void main(String args[]){
  Teacher obj = new University();
  obj.run();
 }
 }
```

Compile by: javac University.java
Run by: java University

Declaring a method as abstract has two results:
• The class must also be declared abstract. If a class contains an abstract method, the class must be abstract as well.
• Any child class must either override the abstract method or declare itself abstract.

A child class that inherits an abstract method must override it. If they do not, they must be abstract and any of their children must override it.

Eventually, a descendant class has to implement the abstract method; otherwise, you would have a hierarchy of abstract classes that cannot be instantiated.

Rule: If there is any abstract method in a class, that class must be abstract.
Rule: If you are extending any abstract class that has abstract method, you must either provide the implementation of the method or make this class abstract.

7.3 Interface in Java

An interface in java is a blueprint of a class. It has static constants and abstract methods only.
The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.
Java Interface also represents IS-A relationship.
It cannot be instantiated just like abstract class.

Why use Java interface?

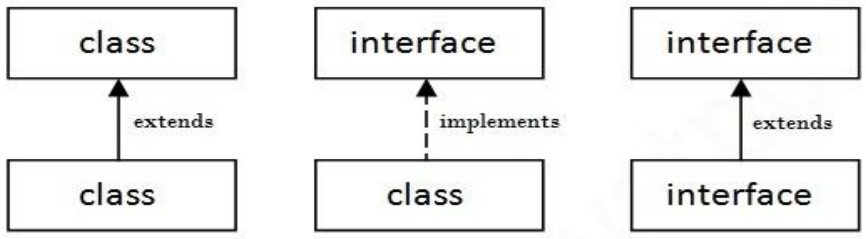
There are mainly three reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

In other words, Interface fields are public, static and final by default, and methods are public and abstract.

Understanding relationship between classes and interfaces
As shown in the figure given below, a class extends another class, an interface extends another interface but a class implements an interface.



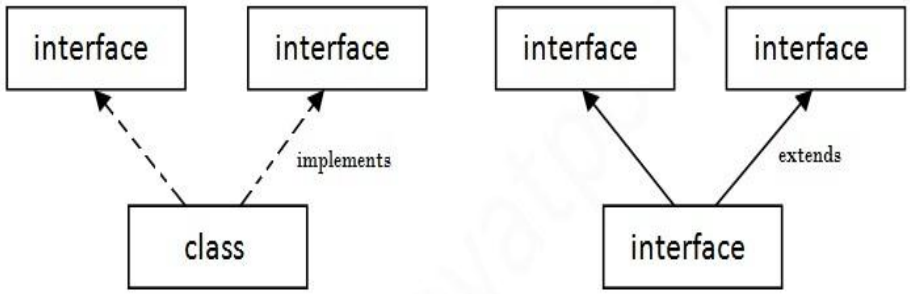
Simple example of java Interface

In this example, Vigo interface have only one method, its implementation is provided in the Test class.

```
interface Vigo
{
	void printme();
}
class Tutor implements Vigo
{
	public void printme()
	{
		System.out.println("I am a Turor");
	}

}
public class Test
{
	public static void main(String args[])
	{
		Tutor t = new Tutor();
			t.printme();
	}
}
```

7.4 Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



Multiple Inheritance in Java

In the following example Tutor class inherits printme() and status() behaviours from Vigo and Vertigo classes respectively.

```
interface Vigo
{
	void printme();
}
interface Vertigo
{
```

```
        void status();
}
class Tutor implements Vigo,Vertigo
{
        public void printme()
        {
                System.out.println("I am a Turor");
        }
        public void status()
        {
                System.out.println("I am alive");
        }

}
public class Test
{
        public static void main(String args[])
        {
                Tutor t = new Tutor();
                        t.printme();
                        t.status();
        }
}
```

Interface inheritance

A class implements interface but one interface extends another interface.

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
```

Exercises

1. What is abstraction?
2. How do we achieve data abstraction in java?
3. Differentiate an interface from an abstract class?
4. Define the following
    a. Interface
    b. Abstract class

Chapter 8. Encapsulation

8.1 Encapsulation in Java

Encapsulation in java is a process of wrapping code and data together into a single unit, for example capsule i.e. mixed of several medicines.

Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding.

Encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. Access to the data and code is tightly controlled by an interface.

The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code. With this feature Encapsulation gives maintainability, flexibility and extensibility to our code.
The Java Bean class is the example of fully encapsulated class.

Simple example of encapsulation in java

Let's see the simple example of encapsulation that has only one field with its setter and getter methods.

```
//save as Student.java
public class Student{
private String name;
public String getName(){
return name;
}
public void setName(String name){
this.name=name
}
}

//save as Test.java
class Test{
public static void main(String[] args){
Student s=new Student();
s.setname("vijay");
System.out.println(s.getName());
}
}
```

The public methods are the access points to this class' fields from the outside java world. Normally, these methods are referred as getters and setters. Therefore any class that wants to access the variables should access them through these getters and setters.

8.2 Benefits of Encapsulation

The fields of a class can be made read-only or write-only. A class can have total control over what is stored in its fields. The users of a class do not know how the class stores its data. A class can change the data type of a field and users of the class do not need to change any of their code

Exercises

1. What is encapsulation
2. What are the benefits of encapsulation
3. Give an example of encapsulation

Chapter 9: Inheritance

9.1. Inheritance

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the IS-A relationship, also known as parent-child relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

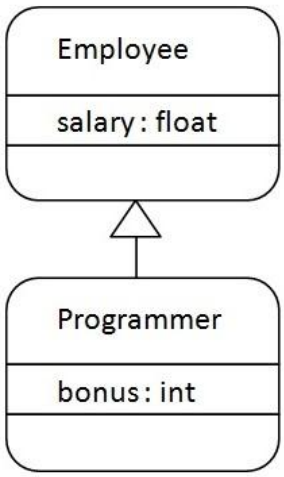
Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class.

In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

Understanding the simple example of inheritance



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.

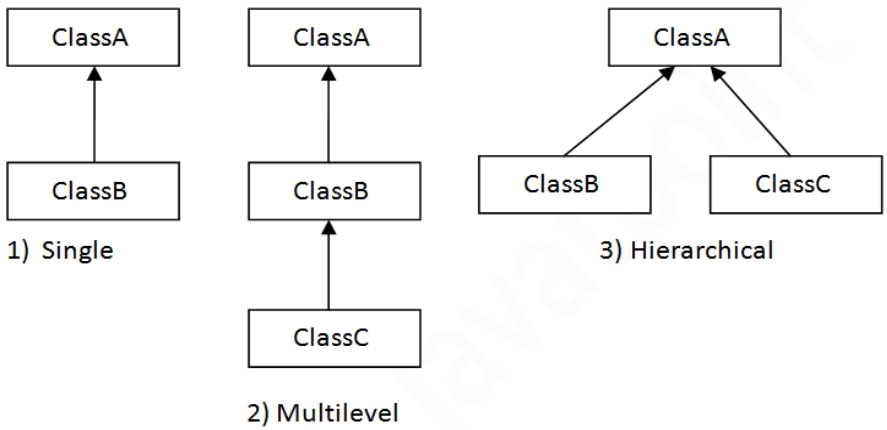
```
class Employee{

 float salary=40000;

}

class Programmer extends Employee{

 int bonus=10000;

 public static void main(String args[]){

 Programmer p=new Programmer();

 System.out.println("Programmer salary is:"+p.salary);

  System.out.println("Bonus of Programmer is:"+p.bonus);

}

}
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

9.2. Types of inheritance in java

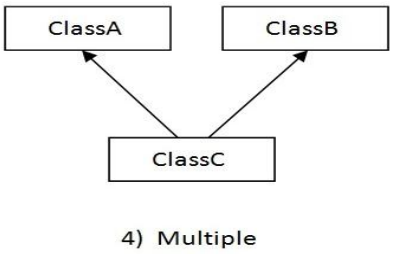
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only.

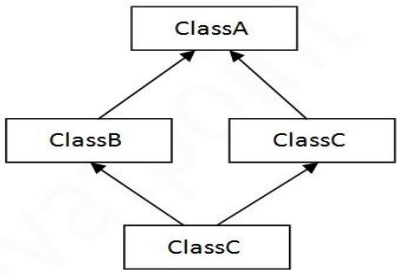


Note: Multiple inheritance is not supported in java through class.

When a class extends multiple classes i.e. known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

.

9.3. Why multiple inheritances are not supported in java?

To reduce the complexity and simplify the language, multiple inheritances are not supported in java.

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were
 Public Static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Exercises

1. Can a class have more than one superclass so that it inherits additional methods and behavior?
2. What is inheritance
3. Can a child always inherit properties from its parent?
4. Explain different types of inheritance in java.

Chapter 10. Exception handling

10.1 Java Exception Handling

Exception is an error event that can happen during the execution of a program and disrupts its normal flow. Java provides a robust and object oriented way to handle exception scenarios, known as **Java Exception Handling**.

Java Exception Handling Overview
We don't like exceptions but we always have to deal with them, great news is that Java Exception handling framework is very robust and easy to understand and use. Exception can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure, Database server down etc. In this section, we will learn how exceptions are handled in java.

Java being an object oriented programming language, whenever an error occurs while executing a statement, creates an **exception object** and then the normal flow of the program halts and JRE tries to find someone that can handle the raised exception. The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurred, type of exception etc. When the exception occurs in a method, the process of creating the exception object and handing it over to runtime environment is called **"throwing the exception"**.

Once runtime receives the exception object, it tries to find the handler for the exception. Exception Handler is the block of code that can process the exception object. The logic to find the exception handler is simple – starting the search in the method where error occurred, if no appropriate handler found, then move to the caller method and so on. So if methods call stack is A->B->C and exception is raised in method C, then the search for appropriate handler will move from C->B->A. If appropriate exception handler is found, exception object is passed to the handler to process it. The handler is said to be **"catching the exception"**. If there are no appropriate exception handler found, then program terminates printing information about the exception.

Note that Java Exception handling is a framework that is used to handle runtime errors only, compile time errors are not handled by exception handling framework.

Specific keywords in java program are used to create an exception handler block; The keywords are shown in the following section.

10.2 Exception Handling Keywords

Java provides specific keywords for exception handling purposes, we will look after them first and then we will write a simple program showing how to use them for exception handling.

1. **throw** – We know that if any exception occurs, an exception object is getting created and then Java runtime starts processing to handle them. Sometime we might want to generate exception explicitly in our code, for example in a user authentication program we should throw exception to client if the

password is null. **throw** keyword is used to throw exception to the runtime to handle it.

2. **throws** – When we are throwing any exception in a method and not handling it, then we need to use **throws** keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to its caller method using throws keyword. We can provide multiple exceptions in the throws clause and it can be used with main() method also.
3. **try-catch** – We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.
4. **finally** – finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. finally block gets executed always, whether exception occurred or not.

Let's see a simple program showing exception handling in java.

```
public class NewSumNumbers {
public static void main(String[] arguments) {
      float sum = 0;
      for (int i = 0; i < arguments.length; i++) {
          try {
             sum = sum + Float.parseFloat(arguments[i]);
          } catch (NumberFormatException e) {
             System.out.println(arguments[i] + " is not a number.");
          }
       }
      System.out.println("Those numbers add up to " + sum);
   }
}
```

After you save and compile the application, run it with a non-numeric command-line argument along with a few numbers, such as the following:
java NewSumNumbers 1 3 6x
Running it with these arguments produces the following output:
6x is not a number

Those numbers add up to 4.0
The try-catch block deals with NumberFormatException errors that are thrown by the Float.parseFloat() method. These exceptions are caught within the NewSumNumbers class, which displays an error message for any argument that is not a number.

Because the exception is handled within the class, the Java interpreter does not display an error message when something like 6x is used as a command-line argument.

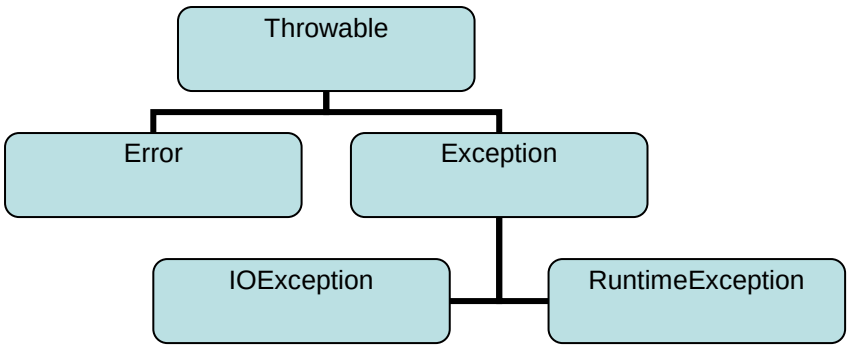
You can often deal with problems related to user input and other unexpected data by using try-catch blocks.

- We can't have catch or finally clause without a try statement.
- A try statement should have either catch block or finally block, it can have both blocks.
- We can't write any code between try-catch-finally block.
- We can have multiple catch blocks with a single try statement.
- try-catch blocks can be nested similar to if-else statements.
- We can have only one finally block with a try-catch statement.

10.3 Exception Hierarchy

As stated earlier, when any exception is raised an **exception object** is getting created. Java Exceptions are hierarchical and inheritance is used to categorize different types of exceptions. Throwable is the parent class of Java Exceptions Hierarchy and it has two child objects – Error and Exception. Exceptions are further divided into checked exceptions and runtime exception.

1. Errors: Errors are exceptional scenarios that are out of scope of application and it's not possible to anticipate and recover from them, for example hardware failure, JVM crash or out of memory error. That's why we have a separate hierarchy of errors and we should not try to handle these situations. Some of the common Errors are OutOfMemoryError and StackOverflowError.
2. **Checked Exceptions**: Checked Exceptions are exceptional scenarios that we can anticipate in a program and try to recover from it, for example FileNotFoundException. We should catch this exception and provide useful message to user and log it properly for debugging purpose. Exception is the parent class of all Checked Exceptions and if we are throwing a checked exception, we must catch it in the same method or we have to propagate it to the caller using throws keyword.
3. **Runtime Exception**: Runtime Exceptions are cause by bad programming, for example trying to retrieve an element from the Array. We should check the length of array first before trying to retrieve the element otherwise it might throw ArrayIndexOutOfBoundException at runtime. RuntimeException is the parent class of all runtime exceptions. If we are throwing any runtime exception in a method, it's not required to specify them in the method signature throws clause. Runtime exceptions can be avoided with better programming.



10.4 Useful Exception Methods

Exception and all of its subclasses doesn't provide any specific methods and all of the methods are defined in the base class Throwable. The exception classes are created to specify different kind of exception scenarios so that we can easily identify the root cause and handle the exception according to its type. Throwable class implements Serializable interface for interoperability.

Some of the useful methods of Throwable class are;

1. **public String getMessage()** – This method returns the message String of Throwable and the message can be provided while creating the exception through its constructor.
2. **public String getLocalizedMessage()** – This method is provided so that subclasses can override it to provide locale specific message to the calling program. Throwable class implementation of this method simply use getMessage() method to return the exception message.
3. **public synchronized Throwable getCause()** – This method returns the cause of the exception or null id the cause is unknown.
4. **public String toString()** – This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.
5. **public void printStackTrace()** – This method prints the stack trace information to the standard error stream, this method is overloaded and we can pass PrintStream or PrintWriter as argument to write the stack trace information to the file or stream.

10.5 Java 7 Automatic Resource Management and Catch block improvements

If you are catching a lot of exceptions in a single try block, you will notice that catch block code looks very ugly and mostly consists of redundant code to log the error, keeping this in mind Java 7 one of the feature was improved catch block where we can catch multiple exceptions in a single catch block. The catch block with this feature looks like below:

```
catch(IOException | SQLException | Exception ex){
    logger.error(ex);
throw new MyException(ex.getMessage());
}
```

There are some constraints such as the exception object is final and we can't modify it inside the catch block.

Most of the time, we use finally block just to close the resources and sometimes we forget to close them and get runtime exceptions when the resources are exhausted. These exceptions are hard to debug and we might need to look into each place where we are using that type of resource to make sure we are closing it. So java 7 one of the improvement was try-with-resources where we can create a resource in the try statement itself and use it inside the try-catch block. When the execution comes out of try-catch block, runtime environment automatically close these resources. Sample of try-catch block with this improvement is:

```
try (MyResource mr = new MyResource()) {

            System.out.println("MyResource created in try-with-resources");

        } catch (Exception e) {

e.printStackTrace();

}
```

10.6 Creating Custom Exception Classes

Java provides a lot of exception classes for us to use but sometimes we may need to create our own custom exception classes to notify the caller about specific type of exception with appropriate message and any custom fields we want to introduce for tracking, such as error codes. For example, let's say we write a method to process only text files, so we can provide caller with appropriate error code when some other type of file is sent as input.

Here is an example of custom exception class and showing its usage.

MyException.java

```
public class MyException extends Exception {
private static final long serialVersionUID = 4664456874499611218L;
private String errorCode="Unknown_Exception";
public MyException(String message, String errorCode){
super(message);
        this.errorCode=errorCode;
}
public String getErrorCode(){
        return this.errorCode;
}
}
```

CustomExceptionExample.java

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
public class CustomExceptionExample {
public static void main(String[] args) throws MyException{
try {
processFile("file.txt");
      } catch (MyException e){
processErrorCodes(e);
}
}
    private static void processErrorCodes(MyException e) throws MyException {
        switch(e.getErrorCode())
{
     case "BAD_FILE_TYPE":
System.out.println("Bad File Type, notify user");
     throw e;
case "FILE_NOT_FOUND_EXCEPTION":
```

```
System.out.println("File Not Found, notify user");
    throw e;
case "FILE_CLOSE_EXCEPTION":
System.out.println("File Close failed, just log it.");
break;
   default:
System.out.println("Unknown exception occured, lets log it for further
debugging."+e.getMessage());
          e.printStackTrace();
}
}
   private static void processFile(String file) throws MyException {
InputStream fis = null;
        try {
fis = new FileInputStream(file);
} catch (FileNotFoundException e) {
throw new MyException(e.getMessage(),"FILE_NOT_FOUND_EXCEPTION");
       }finally{
          try {
             if(fis !=null)fis.close();
} catch (IOException e) {
throw new MyException(e.getMessage(),"FILE_CLOSE_EXCEPTION");
}
}
}
}
```

Notice that we can have a separate method to process different types of error codes that we get from different methods, some of them gets consumed because we might not want to notify user for that or some of them we will throw back to notify user for the problem.

Here I am extending Exception so that whenever this exception is being produced, it has to be handled in the method or returned to the caller program, if we extends RuntimeException, there is no need to specify it in the throws clause. This is a design decision but I always like checked exceptions because I know what exceptions I can get when calling any method and take appropriate action to handle them.

10.7 Exception Handling Best Practices

- **Use Specific Exceptions** – Base classes of Exception hierarchy doesn't provide any useful information, that's why Java has so many exception classes, such as IOException with further sub-classes as FileNotFoundException, EOFException etc. We should always throw and catch specific exception classes so that caller will know the root cause of exception easily and process them. This makes debugging easy and helps client application to handle exceptions appropriately.
- **Throw Early or Fail-Fast** – We should try to throw exceptions as early as possible.
- While debugging we will have to look out at the stack trace carefully to identify the actual location of exception.

- Catch Late – Since java enforces to either handle the checked exception or to declare it in method signature, sometimes developers tend to catch the exception and log the error. But this practice is harmful because the caller program doesn't get any notification for the exception. We should catch exception only when we can handle it appropriately. For example, in above method I am throwing exception back to the caller method to handle it. The same method could be used by other applications that might want to process exception in a different manner. While implementing any feature, we should always throw exceptions back to the caller and let them decide how to handle it.
- Closing Resources – Since exceptions halt the processing of program, we should close all the resources in finally block or use Java 7 try-with-resources enhancement to let java runtime close it for you.
- Logging Exceptions – We should always log exception messages and while throwing exception provide clear message so that caller will know easily why the exception occurred. We should always avoid empty catch block that just consumes the exception and doesn't provide any meaningful details of exception for debugging.
- Single catch block for multiple exceptions – Most of the times we log exception details and provide message to the user, in this case we should use java 7 feature for handling multiple exceptions in a single catch block. This approach will reduce our code size and it will look cleaner too.
- Using Custom Exceptions – It's always better to define exception handling strategy at the design time and rather than throwing and catching multiple exceptions, we can create a custom exception with error code and caller program can handle these error codes. It's also a good idea to create a utility method to process different error codes and use it.
- Naming Conventions and Packaging – When you create your custom exception make sure it ends with Exception so that it will be clear from name itself that it's an exception. Also make sure to package them like it's done in JDK, for example IOException is the base exception for all IO operations.
- Use Exceptions Judiciously – Exceptions are costly and sometimes it's not required to throw exception at all and we can return a boolean variable to the caller program to indicate whether an operation was successful or not. This is helpful where the operation is optional and you don't want your program to get stuck because it fails.

Document the Exceptions Thrown – Use javadoc @throws to clearly specify the exceptions thrown by the method, it's very helpful

Exercises

1. Why some errors are called exceptions?
2. What is the key reason for using finally blocks?
3. What happens to a local reference in a try block when that block throws an Exception?
4. Give a key advantage of using catch (Exception exceptionName).
5. What happens if no catch handler matches the type of a thrown object?

Chapter 11. Java files and I/O

11.1 Introduction

The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, object, localized characters, etc.

11.2 Stream

A stream can be defined as a sequence of data. There are two kinds of Streams

- **InPutStream**: The InputStream is used to read data from a source.
- **OutPutStream**: the OutputStream is used for writing data to a destination.

Java provides strong but flexible support for I/O related to Files and networks but this manual covers very basic functionality related to streams and I/O. We would see most commonly used example one by one:

Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, FileInputStream and FileOutputStream.

Following is an example which makes use of these two classes to copy an input file into an output file:

```
import java.io.*;
public class CopyFile {
  public static void main(String args[]) throws IOException
  {
    FileInputStream in = null;
    FileOutputStream out = null;
    try {
      in = new FileInputStream("input.txt");
      out = new FileOutputStream("output.txt");

      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
      }
    }finally {
      if (in != null) {
        in.close();
      }
      if (out != null) {
        out.close();   }
    }
  }
}
```

Now let's have a file input.txt with the following content:

This is test for copy file.

As a next step, compile above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put above code in CopyFile.java file and do the following:

javac CopyFile.java

java CopyFile

Character Streams

Java Byte streams are used to perform input and output of 8-bit bytes, where as Java Character streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are , FileReader and FileWriter.. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

We can re-write above example which makes use of these two classes to copy an input file (having unicode characters) into an output file:

```
import java.io.*;
public class CopyFile {
  public static void main(String args[]) throws IOException
  {
    FileReader in = null;
    FileWriter out = null;
    try {
      in = new FileReader("input.txt");
      out = new FileWriter("output.txt");
      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
      }
    }finally {
      if (in != null) {
        in.close();
      }
      if (out != null) {
        out.close();
      }
    }
  }
}
```

Now let's have a file input.txt with the following content:

This is test for copy file.

As a next step, compile above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put above code in CopyFile.java file and do the following:

javac CopyFile.java

java CopyFile

Standard Streams

All the programming languages provide support for standard I/O where user's program can take input from a keyboard and then produce output on the computer screen. If you are aware if C or C++ programming languages, then you must be aware of three standard devices STDIN, STDOUT and STDERR. Similar way Java provides following three standard streams

- **Standard Input**: This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as System.in.
- **Standard Output**: This is used to output the data produced by the user's program and usually a computer screen is used to standard output stream and represented as System.out.
- **Standard Error**: This is used to output the error data produced by the user's program and usually a computer screen is used to standard error stream and represented as System.err.

Following is a simple program which creates InputStreamReader to read standard input stream until the user types a "q":

```
import java.io.*;
public class ReadConsole {
  public static void main(String args[]) throws IOException
  {
    InputStreamReader cin = null;
    try {
      cin = new InputStreamReader(System.in);
      System.out.println("Enter characters, 'q' to quit.");
      char c;
      do {
        c = (char) cin.read();
        System.out.print(c);
      } while(c != 'q');
    }finally {
      if (cin != null) {
        cin.close();
      }
    }
  }
}
```

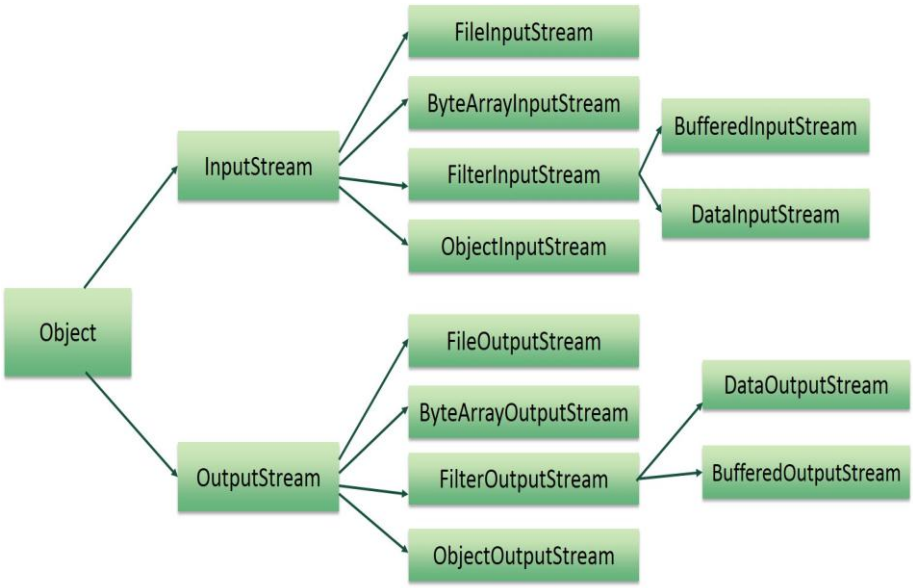
Let's keep above code in ReadConsole.java file and try to compile and execute it as below. This program continues reading and outputting same character until we press 'q':

```
$javac ReadConsole.java
$java ReadConsole
Enter characters, 'q' to quit.
1
1
e
e
q
q
```

11.3 Reading and Writing Files

As described earlier, A stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.

Here is a hierarchy of classes to deal with Input and Output streams.



The two important streams are FileInputStream and FileOutputStream, which would be discussed in this manual:

FileInputStream:

This stream is used for reading data from the files. Objects can be created using the keyword new and there are several types of constructors available.

Following constructor takes a file name as a string to create an input stream object to read the file.

InputStream f = new FileInputStream("C:/java/hello");

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows:

File f = new File("C:/java/hello");

InputStream f = new FileInputStream(f);

Once you have InputStream object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

SN	Methods with Description
1	public void close() throws IOException{} This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.
2	protected void finalize()throws IOException {} This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
3	public int read(int r)throws IOException{} This method reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and -1 will be returned if it's end of file.

4	public int read(byte[] r) throws IOException{} This method reads r.length bytes from the input stream into an array. Returns the total number of bytes read. If end of file -1 will be returned.
5	public int available() throws IOException{} Gives the number of bytes that can be read from this file input stream. Returns an int.

FileOutputStream:

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Here are two constructors which can be used to create a FileOutputStream object.

Following constructor takes a file name as a string to create an input stream object to write the file:

OutputStream f = new FileOutputStream("C:/java/hello")

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows:

File f = new File("C:/java/hello");

OutputStream f = new FileOutputStream(f);

Once you have OutputStream object in hand, then there is a list of helper methods, which can be used to write to stream or to do other operations on the stream.

SN	Methods with Description
1	public void close() throws IOException{} This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException
2	protected void finalize()throws IOException {} This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
3	public void write(int w)throws IOException{} This methods writes the specified byte to the output stream.
4	public void write(byte[] w) Writes w.length bytes from the mentioned byte array to the OutputStream.

Example:

Following is the example to demonstrate InputStream and OutputStream:

```
import java.io.*;
public class fileStreamTest{
   public static void main(String args[]){
         try{
      byte bWrite [] = {11,21,3,40,5};
      OutputStream os = new FileOutputStream("test.txt");
      for(int x=0; x < bWrite.length ; x++){
        os.write( bWrite[x] ); // writes the bytes
      }
```

```
      os.close();
      InputStream is = new FileInputStream("test.txt");
      int size = is.available();
      for(int i=0; i< size; i++){
        System.out.print((char)is.read() + "  ");
      }
      is.close();
    }catch(IOException e){
      System.out.print("Exception");
    }
    }
}
```

The above code would create file test.txt and would write given numbers in binary format. Same would be output on the stdout screen.

11.4 Directories in Java

A directory is a File which can contains a list of other files and directories. You use File object to create directories, to list down files available in a directory. For complete detail check a list of all the methods which you can call on File object and what are related to directories.

Creating Directories:

There are two useful File utility methods, which can be used to create directories:

- The mkdir( ) method creates a directory, returning true on success and false on failure. Failure indicates that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet.
- The mkdirs() method creates both a directory and all the parents of the directory.

Following example creates "/tmp/user/java/bin" directory:

```
import java.io.File;
public class CreateDir {
  public static void main(String args[]) {
    String dirname = "/tmp/user/java/bin";
    File d = new File(dirname);
    // Create directory now.
    d.mkdirs();
  }
}
```

Compile and execute above code to create "/tmp/user/java/bin".

Note: Java automatically takes care of path separators on UNIX and Windows as per conventions. If you use a forward slash (/) on a Windows version of Java, the path will still resolve correctly.

Listing Directories:

You can use list( ) method provided by File object to list down all the files and directories available in a directory as follows:

```
import java.io.File;
public class ReadDir {
  public static void main(String[] args) {
    File file = null;
    String[] paths;
    try{
      // create new file object
      file = new File("/tmp");
      // array of files and directory
      paths = file.list();
     // for each name in the path array
      for(String path:paths)
      {
        // prints filename and directory name
        System.out.println(path);
      }
    }catch(Exception e){
      // if any error occurs
      e.printStackTrace();
    }
  }
}
```

This would produce following result based on the directories and files available in your /tmp directory:

```
test1.txt
test2.txt
ReadDir.java
ReadDir.class
```

11.5 Accepting Input from a User

There are two popular ways for a program to accept inputs from a user: -

- By using Scanner class
- By using command line argument

1.5.1 Scanner class

One of the strengths of Java is the huge libraries of code available to you. This is code that has been written to do specific jobs. All you need to do is to reference which library you want to use, and then call a method into action. One really useful class that handles input from a user is called the Scanner class. The Scanner class can be found in the java.util library. To use the Scanner class, you need to reference it in your code. This is done with the keyword import.

```
import java.util.Scanner;
```

The import statement needs to go just above the Class statement:

```
import java.util.Scanner;
public class StringVariables {
}
```

This tells java that you want to use a particular class in a particular library - the Scanner class, which is located in the java.util library.

The next thing you need to do is to create an object from the Scanner class:

```
Scanner user_input = new Scanner( System.in );
```

So instead of setting up an int variable or a String variable, we're setting up a Scanner variable. We've called ours user_input. After an equals sign, we have the keyword new. This is used to create new objects from a class. The object we're creating is from the Scanner class. In between round brackets we have to tell java that this will be System Input (System.in).

To get the user input, you can call into action one of the many methods available to your new Scanner object. One of these methods is called next. This gets the next string of text that a user types on the keyboard:

```
String first_name;
first_name = user_input.next( );
```

We can also print some text to prompt the user:

```
String first_name;
System.out.print("Enter your first name: ");
first_name = user_input.next( );
```

Notice that we've used print rather than println like last time. The difference between the two is that println will move the cursor to a new line after the output, but print stays on the same line.

We'll add a prompt for a family name, as well:

```
String family_name;
System.out.print("Enter your family name: ");
family_name = user_input.next( );
```

This is the same code, except that java will now store whatever the user types into our family_name variable instead of our first_name variable.

To print out the input, we can add the following:

```
String full_name;
full_name = first_name + " " + family_name;
System.out.println("You are " + full_name);
```

We've set up another String variable, full_name. We're storing whatever is in the two variables first_name and family_name. In between the two, we've added a space. The final line prints it all out in the Output window.

1.5.2Command line argument

You must have now by realized what String[] args within the declaration of the main method means. It represents a String array named args. Since args is nothing more than an identifier, we can replace it with any other identifier and the program will still work. What we need to know now is how a String array can be passed as an argument when we execute the program. We pass it through the command line itself. Consider that we have a class named Add. The following statement normally used to execute the program.

```
java Add
```

When we wish to pass the String array, we simply include the elements of the array as simple Strings beside the class name. Enclosing the Strings in quotes is optional. Consecutive Strings are separated with a space. For example, if we wish to pass a three element String array containing the values "1", "2", and "3" any of the following lines is entered on the command prompt.

```
java Add 1 2 3
java Add "1" "2" "3"
```

Since these arguments are passed through the command line, they are known as command line arguments. The String arguments passed are stored in the array specified in the main() declaration. args[] is now a three element String array. These elements are accessed in the same way as the elements of a normal array. The following is the complete Add program which is capable of adding any number of integers passed as command line arguments.

```
public class Add {

   public static void main(String[] args) {
      int sum = 0;
      for (int i = 0; i < args.length; i++) {
         sum = sum + Integer.parseInt(args[i]);
      }
      System.out.println("The sum of the arguments passed is " + sum);
   }
}
```

Exercises

1. What is a stream
2. Write a program that will accept inputs from the user and store them to a file
3. Write a program that will copy contents of one file to another file
4. Write a program that will accept two numbers typed by a user and compare them

Chapter 12. Building a simple user interface

12.1 Introduction

Programs that use a graphical user interface and mouse control are called windowing software. Although you probably have been using a command-line interface to write Java programs, during this hour you create windowing programs using a group of classes called Swing.

12.2 Swing and the Abstract Windowing Toolkit

Swing and the Abstract Windowing Toolkit include everything you need to write programs that use a graphical user interface, which is also called a GUI (pronounced gooey, as in Huey, Dewey, and Louie). With Java's windowing classes, you can create a GUI that includes all of the following and more:

- Buttons, check boxes, labels, and other simple components
- Text fields, sliders, and other more complex components
- Pull-down menus and pop-up menus
- Windows, frames, dialog boxes, and applet windows

12.3 Using Components

In Java, every part of a graphical user interface is represented by a class in the Swing or Abstract Windowing Toolkit packages. There is a JButton class for buttons, a JWindow class for windows, a JTextField class for text fields, and so on.

To create and display an interface, you create objects, set their variables, and call their methods. The techniques are the same as those you used during the previous topics as you were introduced to object-oriented programming.

When you are putting a graphical user interface together, you work with two kinds of objects: components and containers. A component is an individual element in a user interface, such as a button or slider. A container is a component that can be used to hold other components.

The first step in creating an interface is to create a container that can hold components. In an application, this container is often a frame or a window.

Frames and Windows

Windows and frames are containers that can be displayed on a user's desktop. Windows are simple containers that do not have a title bar or any of the other buttons normally along the top edge of a graphical user interface. Frames are windows that include all of these common windowing features users expect to find when they run software—such as buttons to close, expand, and shrink the window.

These containers are created using Swing's JWindow and JFrame classes. To make the Swing package of classes available in a Java program, use the following statement:

```
import javax.swing.*;
```

One way to make use of a frame in a Java application is to make the application a subclass of JFrame. Your program will inherit the behavior it needs to function as a frame. The following statements create a subclass of JFrame:

```
import javax.swing.*;
public class MainFrame extends JFrame {
   public MainFrame() {
      // set up the frame
   }
}
```

This class creates a frame, but doesn't set it up completely. In the frame's constructor method, you must do several actions when creating a frame:

- Call a constructor method of the superclass, JFrame.
- Set up the title of the frame.
- Set up the size of the frame.
- Define what happens when the frame is closed by a user.

You also must make the frame visible, unless for some reason it should not be displayed when the application begins running.

All of these things can be handled in the frame's constructor method. The first thing the method must contain is a call to one of the constructor methods of JFrame, using the super statement. Here's an example:

```
super();
```

The preceding statement calls the JFrame constructor with no arguments. You also can call it with the title of your frame as an argument:

```
super("Main Frame");
```

This sets the title of the frame, which appears in the title bar along the top edge, to the specified string. In this example, the text greeting "Main Frame" will appear.

If you don't set up a title in this way, you can call the frame's setTitle() method with a String as an argument:

```
setTitle("Main Frame");
```

The size of the frame can be established by calling its setSize() method with two arguments: the width and height. The following statement sets up a frame that is 350 pixels wide and 125 pixels tall:

```
setSize(350, 125);
```

Another way to set the size of a frame is to fill it with components and then call the frame's pack() method with no arguments, as in this example:

```
pack();
```

The pack() method sets the frame up based on the preferred size of each component inside the frame. Every interface component has a preferred size, though this is sometimes disregarded, depending on how components have been arranged within a container. You don't need to explicitly set the size of a frame before calling pack()—the method sets it to an adequate size before the frame is displayed.

Every frame is displayed with a button along the title bar that can be used to close the frame. On a Windows system, this button appears as an X in the upper-right corner of the frame. To define what happens when this button is clicked, call the frame's setDefaultCloseOperation() method with one of four JFrame class variables as an argument:

- setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)— Exit the program when the button is clicked.
- setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE)— Close the frame, dispose of the frame object, and keep running the application.
- setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE)— Keep the frame open and continue running.
- setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE)— Close the frame and continue running.

The last thing that's required is to make the frame visible: call its setVisible() method with true as an argument:

```
setVisible(true);
```

This opens the frame at the defined width and height. You also can call it with false to stop displaying a frame.

The following program contains the source code described in this section. Enter these statements and save the file as SaluteFrame.java.

The Full Text of SaluteFrame.java

```
import javax.swing.*;
public class SaluteFrame extends JFrame {
   public SalutonFrame() {
   super("Hello World !");
     setSize(350, 100);
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       setVisible(true);
     }
     public static void main(String[] arguments) {
     SaluteFrame sal = new SaluteFrame();
    }
 }
```

The program contain a main() method, which turns this frame class into an application that can be run at the command line. After you compiling this into a class, run the application with the following command:

java SaluteFrame

The only thing that SaluteFrame displays is a greeting "Hello World!" The frame is an empty window, because it doesn't contain any other components yet.

To add components to a frame, window, or an applet, you must create the component and add it to the container. Each container has an add() method that takes one argument: the component to display.

Buttons

One simple component you can add to a container is a JButton object. JButton, like the other components you'll be working with during this hour, is part of the java.awt.swing package. A JButton object is a clickable button with a label that describes what clicking the button will do. This label can be text, graphics, or both. The following statement creates a JButton called okButton and gives it the text label OK:

```
JButton okButton = new JButton("OK");
```

After a component such as JButton is created, it should be added to a container by calling its add() method:

```
add(okButton);
```

When you add components to a container, you do not specify the place in the container where the component should be displayed. The arrangement of components is decided by an object called a layout manager. The simplest of these managers is the FlowLayout class, which is part of the java.awt package.

To make a container use a specific layout manager, you must first create an object of that layout manager's class. A FlowLayout object is created with a statement, such as the following:

```
FlowLayout fff = new FlowLayout();
```

Once a layout manager has been created, the container's setLayout() method is called to associate the manager with the container. The only argument to this method should be the layout manager object, as in the following example:

```
pane.setLayout(fff);
```

This statement designates the fff object as the layout manager for the pane container.

The following program displays a frame with three buttons.

The Full Text of Playback.java

```
import javax.swing.*;
import java.awt.*;
public class Playback extends JFrame {
public Playback() {
  super("Playback");
```

```
        setSize(225, 80);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
        FlowLayout flo = new FlowLayout();
        setLayout(flo);
        JButton play = new JButton("Play");
        JButton stop = new JButton("Stop");
        JButton pause = new JButton("Pause");
        add(play);
        add(stop);
        add(pause);
        setVisible(true);
      }
public static void main(String[] arguments) {
        Playback pb = new Playback();
    }
```

Many of the user components available as part of Swing can be added to a container in this manner.

Labels and Text Fields

A JLabel component displays information that cannot be modified by the user. This information can be text, a graphic, or both. These components are often used to label other components in an interface, hence the name. They often are used to identify text fields.

A JTextField component is an area where a user can enter a single line of text. You can set up the width of the box when you create the text field.

The following statements create a JLabel component and JTextField object and add them to a container:

```
JLabel pageLabel = new JLabel("Web page address: ", JLabel.RIGHT);

JTextField pageAddress = new JTextField(20);

FlowLayout flo = new FlowLayout();

setLayout(flo);

add(pageLabel);

add(pageAddress);
```

The pageLabel label is set up with the text Web page address: and a JLabel.RIGHT argument. This last value indicates that the label should appear flush right. JLabel.LEFT aligns the label text flush left, and JLabel.CENTER centers it. The argument used with JTextField indicates that the text field should be approximately 20 characters wide. You also can specify default text that will appear in the text field with a statement such as the following:

JTextField country = new JTextField("US", 29);

This statement would create a JTextField object that is 20 characters wide and has the text US in the field.

The text contained within the object can be retrieved with the getText() method, which returns a string:

String countryChoice = country.getText();

As you might have guessed, you also can set the text with a corresponding method:

countryChoice.setText("Socialist People's Libyan Arab Jamahiriya");

This sets the text to the official name of Libya, which is the longest in the world, edging out the second-place finisher: the United Kingdom of Great Britain and Northern Ireland.

Check Boxes

A JCheckBox component is a box next to a line of text that can be checked or unchecked by the user. The following statements create a JCheckBox object and add it to a container:

JCheckBox jumboSize = new JCheckBox("Jumbo Size");

FlowLayout flo = new FlowLayout();

setLayout(flo);

add(jumboSize);

The argument to the JCheckBox() constructor method indicates the text to be displayed alongside the box. If you wanted the box to be checked, you could use the following statement instead:

JCheckBox jumboSize = new JCheckBox("Jumbo Size", true);

A JCheckBox can be presented singly or as part of a group. In a group of check boxes, only one can be checked at a time. To make a JCheckBox object part of a group, you have to create a ButtonGroup object. Consider the following:

JCheckBox frogLegs = new JCheckBox("Frog Leg Grande", true);

JCheckBox fishTacos = new JCheckBox("Fish Taco Platter", false);

JCheckBox emuNuggets = new JCheckBox("Emu Nuggets", false);

FlowLayout flo = new FlowLayout();

ButtonGroup meals = new ButtonGroup();

```
meals.add(frogLegs);

meals.add(fishTacos);

meals.add(emuNuggets);

setLayout(flo);

add(jumboSize);

add(frogLegs);

add(fishTacos);

add(emuNuggets);
```

This creates three check boxes that are all grouped under the ButtonGroup object called meals. The Frog Leg Grande box is checked initially, but if the user checked one of the other meal boxes, the check next to Frog Leg Grande would disappear automatically.

Combo Boxes

A JComboBox component is a pop-up list of choices that also can be set up to receive text input. When both options are enabled, you can select an item with your mouse or use the keyboard to enter text instead. The combo box serves a similar purpose to a group of check boxes, except that only one of the choices is visible unless the pop-up list is being displayed.

To create a JComboBox object, you have to add each of the choices after creating the object, as in the following example:

```
JComboBox profession = new JComboBox();

FlowLayout flo = new FlowLayout();

profession.addItem("Butcher");

profession.addItem("Baker");

profession.addItem("Candlestick maker");

profession.addItem("Fletcher");

profession.addItem("Fighter");

profession.addItem("Technical writer");

setLayout(flo);
```

```
add(profession);
```

This example creates a single JComboBox component that provides six choices from which the user can select.

To enable a JComboBox component to receive text input, its setEditable() method must be called with an argument of TRue, as in the following statement:

```
profession.setEditable(true);
```

This method must be called before the component is added to a container.

Text Areas

A JTextArea component is a text field that enables the user to enter more than one line of text. You can specify the width and height of the component. For example, the following statements create a JTextArea component with an approximate width of 40 characters and a height of 8 lines, and then add the component to a container:

```
JTextArea comments = new JTextArea(8, 40);

FlowLayout flo = new FlowLayout();

setLayout(flo);

add(comments);
```

You can specify a string in the JTextArea() constructor method to be displayed in the text area. You can use the newline character ("\n") to send text to the next line, as in the following:

```
JTextArea desire = new JTextArea("I should have been a pair\n"
    + "of ragged claws.", 10, 25);
```

Panels

The other components you'll learn to create are panels, which are created in Swing using the JPanel class. JPanel objects are the simplest kind of container you can use in a Swing interface. The purpose of JPanel objects is to subdivide a display area into different groups of components. When the display is divided into sections, you can use different rules for how each section is organized.

You can create a JPanel object and add it to a container with the following statements:

```
JPanel topRow = new JPanel();
```

```
FlowLayout flo = new FlowLayout();

setLayout(flo);

add(topRow);
```

Panels are often used when arranging the components in an interface. Unlike other containers, panels do not have a content pane. Instead, you can add components by calling the panel's add() method directly. You also can assign a layout manager directly to the panel by calling its setLayout() method.

Panels also can be used when you need an area in an interface to draw something, such as an image from a graphics file. Another convenient use of JPanel is to create your own components that can be added to other classes. This is demonstrated in this hour's workshop.

12.4 Laying Out a User Interface

Components are organized in an interface by using a set of classes called layout managers. Each container in an interface can have its own layout manager.

Using Layout Managers

In Java, the placement of components within a container depends on the size of other components and the height and width of the container. The layout of buttons, text fields, and other components can be affected by the following things:

- The size of the container
- The size of other components and containers
- The layout manager that is being used

There are several layout managers you can use to affect how components are shown. The default manager for panels is the FlowLayout class in the java.awt package

Under FlowLayout, components are dropped onto an area in the same way words are organized on a printed page in English—from left to right, then on to the next line when there's no more space.

Here is how a FlowLayout manager is used by calling the Container object's setLayout() method with the FlowLayout object as an argument.

```
FlowLayout topLayout = new FlowLayout();

setLayout(topLayout);
```

You also can set up a layout manager to work within a specific container, such as a JPanel object. You can do this by using the setLayout() method of that container object. The following statements create a JPanel object called inputArea and set it up to use FlowLayout as its layout manager:

```
JPanel inputArea = new JPanel();

FlowLayout inputLayout = new FlowLayout();

inputArea.setLayout(inputLayout);
```

The Crisis application has a graphical user interface with five buttons. Load your word processor and open up a new file called Crisis.java. Enter the following code and save the file when you're done.

```
 The Full Text of Crisis.java
 import java.awt.*;
 import javax.swing.*;
 public class Crisis extends JFrame {
 JButton panicButton = new JButton("Panic");
JButton dontPanicButton = new JButton("Don't Panic");
JButton blameButton = new JButton("Blame Others");
JButton mediaButton = new JButton("Notify the Media");
JButton saveButton = new JButton("Save Yourself");
public Crisis() {
super("Crisis");
setSize(308, 128);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
FlowLayout flo = new FlowLayout();
setLayout(flo);
add(panicButton);
add(dontPanicButton);
add(blameButton);
add(mediaButton);
add(saveButton);
setVisible(true);
}
public static void main(String[] arguments) {
Crisis cr = new Crisis();
 }
}
```

After compiling the Crisis application, you can run it with the following command:

```
java Crisis
```

The FlowLayout class uses the dimensions of its container as the only guideline for how to lay out components. Resize the window of the application to see how components are instantly rearranged. Make the window twice as wide, and you'll see all of the JButton components are now shown on the same line. Java programs often will behave differently when their display area is resized.

The GridLayout Manager

The GridLayout class in the java.awt package organizes all components in a container into a specific number of rows and columns. All components are allocated the same amount of size in the display area, so if you specify a grid that is three columns wide and three rows tall, the container will be divided into nine areas of equal size.

The following statements create a container and set it to use a grid layout that is two rows wide and three columns tall:

```
GridLayout grid = new GridLayout(2, 3);

setLayout(grid);
```

GridLayout places all components as they are added into a place on a grid. Components are added from left to right until a row is full and then the leftmost column of the next grid is filled.

The BorderLayout Manager

The next layout manager to experiment with is the BorderLayout class, also in java.awt. The following statements create a container that uses border layout, placing components at specific locations within the layout:

```
BorderLayout crisisLayout = new BorderLayout();

setLayout(crisisLayout);

add(panicButton, BorderLayout.NORTH);
```

The BorderLayout manager arranges components into five areas: four denoted by compass directions and one for the center area. When you add a component under this layout, the add() method includes a second argument to specify where the component should be placed. This argument should be one of five class variables of the BorderLayout class: NORTH, SOUTH, EAST, WEST, and CENTER are used for this argument.

Like the GridLayout class, BorderLayout devotes all available space to the components. The component placed in the center is given all the space that isn't needed for the four border components, so it's usually the largest.

Separating Components with Insets

As you are arranging components within a container, you might want to move components away from the edges of the container. This is accomplished using Insets, an object that represents the border area of a container.

The Insets class, which is part of the java.awt package, has a constructor method that takes four arguments: the space to leave at the top, left, bottom, and right on

the container. Each argument is specified using pixels, the same unit of measure employed when defining the size of a frame.

The following statement creates an Insets object:

```
Insets around = new Insets(10, 6, 10, 3);
```

The around object represents a container border that is 10 pixels inside the top edge, 6 pixels inside the left, 10 pixels inside the bottom, and 3 pixels inside the right.

To make use of an Insets object in a container, you must override the container's getInsets() method. This method has no arguments and returns an Insets object, as in the following example:

```
public Insets getInsets() {

   Insets squeeze = new Insets(50, 15, 10, 15);

   return squeeze;
}
```

12.5 Java applets

Now that Java has made the transition from a child prodigy to an established language, it is being used for all kinds of large-scale business software and other applications. How ever, for some people, the core appeal of the language remains a type of program that Java made possible: the applet.

Applets are programs designed to run as part of a World Wide Web page. When a Java applet is encountered on a page, it is downloaded to the user's computer and begins running.

Programming applets with Java is much different from creating applications with Java. Because applets must be downloaded from a page each time they are run, they're smaller than most applications to reduce download time. Also, because applets run on the computer of the person using the applet, they have numerous security restrictions in place to prevent malicious or damaging code from being run.

12.6 Standard Applet Methods

The first step in the creation of an applet is to make it a subclass of JApplet, a class that's part of the Swing package, javax.swing. An applet is treated as a visual window inside a web page, so JApplet is part of Swing alongside clickable buttons, scrollbars, and other components of a program's user interface.

JApplet is a subclass of Applet, a class in the java.applet package. Being part of this hierarchy enables the applets you write to use all the behavior and attributes they need to be run as part of a web page. Before you begin writing any other statements in your applets, they will be able to interact with a web browser, load and unload themselves, redraw their window in response to changes in the browser window, and handle other necessary tasks.

In applications, programs begin running with the first statement inside the main() block statement and end with the last closing bracket (}) that closes out the block. There is no main() method in a Java applet, so there is no set starting place for the program. Instead, an applet has a group of standard methods that are handled in response to specific events as the applet runs.

The following are the events that could prompt one of the applet methods to be handled:

- The program is loaded for the first time, which causes the applet's init() and start() methods to be called.
- Something happens that requires the applet window to be redisplayed, which causes the applet's paint() method to be called.
- The program is stopped by the browser, which calls the applet's stop() method.
- The program restarts after a stop, which calls the start() method.
- The program is unloaded as it finishes running, which calls the destroy() method.

The following is an example of a bare-bones applet:

```
public class Skeleton extends javax.swing.JApplet {

   // program will go here

}
```

Unlike applications, applet class files must be public because the JApplet class is also public. (If your applet uses other class files of your own creation, they do not have to be declared public.)

Your applet's class inherits all the methods that are handled automatically when needed: init(), paint(), start(), stop(), and destroy(). However, none of these methods do anything. If you want something to happen in an applet, you have to override these methods with new versions in your applet program.

Painting an Applet Window

The paint() method is used to display text, shapes, and graphics within the applet window. Whenever something needs to be displayed or redisplayed on the applet window, the paint() method handles the task. You also can force paint() to be handled with the following statement in any method of an applet:

```
repaint();
```

Aside from the use of repaint(), the main time the paint() method is handled is when something changes in the browser or the operating system running the browser. For example, if a user minimizes a web page containing an applet, the paint() method will be called to redisplay everything that was onscreen in the applet when the applet is later restored to full size.

Unlike the other methods you will be learning about, paint() takes an argument. The following is an example of a simple paint() method:

```
public void paint(Graphics screen) {

   Graphics2D screen2D = (Graphics2D)screen;

   // display statements go here

}
```

The argument sent to the paint() method is a Graphics object. The Graphics class of objects represents an environment in which something can be displayed, such as an applet window. As you did with Swing programs, you may cast this to a Graphics2D object to employ more sophisticated graphical features.

Later, you'll learn about drawString(), a method for the display of text that's available in both the Graphics and Graphics2D classes.

If you are using a Graphics or Graphics2D object in your applet, you have to add the following import statements before the class statement at the beginning of the source file:

```
import java.awt.Graphics;

import java.awt.Graphics2D;
```

Note: If you are using several classes that are a part of the java.awt package of classes, you can use the statement import java.awt.* to make all of these classes available for use in your program.

Initializing an Applet

The init() method is handled once—and only once—when the applet is run. As a result, it's an ideal place to set up values for any objects and variables that are needed for the applet to run successfully. This method is also a good place to set up fonts, colors, and the screen's background color. Here's an example:

```
public void init() {

FlowLayout flo = new FlowLayout();

   setLayout(flo);

   JButton run = new JButton("Run");

   add(run);

}
```

If you are going to use a variable in other methods, it should not be created inside an init() method because it will only exist within the scope of that method.

For example, if you create an integer variable called displayRate inside the init() method and try to use it in the paint() method, you'll get an error when you attempt to compile the program. Create any variables you need to use throughout a class as object variables right after the class statement and before any methods.

Starting and Stopping an Applet

At any point when the applet program starts running, the start() method will be handled. When a program first begins, the init() method is followed by the start() method. After that, in many instances there will never be a cause for the start() method to be handled again. In order for start() to be handled a second time or more, the applet has to stop execution at some point.

The stop() method is called when an applet stops execution. This event can occur when a user leaves the web page containing the applet and continues to another page. It also can occur when the stop() method is called directly in a program.

Destroying an Applet

The destroy() method is an opposite of sorts to the init() method. It is handled just before an applet completely closes down and completes running.

This method is used when something has been changed during a program that should be restored to its original state. It's another method you'll use more often with animation than with other types of programs.

12.7 Putting an Applet on a Web Page

Applets are placed on a web page in the same way that anything unusual is put on a page: HTML markup tags describe the applet, which a web browser loads along with the other parts of the page.

One way to place applets on a web page is by using an applet tag and several attributes. The following is an example of the HTML required to put an applet on a page:

```
<applet code="StripYahtzee.class" codebase="javadir" height="300" width="400">

Sorry, no dice ... this requires a Java-enabled browser.

</applet>
```

The code attribute identifies the name of the applet's class file. If more than one class file is being used with an applet, code should refer to the main class file that is a subclass of the JApplet class.

If there is no code attribute, all files associated with the applet should be in the same folder as the web page that loads the program. codebase should contain a reference to the folder or subfolder where the applet and any related files can be found. In the preceding example, codebase indicates that the StripYahtzee applet can be found in the javadir subfolder.

The height and width attributes designate the exact size of the applet window on the web page. It must be big enough to handle the things you are displaying in your applet.

In between the opening <applet> tag and the closing </applet> tag, you can provide an alternate of some kind for web users whose browser software cannot run Java programs (less than two percent of all web users run browsers that fall into this group).

In the preceding example, the text "Sorry, no dice…this requires a Java-enabled browser" is displayed in place of the applet on a browser such as Lynx, which does not support Java. You can put instructions here on how to download a Java-enabled browser. You also can include hyperlinks and other HTML elements.

Another useful attribute, align, designates how an applet will be displayed in relation to the surrounding material on the page, including text and graphics. The value align="left" lines up the applet to the left of adjacent page elements and align="right" lines it up to the right.

A Sample Applet

This hour's first project is an applet that displays the string "Hello world !”.

Load your word processor and create a new file called SaluteApplet.java. Enter the following text and save it when you're done.

The Full Text of SaluteApplet.java

```
import java.awt.*;
public class SaluteApplet extends javax.swing.JApplet {
String greeting;
        public void init() {
                greeting = "Hello World !";
        }
public void paint(Graphics screen) {
Graphics2D screen2D = (Graphics2D)screen;
screen2D.drawString(greeting, 25, 50);
}
}
```

This applet does not need to use the start(), stop(), or destroy() methods, so they are not included in the program. Compile the program with the javac compiler tool, if you're an JDK user, or another tool.

Drawing in An Applet Window

Text is displayed in an applet window by using the drawString() method of the Graphics2D class, which draws text in a graphical user interface component.

The drawString() method is similar in function to the System.out.println() method that displays information to the system's standard output device.

Before you can use the drawString() method, you must have a Graphics or Graphics2D object that represents the applet window.

The paint() method of all applets includes a Graphics object as its only argument, which can be cast to a Graphics2D object:

Graphics2D screen2D = (Graphics2D)screen;

When you have created a Graphics2D object like this, you can call its drawString() method to display text on the area represented by the object.

The following three arguments are sent to drawString():

- The text to display, which can be several different strings and variables strung together with the + operator
- The x position (in an (x,y) coordinate system) where the string should be displayed
- The y position where the string should be displayed

The (x,y) coordinate system in an applet is used with several methods. It begins with the (0,0) point in the upper-left corner of the applet window.

Testing the SaluteApplet Program

Although you have compiled the SaluteApplet program into a class file, you cannot run it using a Java interpreter such as java. If you do, you'll get an error message looking like this:

Exception in thread "main" java.lang.NoSuchMethodError: main

The error occurs because a Java interpreter runs applications by calling its main() method. Applets don't include this method. Instead, to run an applet, you need to create a web page that loads the applet. To create a web page, open up a new file on your word processor and call it SaluteApplet.html. Enter the following program and then save the file.

The Full Text of SaluteApplet.html

```
<html>
<head>
<title>Hello World !</title>
</head>
<body bgcolor="#000000" text="#FF00FF">
```

```
<center>
This a Java applet:<br>
<applet code="SaluteApplet.class" height=150 width=300>
You need a Java-enabled browser to see this.
</applet>
</body>
</html>
```

All applets you write can be tested with the appletviewer tool that comes with the Java Development Kit. You can see the output of the SaluteApplet applet by typing the following:

appletviewer SaluteApplet.html

One thing to note about appletviewer is that it only runs the applets that are included in a web page, and does not handle any of the other elements such as text and images.

Applets can also be loaded by web browsers, if they are equipped with the Java Plug-in. To attempt this at a command line, type the following command:

SaluteApplet.html

You can also choose File, Open from the browser's menu to find and open the page.

If you can't get this applet to run in Firefox or another web browser, the most likely reason is that the browser needs the Java Plug-in.

The Java Plug-in

Though popular web browsers began including their own Java interpreters with the first release of the language in the mid-'90s, these interpreters failed to keep current.

If you installed the Java Development Kit, you were given a chance to install the Java Plug-in at the same time.

The Java Plug-in runs Java applets in place of the web browser's Java interpreter. Once the Java Plug-in is installed, all future Java 2 applets will run automatically if they specify that the Plug-in should be used to run them.

The plug-in is part of the Java Runtime Environment, which can be downloaded at no cost from Sun's Java site at the address http://java.com.

12.8 Sending and receiving Parameters from/to a Web Page

Sending Parameters from a Web Page

One of the driving forces behind parameter use in Java applets is the fear and loathing of compilation. Parameters enable you to change elements of an applet without editing or recompiling anything. They also make the program more useful.

Parameters are stored as part of the web page that contains an applet. They are created using the HTML tag param and its two attributes: name and value. You can have more than one param tag with an applet, but all of them must be between the opening <applet> tag and the closing </applet> tag, which also support parameters). The following is an applet tag that includes several parameters:

```
<applet code="ScrollingHeadline.class" height="50" width="400">

<param name="headline1" value="Dewey defeats Truman">

<param name="headline2" value="Stix nix hix pix">

<param name="headline3" value="Man bites dog">

</applet>
```

This example could be used to send news headlines to an applet that scrolls them across the screen. Because news changes all the time, the only way to create a program of this kind is with parameters.

You use the name attribute to give the parameter a name. This attribute is comparable to giving a variable a name. The value attribute gives the named parameter a value.

Receiving Parameters in the Applet

You have to do something in your Java program to retrieve the parameters on the web page or they will be ignored. The getParameter() method of the JApplet class retrieves a parameter from a param tag on a web page. The parameter name, which is specified with the name attribute on the page, is used as an argument to getParameter(). The following is an example of getParameter() in action:

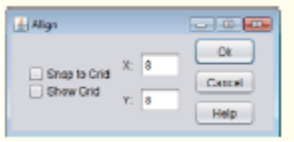
```
String display1 = getParameter("headline1");
```

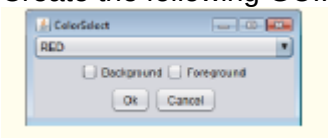
The getParameter() method returns all parameters as strings, so you have to convert them to other types as needed. If you want to use a parameter as an integer, you could use statements such as the following:

```
int speed;
String speedParam = getParameter("speed");
if (speedParam != null) {
   speed = Integer.parseInt(speedParam);
}
```

This example sets the speed variable by using the speedParam string. You have to test for null strings before setting speed because the parseInt() method cannot work with a null string. When you try to retrieve a parameter with getParameter() that was not included on a web page with the param tag, it will be sent as null, which is the value of an empty string.

Exercises

1. How are components arranged if I don't assign a layout manager to a container?
2. Create a frame that contains another frame and make both of them visible at the same time.
3. Create the following GUI. You do not have to provide any functionality.
4. 
5. Create the following GUI. You do not have to provide any functionality.
6. 

7. Create the following GUI. You do not have to provide any functionality.
8. 

9. Write an applet that asks the user to enter two floating-point numbers, obtains the two numbers from the user and draws their sum, product (multiplication), difference and quotient (division).
10. Write an applet that asks the user to enter two floating-point numbers, obtains the numbers from the user and displays the two numbers, then displays the larger number followed by the words "is larger" as a string on the applet. If the numbers are equal, the applet should print the message "These numbers are equal."

References

1. https://docs.oracle.com/javase/tutorial/
2. http://www.tutorialspoint.com/java/
3. http://www.javatpoint.com/java-tutorial
4. http://javabeginnerstutorial.com/core-java/