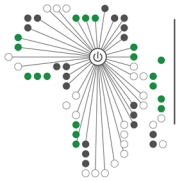


African Virtual University

Applied Computer Science: CSI 3102

OBJECT ORIENTED PROGRAMMING

LUCY GITAU

Foreword

The African Virtual University (AVU) is proud to participate in increasing access to education in African countries through the production of quality learning materials. We are also proud to contribute to global knowledge as our Open Educational Resources are mostly accessed from outside the African continent.

This module was developed as part of a diploma and degree program in Applied Computer Science, in collaboration with 18 African partner institutions from 16 countries. A total of 156 modules were developed or translated to ensure availability in English, French and Portuguese. These modules have also been made available as open education resources (OER) on oer.avu.org.

On behalf of the African Virtual University and our patron, our partner institutions, the African Development Bank, I invite you to use this module in your institution, for your own education, to share it as widely as possible and to participate actively in the AVU communities of practice of your interest. We are committed to be on the frontline of developing and sharing Open Educational Resources.

The African Virtual University (AVU) is a Pan African Intergovernmental Organization established by charter with the mandate of significantly increasing access to quality higher education and training through the innovative use of information communication technologies. A Charter, establishing the AVU as an Intergovernmental Organization, has been signed so far by nineteen (19) African Governments - Kenya, Senegal, Mauritania, Mali, Cote d'Ivoire, Tanzania, Mozambique, Democratic Republic of Congo, Benin, Ghana, Republic of Guinea, Burkina Faso, Niger, South Sudan, Sudan, The Gambia, Guinea-Bissau, Ethiopia and Cape Verde.

The following institutions participated in the Applied Computer Science Program: (1) Université d'Abomey Calavi in Benin; (2) Université de Ougagadougou in Burkina Faso; (3) Université Lumière de Bujumbura in Burundi; (4) Université de Douala in Cameroon; (5) Université de Nouakchott in Mauritania; (6) Université Gaston Berger in Senegal; (7) Université des Sciences, des Techniques et Technologies de Bamako in Mali (8) Ghana Institute of Management and Public Administration; (9) Kwame Nkrumah University of Science and Technology in Ghana; (10) Kenyatta University in Kenya; (11) Egerton University in Kenya; (12) Addis Ababa University in Ethiopia (13) University of Rwanda; (14) University of Dar es Salaam in Tanzania; (15) Universite Abdou Moumouni de Niamey in Niger; (16) Université Cheikh Anta Diop in Senegal; (17) Universidade Pedagógica in Mozambique; and (18) The University of the Gambia in The Gambia.

Bakary Diallo

The Rector

African Virtual University

Production Credits

Author

Lucy Gitau

Peer Reviewer

Victor Odumuyiwa

AVU - Academic Coordination

Dr. Marilena Cabral

Overall Coordinator Applied Computer Science Program

Prof Tim Mwololo Waema

Module Coordinator

Jules Degila

Instructional Designers

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Media Team

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

Gerisson Mulongo

Copyright Notice

Supported By

AVU Multinational Project II funded by the African Development Bank.

Table of Contents

Course Overview

Welcome to OBJECT ORIENTED PROGRAMMING

Object-oriented programming (OOP) has become exceedingly popular in the past few years. Software producers rush to release object oriented versions of their products. This module will introduce you to Introduction to object oriented programming , objects, classes, inheritance, interfaces, packages and Libraries and Handling of Exception. Each discussion focuses on how these concepts relate to the real world, while simultaneously providing an introduction to the syntax of the Java programming language.

This course enables the learners to develop simple desktop applications using java programming language.

Prerequisites

Structured programming and basic programming skills

Materials

The materials required to complete this course are:

- Introduction to object oriented programming with java , 5th edition Wu, C. Thomas, 2010
- Java software
- Computer and its applications

Course Goals

Upon completion of this course the learner should be able to:

- create and use objects to write programs in the Java Programming Language.
- use java libraries to manipulate objects.
- create simple desktop applications.

Units

Unit 0: Pre-Assessment

This unit will help the student to recall the evolution of programming languages and advantages of high level programminf languages in comparison to low level languages.

Unit 1: Introduction to object oriented programming

This unit defines object oriented programming, distinguishes it from structured programming and describes the key concepts of object oriented programming.

Unit 2: Classes , objects and class members

In this unit, you will we learn the concept of class and how to create method and objects. You also learn the concept of inheritance, abstraction and polymorphism.

Unit 3: Libraries and Exception handling

In this unit, you will learn how Java's exception handling mechanism works, how to try and catch exceptions, how to write exception classes and how to throw exceptions.

Unit 4: Introduction to GUI programming

In this unit, you learn about AWT and Swing components, applets and events. You also learn how to create simple desktop applications.

Assessment

Formative assessments, used to check learner progress, are included in each unit.

Summative assessments, such as final tests and assignments, are provided at the end of each module and cover knowledge and skills from the entire module.

Summative assessments are administered at the discretion of the institution offering the course. The suggested assessment plan is as follows:

1	Assignments	10%
2	Continous Assessment	20%
3	Final Exam	70%

Schedule

Unit	Activities	Estimated time
1	4 lecture activities and practical exercises	36 hrs
2	3 lecture activities and practical exercises	27 hrs
3	3 lecture activities and practical exercises	27 hrs
4	2 lecture activities and practical exercises	27 hrs

UNIT	UNIT TITLE	ACTIVITIES
UNIT 0	Pre-assessment	0.1 Fundamentals of programming
UNIT 1	Introduction to object oriented programming	1.1 What is OOP
		1.2 Introduction to Java
		1.3 Variables and Data Types
		1.4 Operations
UNIT 2	Classes, Objects and Members	2.1 Objects and Classes in Java
		2.2 Constructors
		2.3 How to implement abstraction, inheritance, generalization and polymorphism.
UNIT 3	Libraries and Handling of Exception	3.1 Java Packages
		3.2 Exception Handling
		3.3 IO streams
UNIT 4	Introduction to GUI programming	4.1 Creating Applets
		4.2 GUI programming
		4.3 Swing

Readings and Other Resources

The readings and other resources in this course are:

Unit 0

Required readings and other resources:

Introduction to object oriented programming with java , 5th edition Wu, C. Thomas, 2010

Unit 1

Required readings and other resources:

- Introduction to object oriented programming with java , 5th edition Wu, C. Thomas, 2010
- Java 2: The Complete Reference, Fifth Edition, Herbert Schildt, Tata McGraw Hill.

Unit 2

Required readings and other resources:

- Introduction to object oriented programming with java , 5th edition Wu, C. Thomas, 2010
- Java 2: The Complete Reference, Fifth Edition, Herbert Schildt, Tata McGraw Hill.

Unit 3

Required readings and other resources:

- Introduction to object oriented programming with java , 5th edition Wu, C. Thomas, 2010
- Java 2: The Complete Reference, Fifth Edition, Herbert Schildt, Tata McGraw Hill.

Unit 4

Required readings and other resources:

- Introduction to object oriented programming with java , 5th edition Wu, C. Thomas, 2010
- Java 2: The Complete Reference, Fifth Edition, Herbert Schildt, Tata McGraw Hill.

Unit 0. Pre-Assessment

Unit Introduction

The purpose of this unit is to determine your grasp of knowledge related to this course .

This unit requires you to know the brief history of programming languages from low-level machine languages to today's object oriented languages.

Unit Objectives

Upon completion of this unit you should be able to:

- state three levels of programming languages.
- describe the difference between low-level and high-level programming languages.
- discuss advantages of high-level programming languages].

Key Terms

Program: A set of instructions

Machine-language: A programming language where instructions are binary coded

Assembly language: A language that allows higher level symbolic programming by use of symbolic operation codes

Unit Assessment

Check your understanding!

Test

Instructions

1. This simple test is meant to help you recall the basic fundamentals of programming.
2. Answer the following questions:
3. Name three levels of programming languages. (3 Marks)
4. Explain why binary coded instructions are more difficult to use in programming.(1 Mark)

5. Compare and contrast assembly language with high-level programming languages.(2 Marks)
6. Give four examples of high-level programming languages. (4 Marks)
7. Provide the meaning of the following symbolic operation codes: MV, NUM, TOT, AC

Grading Scheme

Total mark is 14.

Feedback

1. machine, assembly and high-level languages
2. difficult for humans to work with machine codes
3. programs written in assembly language are not recognized by computers hence must be translated to machine language by an assembler. High level languages enables programmers to write programs faster but stiil not recognized by CPU. A compiler translates high-levl language to assembly language equivalents.
4. MV…...move , NUM…., TOT..TOTAL

Unit Readings and Other Resources

The readings in this unit are to be found at the course-level section "Readings and Other Resources"

Unit 1. Introduction to Object Oriented Programming

Unit Introduction

This unit introduces the student to object oriented paradigms. This unit will enable the student to differentiate between objected oriented programming and structured programming. The student should also be able to define the basic concepts of OOP.

Unit Objectives

Upon completion of this unit you should be able to:

- State the meaning of the term object oriented programming
- Discuss the differences between structured and object oriented programming.
- State the importance of object oriented programming.
- describe variables and show how variables are declared

Key Terms

Object orientation is a software engineering concept, in which concepts are represented as "objects"

Object: Objects are real world entities . A set or group of objects forms a class.

Variables: these are lables that express a particular position in memory and connect to it with a data type.

Constant: A fixed value that does not change during the execution of a program.

Learning Activities

Activity 1.1 what is object oriented programming:

Introduction

Object oriented programming is a programming approach in which all computations are done in the context of objects. A program written in an object oriented programming language can be seen as a collection of objects collaborating to perform a given task.

Why object orientation?

Object-orientation is a different way of looking at systems and data. Over the years we have become used to the separation of processes (represented by systems and programs) and data (represented by databases and files). This is not a natural division; this has largely arisen because of the limitation of previous hardware and software technology. Traditionally there was no enough memory to load the program code, but not enough to load data in its entirety. This meant that we had to leave the bulk of our data on secondary storage (e.g. disk) and load only small portions of it into real memory to work on at a time. Typically a block of records would be moved between disk and memory with one input or output operation. This also placed on the programmer the burden of devising the logic to make files accessible (by opening them), reading in the necessary records required, and matching the position on various related files. Unfortunately this meant that the programmer (and the program) had to know a lot about the implementation of the files and their structure. This is problematic from a maintenance perspective, since programs have to be changed if we decide to alter the data implementation. Since multiple programs access the same data, the possibility arises that they try to do so in different ways, thus corrupting the structure or content for other programs.

With larger real memories, and good virtual memory management techniques that can make storage limitations transparent to a programmer, the separation of code and data is no longer necessary.

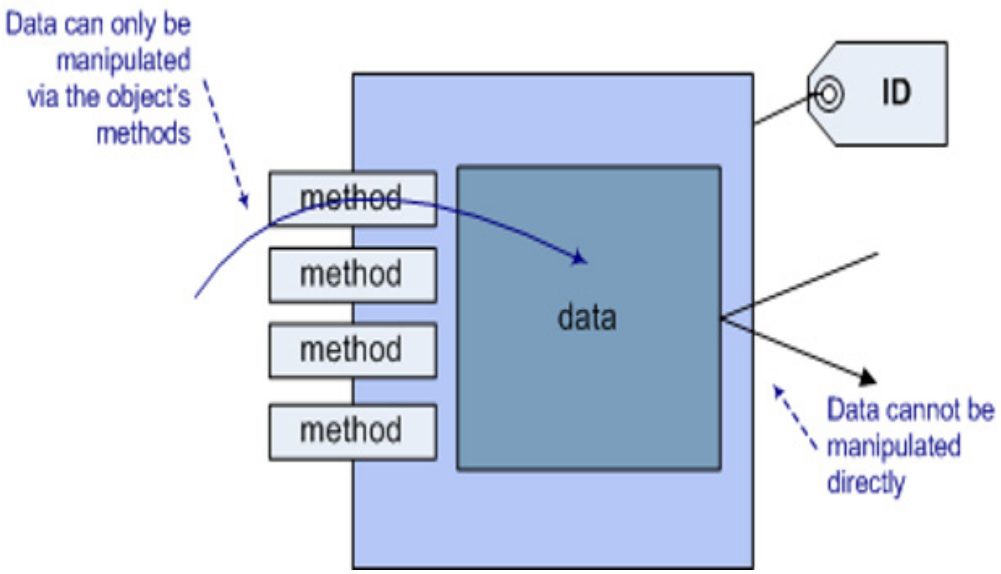
Object Orientation narrows the gap between problem and solution models as it enables us to represent solution domain in terms of problem domain elements. The problem domain elements include real world objects that interact to accomplish a given task. For example, a library system deals with librarians, borrowers, books, and their interactions. While we model the problem, we model each problem domain element with their states and behaviors and identify their interactions. For example, any system to automate the operations in a library should deal with the objects in the problem domain i.e. librarian, library member, book, journal, magazine etc.

In a school scenario, we would consider the following as objects: student, department, course, lecturer etc

Objects sometimes correspond to things found in the real world. For example, a graphics program may have objects such as “circle,” “square,” “menu.” The graphics program will support behaviors’ such as draw, compute area, computer perimeter etc. An online shopping system will have objects such as “shopping cart,” “customer,” and “product.” The shopping system will support behaviors such as “place order,” “make payment,” and “offer discount.”

Good, structured, modular programming dictates that we should have small, highly cohesive modules that have minimum interaction with other modules (high cohesion, low coupling). Object-orientation takes this concept to the ultimate conclusion by combining the data structures and the code that will manipulate the data. So, instead of a customer file and several programs that may access and update it, we will have a customer object that will contain the customer attributes (data items) as well as methods that perform the necessary processing of customer data. This is called encapsulation. If encapsulation is enforced, it means that objects don’t know or care how other objects store or process data. It means that objects are safe.

What is an object?



An Object is a structure containing both data and methods (the object's interface) that operate on the data. It has state and identity. Only methods within the object can manipulate data within the object hence the object is safe.

Activity 1.2 Introduction to java

Java is a general-purpose, object-oriented programming language developed by Sun Microsystems of USA in 1991.It was originally called Oak by James Gosling (one of the inventor of the language). Java was invented for the development of software for consumer electronic devices like TVs, tosters, etc. The main aim was to make java simple, portable and reliable.

Java Authors: James, Arthur Van, and others

Java is first programming language which is not attached with any particular hardware or operating system. Programs developed in Java can be executed anywhere and on any system.

Java comes with a compiler called java compiler which unlike other programming languages convert the source code to intermediate code called byte code.

The virtual machine code (byte code) is not machine specific. The machine specific code is generated by java interpreter which acts as an intermediary between virtual machine and real machine.

Java environment includes a number of development tools, classes and methods. The development tools are part of the system known as Java Development Kit (JDK) and the classes and methods are part of the Java Standard Library (JSL), also known as the Application Programming Interface (API).

The JDK comes with a set of tools that are used for developing and running Java programs. It includes:

1. Applet viewer (used for viewing the applet)
2. Java (this is a Java Compiler)
3. Java (It is a java interpreter)
4. Javap (Java diassembler, converts byte code into program description)
5. Javah (for java C header files)
6. Javadoc (for creating HTML document)

Jdb (Java debugger)

Important commands used in running and compiling the java programs:

a) javac (Java compiler)

In java, we can use any text editor for writing program and then save that program with .java extension. Java compiler convert the source code or program in bytecode and interpreter convert .java file in .class file.

To compile a java program use the following syntax:

```
C:\javac filename.java
```

```
Egg. If my filename is abc.java, then the syntax will be
```

```
C:\javac abc.java
```

b) Java (Java Interpreter)

To run the program, use the following syntax:

```
C:\java filename
```

```
E.g. If my filename is abc.java then the syntax will be
```

```
C:\java abc
```

Java Libraries:

Exploring the Java class libraries and their methods and instance variables is a great way to figure out what Java can and cannot do, as well as a starting point for your own development. Here are the class packages that are part of the Java class libraries:

java.lang: Classes that apply to the language itself, which includes the Object class, the String class and the System class. It also contains the special classes for the

Primitive types (Integer, Character, Float, and so on).

java.util: Utility classes, such as Date, as well as simple collection classes, such as

Vector and Hashtable.

java.io: Input and output classes for writing to and reading from streams (such as

Standard input and output) and for handling files.

java.net: Classes for networking support, including Socket and URL (a class to represent references to documents on the World Wide Web).

java.awt: (the Abstract Window Toolkit): Classes to implement a graphical user interface, including classes for Window, Menu, Button, Font, CheckBox, and so on. This package also includes classes for processing images (the java.awt.Image package).

java.applet: Classes to implement Java applets, including the Applet class itself, as well as the Audio Clip class.

In addition to the Java classes, your development environment may also include additional classes that provide other utilities or functionality. Although these classes may be useful, because they are not part of the standard Java library, they won't be available to other people trying to run your Java program. This is particularly important for applets, because applets are expected to be able to run on any platform, using any Java-aware browser. Only classes inside the java package are guaranteed to be available on all browsers and Java environments.

Components of a java program

A java program is composed of:

- comments
- import statements
- class declaration

Comments: Here we state the purpose of the program, explain the meaning of the code and provide any descriptions to help programmers understand the program.

Comments start with /* at the beginning of the comment and terminate the comment with */ this is commonly used in multiline comments and is important for program documentation.

In single line comments, we use double slashes i.e. // both at the beginning and at the end of the comment.

import statement used to develop Object Oriented programs from predefined classes but it is also possible to define our own classes. Normally, defined classes are grouped into packages. The import statement allows the program to use classes (and their instances) defined in the designated package. A package can include sub packages e.g. java.awt.image.colormodel. Import statement is terminated by a semicolon. I.e. import <package name>.*;

A simple java program:

```
/* Program Hello World

This program displays the statement ''Hello all, this is my first java program''.*/
```

```
import javabook.*;

class Hello World

{

public static void main (string args[])

{

system.out.printIn (''Hello all, this is my first java program'');

}

}
```

The name of the file is HelloWorld. This is equivalent to the class name containing the main method.

Java is case sensitive. This program defines a class called HelloWorld. A class is an object oriented term which is designed to perform a specific task. A Java class is defined by its class name, an open curly brace, a list of methods and fields, and a close curly brace. The name of the class is made of alphabetical characters and digits without spaces, the first character must be alphabetical.

The line public static void main (String [] args) shows where the program will start running. The word main means that this is the main method.

The JVM starts running any program by executing this method first. The main method in Hello World consists of a single statement

System.out.println("Hello all, this is my first program"); The statement outputs the character between quotes to the console.

We can also have the same statement displayed in a window almost as big as a screen.

See the example below and compare with the simple java program shown above:

```
/* Program Hello World,

This program displays a window on the screen. The window is positioned
at the center of the screen and the size of window is almost as big as
the screen.*/


import javabook.*;

Class Hello Word

{

      public static void main (string []args)

            {
```

```
            MainWindow main Window;
            Main Window= new MainWindow ();
            mainWindow.setvisible(true);
            }
}
```

Activity Details

Write a program in java to display the following shopping list in the console window.

Hint: Don't forget to include blank spaces so the item names appear indented. (\n)

Shopping List:

Apple

Banana

Low-fat Milk

Conclusion

Object-Oriented Programming has the following advantages over conventional approaches:

1. OOP provides a clear modular structure for programs which makes it good for defining abstract data types where implementation details are hidden and the unit has a clearly defined interface.
2. OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
3. OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer.

How to write, compile and run a java program:

1. Write and edit the program by the use of Notepad.
2. Save the program to the hard disk.
3. Compile the program with the javac command.(Java compiler)
4. If there are syntax errors, go back to Notepad and edit the program.
5. Run the program with the java command.(Java Interpreter)
6. If it does not run correctly, go back to Notepad and edit the program.
7. Compile and run the program again.

Assessment

Define object oriented programming and outline its advantages and disadvantages.

Explain the meaning of the term encapsulation with regard to object structure.

Activity 1.3 - Variables and data types

Introduction

A variable is as a named container for data (and objects). Memory space is allocated for a variable at runtime and its name is used to get or set its value. You must declare a variable before using it by giving it a name and type. A variable can be declared in mainly four different contexts: within a class as a member variable (outside any methods), within method declaration as a method parameter, within a method as a local parameter, and within an exception handler as an exception handler parameter. (read more on this concept in unit four)

Activity Details

Suppose you want to compute the sum and difference of two numbers. Say the

two numbers are x and y. In mathematics, we say

x + y

and

x – y

To compute the sum and the difference of x and y in a Java program, you must first declare what kind of data will be assigned to them. After assigning values to them, compute their sum and difference.

Let's say x and y are integers. To declare that the type of data assigned to them

is an integer, we write

int x, y;

When this declaration is made, memory locations to store data values for x and y are allocated. These memory locations are called variables, and x and y are the names we associate with the memory locations. Any valid identifier can be used as a variable name. After the declaration is made, we can assign only integers to x and y.

How variables are declared

The general syntax for declaring variables is

```
<data type> <variables> ;
```

where <variables> is a sequence of identifiers separated by commas. Every variable we use in a program must be declared. It is possible to have as many declarations as you wish. For example, declare x and y separately as

```
int x;

int y;
```

However, you cannot declare the same variable more than once; therefore, the second declaration below is invalid because y is declared twice:

```
int x, y, z;

Int y;
```

Data types

There are six numerical data types in Java: byte, short, int, long, float, and double. The data types byte, short, int, and long are for integers; and the data types float and double are for real numbers. The data type names byte, short, and others are all reserved words. The difference among these six numerical data types is in the range of values as shown in the table below:

Data type	Content	Default value	minimum value	maximum value
byte	Integer	0	-128	127
short	Integer	0	-32768	32767
int	Integer	0	-2147483648	2147483647
long	Integer	0	-9223372036854750000	9223372036854770000
float	Real	0	-3.40282347E+38‡	3.40E+38
double	Real	0	-1.79769313486231570E+308	1.79769313486231570E+308

A data type with a larger range of values is said to have a higher precision. For example, the data type double has a higher precision than the data type float. The tradeoff for higher precision is memory space—to store a number with higher precision, you need more space. Avariable of type short requires 2 bytes and a variable of type int requires 4 bytes, for example. If your program does not use many integers, then whether you declare them as short or int is really not that critical. The difference in memory usage is very small and not a deciding factor in the program design.

The storage difference becomes significant only when the program uses thousands of integers. Therefore, you can always use the data type int for integers.

We use long when we need to process very large integers that are outside the range of values int can represent. For real numbers, it is more common to use double although double requires more memory space than float, but it is preferable because of its higher precision in representing real numbers.

How to declare variables of different data types:

```
int       i, j, k;
float     numberOne, numberTwo;
long      bigInteger;
double    bigNumber;
```

At the time a variable is declared, it also can be initialized. For example, we may initialize the integer variables count and height to 10 and 34 as in

```
int count = 10, height = 34;
```

Conclusion

A variable has three properties:

- a memory location to store the value,
- the type of data stored in the memory location, and
- the name used to refer to the memory location.

Unit Assessment

1. What do you understand by the term variable?
2. Outline three properties of a variable.
3. Use a snippet code to demonstrate how a variable can be declared.
4. If you have the source code for a Java program, and you want to run that program, you will need both a compiler and an interpreter. What does the Java compiler do, and what does the Java interpreter do?
5. Give the meaning of each of the following Java operators:

   - ++
   - &&
   - !=

Activity 1.4 -OPERATORS

Introduction

This activity will help you be able to distinguish between the different types of operators

Activity Details

Operators perform a function on its operands to produce a value. Operators behave like predefined functions in the Java programming language. For, example the expression A+B, which is built from the operands A and B, and the operator + If A and B both are of type int, A+B returns their sum.

An operator may require one, two or three operands. An operator requiring one operand is called a unary operator, requiring two operands is called a binary, and requiring three operators is called a ternary operator. Most of the operators operate on expressions returning primitive data types. Here expression refers to variables, literals, value of a primitive type returned from a method, or other expressions formed by operators. Only the operators =, ==, and != work also with reference types and + and += works also on String objects. There are six basic kinds of operators: arithmetic, logical, bitwise, assignment, comparison, and ternary if-else operators.

It is also possible to combine expressions formed by operators to form compound expressions with more than one operator.

Precedence rules are applied to determine the order of evaluation in a compound statement built using more than one operator. For example, * and / are evaluated before + and -.

Types of operators:

Arithmetic operator can perform mathematical functions on numeric data types (Integers and Decimal Numbers).

Operator	Description
+	Addition operator
-	Substraction operator
*	Multiplication operator
/	Division operator
%	Remainder operator

Comparison operators compare the values of two operands and produce a boolean result.

Logical operators operate on boolean values to perform boolean operations like NOT, AND, OR, or XOR (exclusive or).

There is only one ternary operator, ?: in the Java programming language. The expression A?B:C returns the value of B or C depending on the value of A. The first operand, A, must be an expression that returns a boolean value, and if A is true then B is evaluated and its value will be the result of the operation. If A is false then C is evaluated its value will be the result of the operation. Note that, depending on the value of A, one of the operands (B or C) may not be evaluated

Operator	Description
+	unary plus; indicates positive value
-	unary minus operator; negates an expression
++	increment operator; increments by a value of 1
--	decrement operator; decrements by a value of 1
!	logical complement operator; inverts the value of a boolean

Bitwise operators perform bit by bit operations, such as NOT (~), AND (&), OR (|), XOR (^), or shift (left or right), on operands. Operands in these operators must be integers (& and | also operate on booleans as we have seen before), and operations are carried out on binary representations of the values of operands. Shift operators shift the bits of the first operand to left or right by distance specified in the second operand, and fills the empty bits with 0 (except right shift operator >>).

bit 1	bit 2	OR(|)	AND(&)	XOR(^)
0	0	0	0	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	0

Statements

A statement is a single command in a program. An individual statement ends with a semicolon (;). A single statement may cover multiple lines of code, but everything up to semicolon is processed as a single command. A statement may

- Declare a variable,
- Create a reference data type,

- Assign a value to a variable,
- Increment or decrement a variable (using ++ or -- operators)
- Call a method,
- Complete the execution of a method (return return_value;).

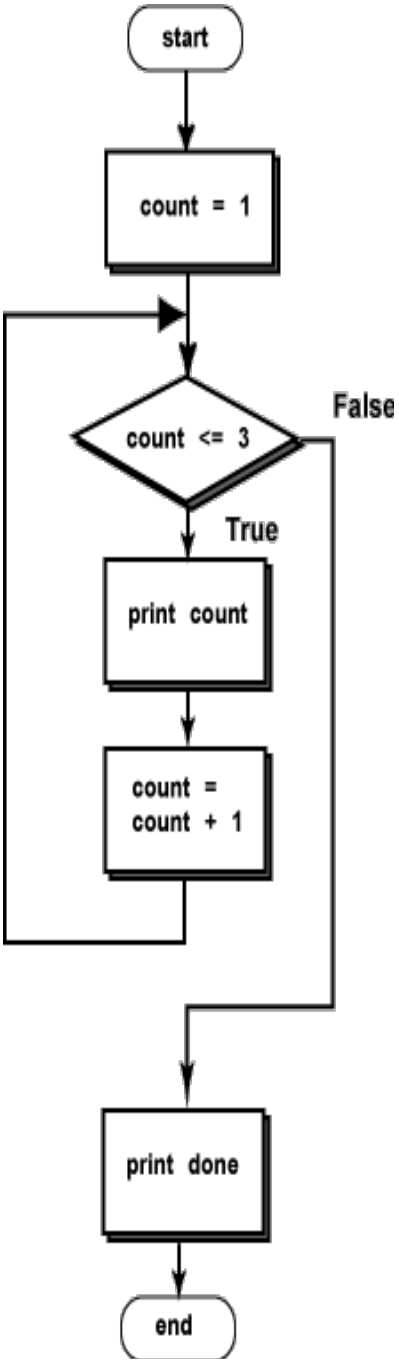
Multiple statements may be enclosed by curly braces ({ and }) to form compound statements. Such statements can be used where only one statement is allowed (in execution control statements that will be explained). The following example illustrates a simple statement block

```
...
{
int i = 0;
i += i+1;
String s = i.toString();
s = "2*2=" + s;
}
...
```

While the program executes, statements are executed in the order in which they are written. However, It is possible to control the execution of statements according to some condition using control flow statements. There are three main groups of control flow statements: Loop statements, conditional statements, and branching statements. Since loop and conditional statements are themselves also statements, they can be nested in any depth (they can contain other control flow statements in their body).

Loop Statements

Loop statements enable execution of the same statement (or statements) zero or more times according to a given condition expression. There are three kinds of loop statements in the Java programming language: while, do-while, and for. A while statement executes a single statement or a statement block (if you want to execute more than one statements) as long as the condition given to it is true. The condition expression must return a boolean value. The format of while statement is as follows:



The flowchart shows how the program works. First, count is set to one. Then it is tested by the while statement to see if it is less than or equal to three.

The test returns true so the statements in the block following the while are executed. The current value of count is printed, and count is incremented. Then execution goes back to the while statement and the test is performed again.

Count is now two, the test returns true and the block is executed again. The last statement of the block increments count to three, then execution goes back to the while statement.

Count is now three, the test returns true and the block is executed again. The last statement of the block increments count to four, then execution goes back to the while statement.

After the block has executed three times, count is four. Execution goes back to the while

statement, but now the test returns false, and execution goes to the "Done with loop" statement. Then the program ends.

Syntax for while statements in java:

```
...
while (condition) statement;
// or
while (condition){
statement 1;
...
statement n;
}
```

Application areas in the use of machines that use cycles:

- Bicycle — your legs drive the pedals connected to a gear which spins.
- CD Player — the disk spins (cycles) as the laser moves across it.
- TV Set — pictures are put one the screen one after another as long as the set is on.
- Water Pump — often a piston repeatedly moves in and out of a cylinder.
- Laundry Dryer — rotating drum.
- Clock — shows the same times every day. If the clock is mechanical, its insides are gears and springs with many mechanical cycles. If the clock is electronic the cycles are still there, but harder to see.
- Sun and the Earth — endlessly cycling, seasons flowing one into the next.

A do-while statement does a similar execution. It executes one statement or statement block as long as the condition given to it is true. However, it differs from while statement in that the statement(s) following do is executed at least once, whereas the statement(s) following while may not be executed depending on the value of the condition expression. The format of the do-while statement is as follows:

```
...
do statement while(condition);
// or
do {
statement 1;
...
```

```
statement n;
} while (condition);
...
```

A for statement is used to iterate over a range of values. The format of the for statement is as follows:

```
...
for(initialization;condition;iteration) statement;
// or
for(initialization;condition;iteration){
statement 1;
...
statement n;
};
...
```

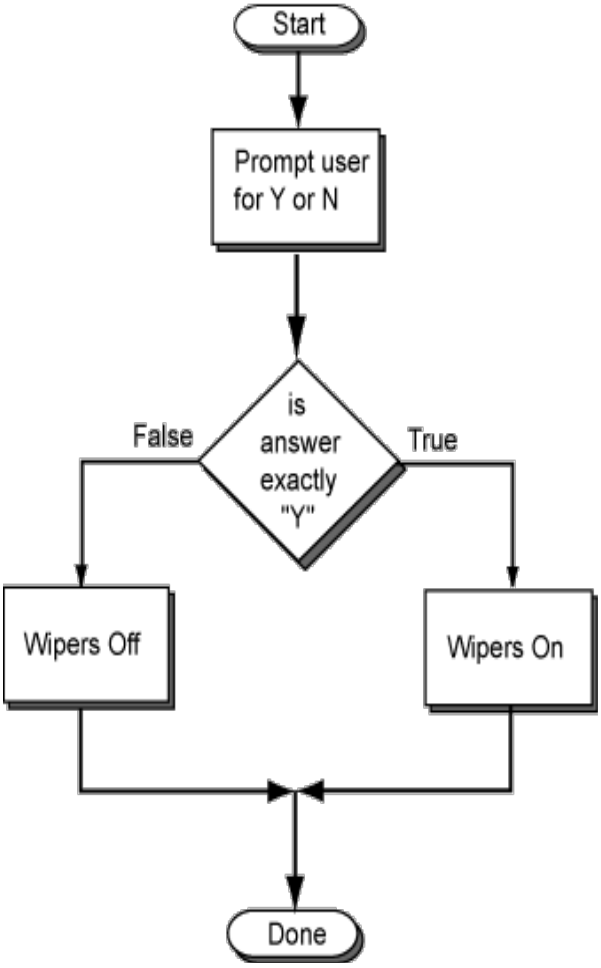
Here initialization is a statement that is executed at the beginning once. It may contain a variable declaration and initialization, or it may only initialize a control variable. iteration statement is executed each time the following statement(s) is executed until the condition expression evaluates a false.

Conditional statements

Conditional statements enable deciding which statement (or statements) to execute according to the given condition expression. There are three kinds of loop statements in the Java programming language: if, if-else, and switch. An if statement is used to decide whether to execute or not a statement or a statement block. If the condition is true, the statement(s) is executed, otherwise, the execution continues from the next statement following if.

The format of an 'if' statement

The format of the if statemWWent is further illustrated in the flowchart :



The if statement

if (answer.equals("Y")) picks either the "true branch" or the "false branch" . Only one branch or the other is executed, just as in the flow chart. This part of the statement answer.equals("Y") evaluates to true if the string referenced by answer contains exactly the single character "Y"however, for anything else it evaluates to false.

syntax for if statements:

```
…
if(condition) statement;
// or
if(condition){
statement 1;
...
statement n;
}
...
```

Example two: Consider the following segment of a program that is written for processing marks obtained in an entrance examination.

```
……..
……..
if( category== sports)
{
marks=marks + bonus_marks:
}
system.out.printIn(marks):
……..
……..
```

The program tests the type of category of the student. If the student belongs to category of Sports, additional bonus_marks are added to his marks before they are printed otherwise no marks are added.

Consider a case of two test conditions, one for weight and another for height. This is done using the compound relation.

```
if(weight < 50 && height > 170) count = count +1;
```

alternatively you can use two if statements:

```
if(weight < 50)
if(height >170)
count = count +1 ;
```

The condition expression must return a boolean value. A do-while statement does a similar execution.

An if-else statement is used to decide which statement or statement block to execute. If the condition is true, the first statement or statement block is executed, otherwise the second statement or statement block is executed. The format of the if else statement is as follows:

if(condition) statement1; else statement2;

```
// or
if(condition){
statement 1;
...
statement 1_n;
```

```
} else
 {
statement 2_1;
...
statement 2_m;
}
```

The else part may also contain other if statements:

```
if(condition 1)
{
statement 1_1;
...
statement 1_n;
}
```

else if (condition 2)

```
{
statement 2_1;
...
statement 2_m;
}
```

else if (condition 3)

```
{
statement 3_1;
...
statement 3_k;
} else
     {
statement 4_1;
...
statement 4_t;
}
```

A switch statement can be used to choose a statement or statements to execute according to an integer or character value (coming from a variable or expression). The format of the switch statement is as follows:

switch(expression)

```
{
case <value1>:
statement 1;
break;
case <value2>:
statement 2;
break;
...
case <value n>:
statement n;
break;
default:
statement n+1;
}
```

Where, expression produces an integer or character value, and the statement to execute is determined according to its value. For example, if its value equal to <value 2> the section case <value 2> is executed until the break statement. The break statement causes termination of switch block and execution continues with the next statement following switch block. When none of the case <value i> lines match, the default part is executed (if not required default section may be omitted).

It is possible to combine multiple case sections as: switch(expression)

```
{
case <value1>:
case <value2>:
case <value3>:
statement 1;
break;
case <value4>:
```

```
case <value5>
statement 2;
break;
...
case <value n>:
statement n;
break;
default:
statement n+1;
}
```

In this case, if expression is equal to one of <value1>, <value2> or <value3>, the statement 1 is executed.

do while statements

The while statement is characterized as a pretest loop because the test is done before execution of the loop body. Because it is a pretest loop, the loop body may not be executed at all. The do–while is a repetition statement that is characterized as a posttest loop. With a posttest loop statement, the loop body is executed at least once.

The general format for the do–while statement is

do

<statement>

while (<boolean expression>) ;

The <statement> is executed until the <boolean expression> becomes false. Remember that <statement> is either a <single statement> or a <compound statement>.

Example. Supposing you want to add the whole numbers 1, 2, 3, . . . until the sum becomes larger than 1,000,000. The snippet code would appear as follows ie using a do–while statement:

```
int sum = 0, number = 1;
do {
sum += number;
number++;
} while ( sum <= 1000000 );
```

Logical operators

The if, if...else, while, do...while and for statements each require a condition to determine how to continue a program's flow of control. Simple conditions are expressed in terms of relational operators <, >, >=, <= and != and each tests a single condition. To test multiple conditions in the process of making a decision , we perform tests in separate statements. Java provides logical operators to enable programmers to form more complex conditions by combining simple conditions. The logical operators are && (conditional AND), || (Conditional OR), and !(logical NOT)

EG.

To ensure two conditions are both true, we use conditional AND operator

```
if ( gender == female && age >= 65)
      ++seniorfemales; //increment senior females by 1.
```

conditional AND && operator

expression 1	expression 2	expression 1 && expression2
False	False	false
False	True	false
True	False	false
True	True	true

Conditional OR || operator

```
if ( (semesterAverage >= 90) || (finalExam >= 90))
      system.out.printIn('' student grade is A'');
```

expression 1	expression 2	expression 1 || expression 2
True	True	true
true	False	true
False	True	true
False	False	false

Conclusion

Repetition control statements are used to repeatedly execute a group of code until a certain condition is met. Logical operators enable programmers to make complex decisions by combining simple conditions.

Unit Assessment

1. Write a do–while loop to compute the sum of the first 30 positive odd integers.
2. Rewrite the following while loops as do–while loops.

a.
```
int count = 0, sum = 0;
while ( count < 10 ) {
sum += count;
count++;
}
```

b.
```
int count = 1, sum = 0;
while ( count <= 30 ) {
sum += count;
count += 3;
}
```

Unit Summary

This unit introduces the students to conditional statements and loop statements. It also provides real life examples where such statements can be applied and finally provides syntax in each case.

An exercise on Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

Assume integer variable A holds 10 and variable B holds 20, complete the table below

Grading Scheme

Each answer carries 1 mark therefore the total score is seven.

Feedback

30, -10, 200 , 2, 0, 21, 19

Operator	Description	Answer
+	adds value on either side of an operator	
-	substracts right hand operand from left hand operand	
*	multiplies values on either side of an operator	
/	divides left hand operand by right hand operand	
%	divides left hand operand by right hand operand and returns the remainder	
++	increases value of operand by 1	
--	decreases value of operand by 1	

Unit Assessment

Check your understanding!

Differentiate between arithmetic and relational operators.

Suggest examples of real life experiences other than the ones above where 'while', 'if else' and switch statements may be used.

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Unit 2. Classes, objects and members

Unit Introduction

Java is an Object-Oriented Language. As a language it has the Object Oriented feature and supports the following fundamental concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Parsing

This unit will demonstrate the concept of classes and objects as well as the object oriented programming design concepts e.g. abstraction, encapsulation, inheritance and polymorphism.

Unit Objectives

Upon completion of this unit you should be able to:

- state the meaning of a class, object and class member
- define the term constructor
- describe the three steps followed when creating an object from a class
- create programs that demonstrate the concepts above

Key Terms

Class: A class is a collection of objects of a similar type. Once a class is defined, any number of objects can be created which belong to that class. A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind. Class and object Members: A class member is either a data value or a method.

Abstraction: Abstraction refers to the act of representing essential features without including the background details or explanations.

Encapsulation: It is the mechanism that binds together code and data in manipulates, and keeps both safe from outside interference and misuse.

Polymorphism: Polymorphism is the ability to take more than one form. An operation may exhibit different behaviours in different instances. The behaviour depends on the data types used in the operation.

Inheritance: It is the process by which one object acquires the properties of another object. This supports the hierarchical classification.

Learning Activities

Activity 2.1 - Objects and Classes in Java

Introduction

Many programs are written to do things that are concerned with the real world. It is convenient to have software objects that are similar to real world objects. This makes the program and what it does easier to think about. Software objects have identity, state and behaviour just like real world objects.

When a Java application is run, objects are created and their methods are invoked. To create an object, you need to describe it. A class is a description of a kind of object. A programmer may define a class using Java or may use predefined classes that come in class libraries. A class is merely a plan for a possible object. It does not by itself create any objects. When a programmer wants to create an object the new operator is used with the name of the class. Creating a class is called instantiation.

Activity Details

Objects in Java

Let us now look deep into what are objects. If we consider the real-world we can find many objects around us, Students, Cars, Dogs, Humans, etc. All these objects have a state and behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging, running

If you compare the software object with a real world object, they have very similar characteristics.

Software objects also have a state and behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

Classes in Java

A class is a blue print from which individual objects are created.

A sample of a class is given below:

```
public class Dog
{
  String breed;
  int age;
  String color;
  void barking(){
  }
   void hungry(){
  }
   void sleeping(){
  }
}
```

A class can contain any of the following variable types.

- Local variables: Variables defined inside methods, constructors or blocks are called local variables. The local variable is declared and initialized within the method and destroyed when the method is complete.
- Instance variables: Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- Class variables: Class variables are variables declared with in a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods.

In the above example, barking(), hungry() and sleeping() are methods.

How to create an object from a class:

As mentioned above, a class provides the blueprints for objects. So basically an object is created from a class. In Java, the 'new' key word is used to create new objects.

There are three steps used when creating an object from a class:

- Declaration: A variable declaration with a variable name with an object type.
- Instantiation: The 'new' key word is used to create the object.
- Initialization: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example of creating an object is given below:

```
public class Puppy{

  public Puppy(String name){

      // This constructor has one parameter, name.

      System.out.println("Passed Name is :" + name );

  }

  public static void main(String []args){

      // Following statement would create an object myPuppy

      Puppy myPuppy = new Puppy( "tommy" );

  }

}
```

If we compile and run the above program, then it would produce the following result:

Passed Name is :tommy

Visibility Modifiers:

Visibility modifiers designate the accessiblity of data members. There are four main types of visibility modifiers public, private, protected and package. If an instance variable is declared private the no outside methods can directly access it. If declared public, then any outside method can access it. Declaring data members private ensures the integrity of the class. In the example above, the class puppy can be accessed by other methods.

Access Control Modifiers:

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

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

Non Access Modifiers:

Java provides a number of non-access modifiers to achieve many other functionality

- The static modifier for creating class methods and variables
- The final modifier for finalizing the implementations of classes, methods, and variables.
- The abstract modifier for creating abstract classes and methods.
- The synchronized and volatile modifiers, which are used for threads.

Conclusion

In this section, you learnt how to declare, instantiate and initialize a class. You also learnt that there are three variable types in a class: local, instance and class variables. Lastly you learnt about visibility, how to implement visibility and the reason behind visibility.

Assessment

Consider the following class:

```
public class IdentifyMyParts {

    public static int x = 7;

    public int y = 3;

}
```

1. What are the class variables?
2. What are the instance variables?
3. What is the output from the following code:

```
IdentifyMyParts a = new IdentifyMyParts();

IdentifyMyParts b = new IdentifyMyParts();

a.y = 5;
```

```
b.y = 6;

a.x = 1;

b.x = 2;

System.out.println("a.y = " + a.y);

System.out.println("b.y = " + b.y);

System.out.println("a.x = " + a.x);

System.out.println("b.x = " + b.x);

System.out.println("IdentifyMyParts.x = " + IdentifyMyParts.x);
```

Activity2.2 - Constructors

Introduction

A constructor is a special method that is executed when a new object is created. Its purpose is to initialize the object into a valid state. A constructor can be thought of as a specialized method of a class that creates (instantiates) an instance of the class and performs any initialization tasks needed for that instantiation. The sytnax for a constructor is similar but not identical to a normal method and adheres to the following rules:

The name of the constructor must be exactly the same as the class it is in.

Constructors may be private, protected or public.

A constructor cannot have a return type (even void). This is what distinguishes a constructor from a method. A typical beginner's mistake is to give the constructor a return type and then spend hours trying to figure out why Java isn't calling the constructor when an object is instantiated.

Multiple constructors may exist, but they must have different signatures, i.e. different numbers and/or types of input parameters.

The existence of any programmer-defined constructor will eliminate the automatically generated, no parameter, default constructor. If part of your code requires a no parameter constructor and you wish to add a parameterized constructor, be sure to add another, no-parameter constructor.

Example

```
public Person {

private String _name;

public Person(String name) {//constructor

name = name;

}

}
```

Activity Detail

Every class has a constructor. If we do not explicitly write a constructor for a class the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Example of a constructor is given below:

```
public class Puppy{

  public puppy(){

  }

  public puppy(String name){

      // This constructor has one parameter, name.

  }}
```

Abstract classes

An abstract class is a class defined with the modifier abstract, and no instances can be created from an abstract class.

```
abstract class Student {//The keyword abstract here denotes an abstract class.

protected final static int NUM_OF_TESTS = 3;

protected String name;

protected int[] test;

protected String courseGrade;

public Student() {

this(“No name”);

}

public Student(String studentName) {

name = studentName;

test = new int[NUM_OF_TESTS];

courseGrade = “****”;

}
```

abstract public void computeCourseGrade();//The keyword abstract here denotes an abstract method. Abstract method has no method body, just a semicolon.

Abstract class

```
public String getCourseGrade() {

return courseGrade;

}

public String getName() {

return name;

}

public int getTestScore(int testNumber) {

return test[testNumber-1];

}

public void setName(String newName) {

name = newName;

}

public void setTestScore(int testNumber,int testScore){

test[testNumber-1] = testScore;

}

}
```

An abstract method is a method with the keyword abstract, and it ends with a semicolon instead of a method body. A class is abstract if the class contains an abstract method or does not provide an implementation of an inherited abstract method.

Conclusion

Constructors are member functions which initialize a class and have no returntype. They are called automatically whenever a new instance is created.

Assessment

1. Which of the following constructors are invalid?

```
public int ClassA(int one) {

...

}

public ClassB(int one, int two) {

...
```

```
}
void ClassC( ) {
...
}
```

2. What is the main purpose of a constructor?

3. Complete the following constructor.

```
class Test {
private double score;
public Test(double val) {
//assign the value of parameter to
//the data member
}
}
```

Activity 2.3 - How to Implement Abstraction, Inheritance, Generalization and Polymophism

Introduction

Paradigms of Object Oriented Programming

Abstraction

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes.

Encapsulation

It is the mechanism that binds together code and data in manipulates, and keeps both safe from outside interference and misuse. In short, it isolates a particular code and data from all other codes and data. A well-defined interface controls the access to that particular code and data. The act of placing data and the operations that perform on that data in the same class. The class then becomes the 'capsule' or container for the data and operations.

Storing data and functions in a single unit (class) is encapsulation. Data cannot be accessible to the outside world and only those functions which are stored in the class can access it.

Inheritance

It is the process by which one object acquires the properties of another object. This supports the hierarchical classification. Without the use of hierarchies, each object would need to define all its characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. A new subclass inherits all of the attributes of all of its ancestors.

Polymorphism

Polymorphism means the ability to take more than one form. An operation may exhibit different behaviours in different instances. The behaviour depends on the data types used in the operation. It is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. In general, polymorphism means "one interface, multiple methods", This means that it is possible to design a generic interface to a group of related activities. This helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler's job to select the specific action (that is, method) as it applies to each situation.

Generalization

Generalization describes an is-a relationship which represent a hierarchy between classes of objects. Eg:- a "fruit" is a generalization of "apple", "orange", "mango" and many others.

Specialization

Specialization means an object can inherit the common state and behavior of a generic object. However, each object needs to define its own special and particular state and behavior. Specialization means to subclass. animal is the generalization and pet is the specialization, indicating that a pet is a special kind of animal.

Activity Details

Define the term inheritance? It is the process where one object acquires the properties of another.

How the keywords extend and implement are used to achieve inheritance? First, this words determine whether IS-A type of another. By using this words, it is possible to make one object acquire the properties of another object. e.g. A mammal is a type of animal, a dog is a type of mammal, a reptile is a type of animal.

```
public class Animal {}

public class Mammal extends Animal {}

public class Reptile extends Animal{}

public class Dog extends Mammal{}
```

Note that the general class is known as the superclass in this case ANIMAL while the specialized classes are subclasses e.g. mammal, dog and reptile. Subclasses inherit all the properties of a superclass except the private properties of a superclass.

```
public class Dog extends Mammal{

public static void main(String args[]){

Animal a = new Animal();

Mammal m = new Mammal();

Dog d = new Dog();

System.out.printIn(m instanceof Animal);

System.out.printIn(d instanceof Mammal);

System.out.printIn(d instanceof Animal);

}

}
```

What results would the snippet code above produce i.e true or false.

Keyword implement is used by classes by inheriting from interfaces. Interfaces can never be extended by the classes.

Example:

```
public interface Animal {}

public class mammal implements Animal{

}

public class Dog extends Animal{

}
```

Generalization: is a HAS A type of a relationship which is based on usage. It helps to reduce duplication of codes and bugs.

```
public class Vehicle{ }

public class Speed{ }

public class Van extends Vehicle{

  private Speed sp

}
```

This shows that class Van HAS-A Speed. By having a separate class for speed, its not necessarily to put the entire code that belongs to Speed inside the Van class. This makes it possible to reuse the Speed class in multiple applications.

In object oriented feature, the users do not need to bother about which object is doing the real work. To achieve this, Van class hides the implementation details from users of Van class. The users will ask the Van class to carryout an action and the Van class either does the work or asks another class to perform the action.

Java does only supports single inheritance ie. a class cannt extend more than one class . However a class can implement more than one interface.

Polymorphism

Polymorphism plays a main role in allocate objects having different internal structures to share the same external interface. This means that a general class of operations may be accessed in the same manner even though specific activities associated with each operation may differ. Polymorphism is broadly used in implementing inheritance.

It means objects that can take on or assume many different forms. Polymorphism means that the same operations may behave differently on different classes. Booch defines polymorphism as the relationship of objects many different classes by some common super class. Polymorphism allows us to write generic, reusable code more easily, because we can specify general instructions and delegate the implementation detail to the objects involved.

For Example:

In a pay roll system, manager, office staff and production worker objects all will respond to the compute payroll message, but the real operations performed are object particular

Overloading

This a technique required to perform similar functionality e.g. method could add 2 integer numbers together and another method required to add two floating point numbers. In order to give each method a unique name to belonger and more specific. Calling one method addint() and another addFloat() could lead to proliferation of names of each one describing different methods that essentially performing the same operation ie. adding two numbers. To overcome this, methods should not be required to have a unique name. To distinguish the two methods at runtime, Java Runtime Environment(JRE) looks at the parameter list.

If 2 integers are being passed, the first method is invoked. If 2 floating points are passed the the second method is invoked.

This is useful where several methods share the same name leading to fllexible programs.

Overloading methods don't just provide more flexibility to users but also provide flexibility for programmers who may need to extend the system in the future and thus overloading methods hence make a system more future proof and robust to changing requirements.

Note that it is possible to overload methods and constructors and this make programs more adaptable to changing requirements.

Conclusion

Polymorphism allows us to refer to objects according to a superclass rather than the actual class. It also makes it easy to extend our programs by adding additional classes without needing to change other classes. We can manipulate objects by invoking operations defined for the superclass without worrying about which subclass is involved in any specific area. Java ensures that the appropriate method for the actual class of object is invoked at runtime.

Assessment

In a University Management System , the method is required to enroll or register a new student.

enrollStudent(String Name, String Address, String programmecode).

What if a student doesn't have an adress, what can be done to allow the system register such a student. Hint: overload this method.

Unit Summary

This unit covered class and object declarartion, variables and how to declare variables, how to create constructors and finally the various object oriented concepts such as inheritance, polymorphism, overloading and abstraction.

Unit Assessment

UndergraduateStudent and GraduateStudent are subclasses of the Student class. In Java, we say a subclass extends its superclass. The difference between the classes GraduateStudent and UndergraduateStudent lies in the way their final course grades are computed. Lets define the two subclasses:

```
 class GraduateStudent extends Student {

public void computeCourseGrade() {

int total = 0;

for (int i = 0; i < NUM_OF_TESTS; i++) {

total += test[i];

}

if (total/NUM_OF_TESTS >= 80) {

courseGrade = "Pass";

} else {

courseGrade = "No Pass";

}

}

}

public int getTestScore(int testNumber) {

return test[testNumber-1];
```

```
public void setName(String newName) {

name = newName;

}

public void setTestScore(int testNumber, int testScore) {

test[testNumber-1] = testScore;

}

public void computeCourseGrade( ) {

//do nothing - override this method in the subclass

}

}

extends

class UndergraduateStudent extends Student {

public void computeCourseGrade() {

int total = 0;

for (int i = 0; i < NUM_OF_TESTS; i++) {

total += test[i];

}

if (total/NUM_OF_TESTS >= 70) {

courseGrade = “Pass”;

} else {

courseGrade = “No Pass”;

}

}

}
```

Instructions

Write a code to define another subclass called NonRegularStudent who scores a pass for courseGrade equal to or greater than 50

Grading Scheme

10 points for correct code.

Feedback

```
class NonRegularStudent extends Student {

public void computeCourseGrade() {

int total = 0;

for (int i = 0; i < NUM_OF_TESTS; i++) {

total += test[i];

}

if (total/NUM_OF_TESTS >= 50) {

courseGrade = “Pass”;

} else {

courseGrade = “No Pass”;

}

}

}
```

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Unit 3. Libraries and exception handling

Unit Introduction

This unit introduces learners to ways of ensuring that programs are robust and reliable. A program often encounters problems as it executes. It may have trouble reading data or there might be illegal characters in the data, or an array index might go out of bounds. The Java Exception class enables you to deal with such problems. Using it, you can write programs that recover from problems and keep on running. Most programs should not crash when the user makes an error!

Unit Objectives

Upon completion of this unit you should be able to:

- Synthesize reliability of code by incorporating exception-handling and assertion mechanisms.
- Write methods that propagate exceptions.
- Implement the try-catch blocks for catching and handling the thrown exceptions.
- Write programmer-defined exception classes.

Key Terms

Exceptions: Exceptions are generated when an error condition occur during the execution of a method.

Exception handling: a technique that enables programmers to create applications that can resolve exceptions

Learning Activities

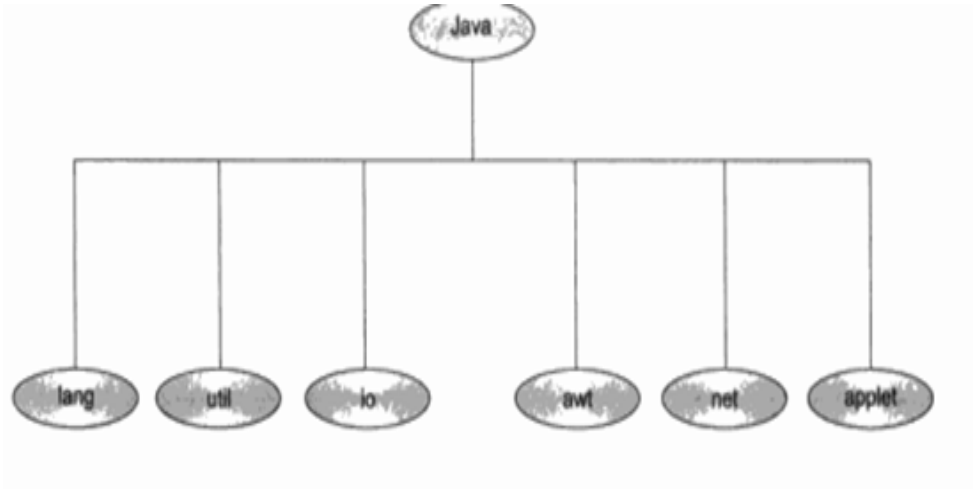
Activity 3.1 - java packages

Introduction

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 util- language utility classes such as vectors, hashtables, random numbers, data etc

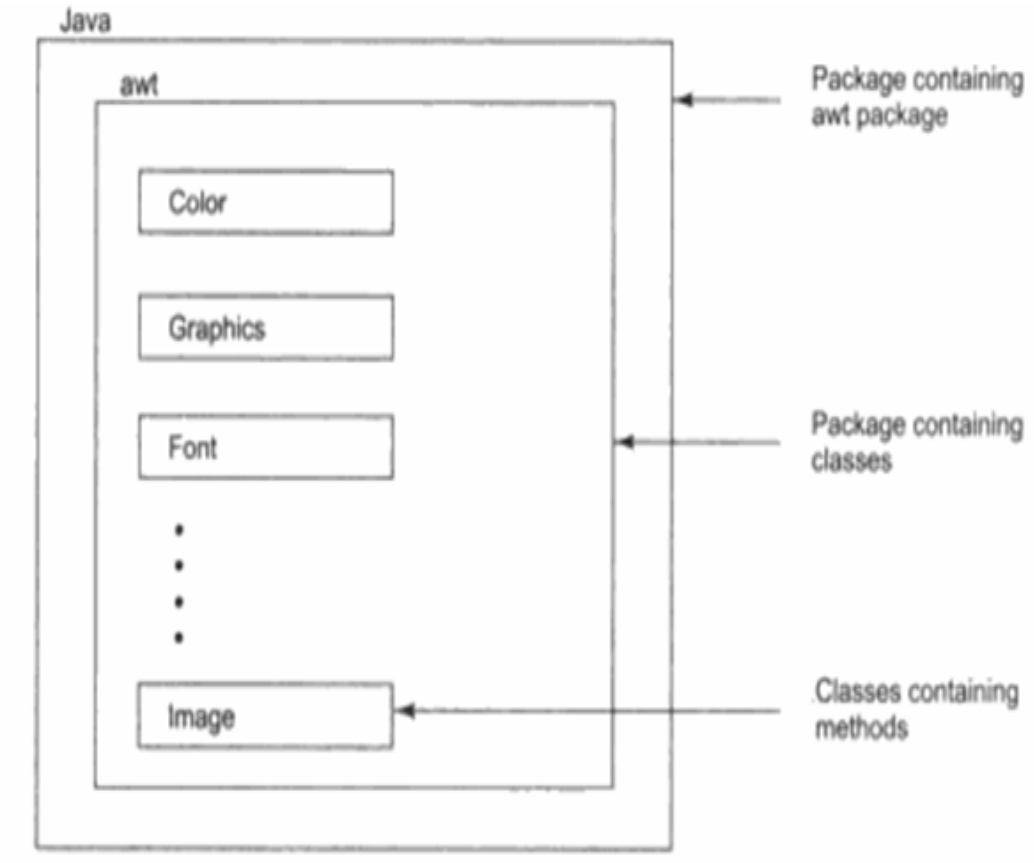
java awt-set of classes implementing graphical user interfaces eg windows, buttons, lists , menus etc

java net-classes for networking

java.applets- classes for implementing applets

java.io - classes for input , output functions are bundled in this package

Packages are organized in a hierarchical structure: e.g.



Package java contains package awt which in turn contains classes used to implement graphical user interfaces.

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.

How to create a package

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

The package statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.

If a package statement is not used then the class, interfaces, enumerations, and annotation types will be put into an unnamed package. Example:

Create a package called animals. Use lowercase to prevent conflicts with the names of classes.

```
/* File name : Animal.java */

package animals;

 interface Animal {

   public void eat();

   public void breath();

}
```

Note that this is a user defined package.

Activity Detail

Assume you have written some classes. Belatedly, you decide they should be split into three packages, as listed in the following table. Furthermore, assume the classes are currently in the default package (they have no package statements).

Package Name	Class Name
mygame.server	Server
mygame.shared	Utilities
mygame.client	Client

1. Which line of code will you need to add to each source file to put each class in the right package?

To adhere to the directory structure, you will need to create some subdirectories in the development directory and put source files in the correct subdirectories. What subdirectories must you create? Which subdirectory does each source file go in?

Feedback:

The first line of each file must specify the package:

In Client.java add:

package mygame.client;

In Server.java add:

package mygame.server;:

In Utilities.java add:

package mygame.shared;

2. To adhere to the directory structure, you will need to create some subdirectories in your development directory, and put source files in the correct subdirectories. What subdirectories must you create? Which subdirectory does each source file go in?

Within the mygame directory, you need to create three subdirectories: client, server, and shared.

```
In mygame/client/ place:

Client.java

In mygame/server/ place:

Server.java

In mygame/shared/ place:

Utilities.java
```

Conclusion

To create a package for a type, you need to put a package statement as the first statement in the source file that contains the type class, interface, ennumeration or annotation.

To use a public type that is in a different package, you need to have three options:

i. To use the fully qualified name of a type

ii. import the type or

iii. import the entire package of which the type is a member.

The path names for a package's source and class files mirror the name of the package.

You might also have to set the CLASSPATH so that the compiler and the JVM can find the. class files for your types.

Activity 3.2 -Exceptions and exception Handling

Introduction

There are Three Kinds of Exceptions

The first kind of exception is called checked exception. These are exceptional conditions that a well-written application should anticipate and recover from. For example, suppose an application prompts a user for an input file name, then opens the file by passing the name to the constructor for java.io.FileReader. Normally, the user provides the name of an existing, readable file, so the construction of the FileReader object succeeds, and the execution of the application proceeds normally. What if the user supplies the name of a nonexistent file, and the constructor throws java.io.FileNotFoundException. A well-written program will catch this exception and notify the user of the mistake, and prompt the user for a corrected file name.

Checked exceptions are subject to the Catch or Specify Requirement. All exceptions are checked exceptions, except for those indicated by Error, RuntimeException, and their subclasses.

Second kind of exception is the error. These are exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from. For example, suppose that an application successfully opens a file for input, but is unable to read the file because of a hardware or system malfunction. The unsuccessful read will throw java.io.IOError. An application might choose to catch this exception, in order to notify the user of the problem — but it also might make sense for the program to print a stack trace and exit. **Note** that Errors are not subject to the Catch or Specify Requirement. Further, Errors are those exceptions indicated by Error and its subclasses.

Third kind of exception is the runtime exception. These are exceptional conditions that are internal to the application, and that the application usually cannot anticipate or recover from. These includes programming bugs, such as logic errors or improper use of an API. For example, consider the application described previously that passes a file name to the constructor forFileReader. If a logic error causes a null to be passed to the constructor, the constructor willthrowNullPointerException. The application can catch this exception, but it probably makes more sense to eliminate the bug that caused the exception to occur. Runtime exceptions are also not subject to the Catch or Specify Requirement. Runtime exceptions are those indicated by RuntimeException and its subclasses. NOTE:Errors and runtime exceptions are collectively known as unchecked exceptions.

What is the meaning of Bypassing Catch or Specify?

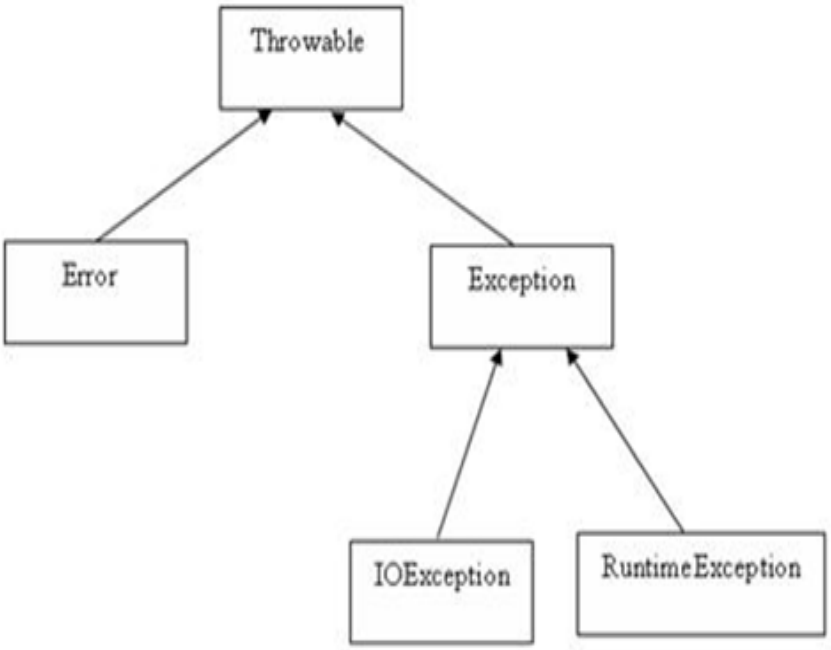
Some programmers consider the Catch or Specify Requirement a serious flaw in the exception mechanism and bypass it by using unchecked exceptions in place of checked exceptions. In general, this is not recommended.

When is it appropriate to use unchecked exceptions?. When a program is claimed to be reliable, we certainly expect the program will produce correct results for all valid input. It is hardly a reliable program if it produces correct results only for some input values. If we are not diligent and careful enough, we can easily introduce bugs in our programs.

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

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

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



Exceptions Methods:

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

SN	Methods with Description
1	**public String getMessage()**
	Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
2	**public Throwable getCause()**
	Returns the cause of the exception as represented by a Throwable object.
3	**public String toString()**
	Returns the name of the class concatenated with the result of getMessage()
4	**public void printStackTrace()**
	Prints the result of toString() along with the stack trace to System.err, the error output stream.
5	**public StackTraceElement [] getStackTrace()**
	Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	**public Throwable fillInStackTrace()**
	Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

Program correctness guarantees correct results for all valid input. But what happens when the input is invalid?

Another important criterion of program reliability is the robustness, which measures how well the program runs under various conditions. If a program crashes too easily when a wrong type

of argument is passed to a method or an invalid input value is entered, we cannot say the program is very robust.

A mechanism called exception handling can be used to improve the program's robustness. In this activity, we will describe how to code this exception-handling mechanism in Java to improve the program's robustness.

An exception represents an error condition that can occur during the normal course of program execution. When an exception occurs, the normal sequence of flow is terminated and the exception-handling routine is executed. When an exception occurs, we say an exception is thrown. When the matching exception-handling code is executed, we say the thrown exception is caught. By using exception-handling routines in our code, we can increase its robustness.

Catching an Exception: This is done by enclosing the statements that may cause an error in a try … catch block, as follows:

try

```
{
statements that may cause an error ….
}
catch (exceptionclass objectname)
{
statements that deal with the error
}
```

The exception class in the catch block must be either the one thrown by the statements in the try block, or a superclass of it.

The statements in the catch block can use the methods associated with the object caught in order to retrieve information from it.

Another example:

Catching Exceptions:

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

try

```
{
    //Protected code
}catch(ExceptionName e1)
```

```
{
    //Catch block
}
```

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

Example:

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

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

This would produce the following result:

Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3

Out of the block

Catching and Handling Exceptions

This section describes how to use the three exception handler components — the try, catch, and finally blocks — to write an exception handler. Then, the try-with-resources statement, introduced in Java SE 7, is explained. The try-with-resources statement is particularly suited to situations that use Closeable resources, such as streams.

The last part of this section walks through an example and analyzes what occurs during various scenarios.

The try BlockThe first step in constructing an exception handler is to enclose the code that might throw an exception within a try block. In general, a try block looks like the following:

```
try {
    code
}
catch and finally blocks . . .
```

The segment in the example labeled code contains one or more legal lines of code that could throw an exception. (The catch andfinally blocks are explained in the next two subsections.)

To construct an exception handler for the writeList method from the ListOfNumbers class, enclose the exception-throwing statements of the writeList method within a try block. There is more than one way to do this. You can put each line of code that might throw an exception within its own try block and provide separate exception handlers for each. Or, you can put all thewriteList code within a single try block and associate multiple handlers with it. The following listing uses one try block for the entire method because the code in question is very short.

```
private List<Integer> list;
private static final int SIZE = 10;
public void writeList() {
    PrintWriter out = null;
      try {
        System.out.println("Entered try statement");
        out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + list.get(i));
        }
      }
      catch and finally blocks  . . .
}
```

If an exception occurs within the try block, that exception is handled by an exception handler associated with it. To associate an exception handler with a try block, you must put a catch block after it.

The catch Blocks

You associate exception handlers with a try block by providing one or more catch blocks directly after the try block. No code can be between the end of the try block and the beginning of the first catch block.

```
try {
} catch (ExceptionType name) {
} catch (ExceptionType name) {
}
```

The following are two exception handlers for the writeList method:

```
try {
} catch (IndexOutOfBoundsException e) {
    System.err.println("IndexOutOfBoundsException: " + e.getMessage());
} catch (IOException e) {
    System.err.println("Caught IOException: " + e.getMessage());
}
```

The finally Block

The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling — it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

The following finally block for the writeList method cleans up and then closes the PrintWriter.

```
finally {
      if (out != null) {
        System.out.println("Closing PrintWriter");
        out.close();
      } else {
        System.out.println("PrintWriter not open");
      }
}
```

Conclusion

1. What does it mean to say a program is robust?
2. Java has predefined class throwable. What does this class represent?
3. Why does it exist?.
4. What is the purpose of try-catch block?

Assessment

1. Define the terms exception, error and throwable classes.
2. If the last catch block catches exception, then it will catch a type of exception not caught in the preceeding blocks. This is done in order to ensure that the user sees a pleasant error message rather than a confusing stack trace. In the following excerpt, ArithmetricExceptions are caught in the first block and all other Exceptions in the other.

```
try
    {
      System.out.print("Enter the numerator: ");
      num = scan.nextInt();
      System.out.print("Enter the divisor  : ");
      div = scan.nextInt();
      System.out.println( num + " / " + div + " is " + (num/div)
+ " rem " + (num%div) );
    }
    catch (ArithmeticException ex )
 {
System.out.println("You can't divide " + num + " by " + div);
    }
    catch (Exception ex )
    {
System.out.println("Something went wrong." );
    }
       System.out.println("Good-by" );
```

Identify the block where the RunTimeException will be caught.

Activity 3.3 -IO streams

Introduction

Stream:

The Java Input/Output (I/O) is a part of java.io package. The java.io package contains a relatively large number of classes that support input and output operations. The classes in the package are primarily abstract classes and stream-oriented that define methods and subclasses which allow bytes to be read from and written to files or other input and output sources.

The core Java language does not have any I/O methods. For a program to do I/O, it must import an I/O package. These notes (and most programs) use the package java.io.

Data for a program may come from several sources. Data created by a program may be sent to several destinations. The connection between a program and a data source or destination is called a stream.

An input stream handles data flowing into a program. An output stream handles data flowing out of a program.

There are many types of I/O devices. Some can be a source, others can be a destination, and others can be both source and destination. Often the destination for one stream is the source for another. For example a disk file might be the destination for the output of one program, and later it may be the source for another program.

A processing stream operates on the data supplied by another stream. Often a processing stream acts as a buffer for the data coming from another stream. A buffer is a block of main memory used as a work area. For example, disks usually deliver data in blocks of 512 bytes, no matter how few bytes a program has asked for. Usually the blocks of data are buffered and delivered from the buffer to the program in the amount the program asked for.

For reading the stream:

Open the stream

Read information

Close the stream

For writing in stream:

Open the stream

Write information

Close the stream

There are two types of stream as follows:

o Byte stream: It supports 8-bit input and output operations

Character stream:

The InputStream class is used for reading the data such as a byte and array of bytes from an input source. An input source can be a file, a string, or memory that may contain the data. It is an abstract class that defines the programming interface for all input streams that are inherited from it. An input stream is automatically opened when you create it and it can be explicitly closed with the close( ) method, or closed implicitly when the object is found as a garbage.

Examples of classes that inherit from inputStream are : ByteArrayInputStream, FileInputStream, ObjectInputStream, FilterInputStream.

The OutputStream class is used for writing byte and array of bytes to an output source. An output source can be as a file, a string, or memory containing the data. An output stream is automatically opened when you create it and can be explicitly closed with the close( ) method, or be closed implicitly when the object is garbage collected. Examples of classes that inherit from OutputStream are: ByteArrayOutputStream, FileOutputStream, ObjectOutputStream etc

Activity Details

The file datafile begins with a single long that tells you the offset of a single int piece of data within the same file. Write a program that gets the int piece of data. What is the int data?

Conclusion

The java.io packages contains many classes that your programs can use to read and write data. Most of these classes implement sequential access streams. Sequential access streams can be divided into two groups: those that read and write bytes and those that read and write unicode characters. Each of the sequential access stream has a speciality eg.i) reading from a file or

i. writing to a file or

ii. filtering data as its read or written or

iii. serializing an object.

Assessment

1. What class and method would you use to read a few pieces of data that are at known positions near the end of a large file?
2. When invoking format, what is the best way to indicate a new line?
3. How would you determine the MIME type of a file?
4. What method(s) would you use to determine whether a file is a symbolic link?

Unit Summary

Exception handling is another type of control flow.

An exception represents an error condition, and when it occurs, we say an exception is thrown.

A thrown exception must be handled by either catching it or propagating it to other methods. If the program does include code to handle the thrown exceptions, then the system will handle them.

A single method can be both a catcher and a propagator of an exception.

The standard classes described or used in this unit:

Throwable RuntimeException

Error IllegalArgumentException

Exception InputMismatchException

IOException

Unit Assessment

Instructions

Determine the output of the following code when the input is (a) 1, (b) 0, and (c) 12XY.

Scanner scanner = new Scanner(System.in);

```
try {
int num = scanner.nextInt();
if (num != 0) {
throw new Exception("Not zero");
}
System.out.println("I'm happy with the input.");
} catch (InputMismatchException e) {
System.out.println("Invalid Entry");
} catch (Exception e) {
System.out.println("Error: "+ e.getMessage());
}
```

Grading Scheme

10 marks for correct answer

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources

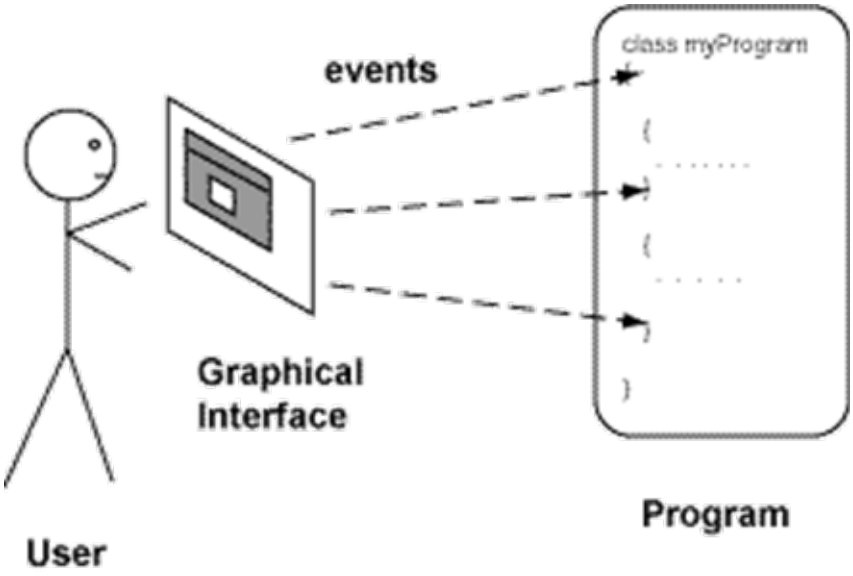
Unit 4. Introduction to GUI programming

Unit Introduction

This unit introduces the learner to GUI programming using applets , AWT and swing. Users interact with modern applications using graphical components such as windows, buttons, text boxes and menus. A GUI application program shows the user a graphical interface containing several graphical components. The user controls the program by interacting with the graphical components in the following ways:

- clicking on a button to choose a program option.
- making a choice from a menu.
- entering text in a text field.
- dragging a scroll bar.

Such actions are called events. The program responds to these events and the order of events is determined by the user and not the program. See the figure below:



A GUI has three parts:

- Graphical Components that make up the Graphical User Interface.
- Listener methods that receive the events and respond to them.
- Application methods that do useful work for the user.

The graphical components are swing objects usually extended to make them fit your application.

Listener methods are java methods that you write. They respond to events by calling application methods.

Application methods are ordinary java methods that perfor useful computations. They receive data from GUI and send data to GUI to display. Note : they are not usually concerned with the user interface e.g. Your Web broser has components such as 'back' and 'forward' buttons, listener methods that receive events such as click on 'back' button and application methods that do useful things like moving backward or forward one page.

What is an Event?

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page are the activities that causes an event to happen.

Types of Event

The events can be broadly classified into two categories:

- Foreground Events - Those events which require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page etc.
- Background Events - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

- Source - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.
- Listener - It is also known as event handler.Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received , the listener process the event an then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model ,Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

Steps involved in event handling

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class. the method is now get executed and returns.

Points to remember about listener

- In order to design a listener class we have to develop some listener interfaces. These Listener interfaces forecast some public abstract callback methods which must be implemented by the listener class.
- If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

Callback Methods

These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer. Here the callback methods represents an event method. In response to an event java jre will fire callback method. All such callback methods are provided in listener interfaces.

If a component wants some listener will listen to it's events the the source must register itself to the listener.

Unit Objectives

Upon completion of this unit you should be able to:

- Define an applet.
- Create compile and run applets.
- Create desktop applications using AWT.
- create basic applications using swing.

Key Terms

GUI: Programs implemented by use of standard classes from javax.swing and java.awt packages.

Applet: Programs that can run on a webbrowser

Learning Activities

Activity 4.1 - Creating applets

Introduction

In Java, GUI-based programs are implemented by using the classes from the standard javax. swing and java.awt packages. We will refer to them collectively as GUI classes

In a GUI environment, there are basically two types of windows: a generalpurpose frame and a special-purpose dialog. In Java, we use a JFrame object for a frame window and a JDialog object for a dialog. The first argument to the showMessageDialog method is a frame object that controls this dialog, and the second argument is the text to display. In the example statement, we pass null, a reserved word, meaning there is no frame object. If we pass null as the first argument, the dialog appears on the center of the screen. If we pass a frame object, then the dialog is positioned at the center of the frame. Run the ShowMessageDialog class and confirm this behavior.

Sample Program: Shows a Message Dialog

```
File: ShowMessageDialog.java

import javax.swing.*;

class ShowMessageDialog {

public static void main(String[] args) {

JFrame jFrame;

jFrame = new JFrame();

jFrame.setSize(400,300);

jFrame.setVisible(true);

JOptionPane.showMessageDialog(jFrame, "How are you?");

JOptionPane.showMessageDialog(null, "Good Bye");

}

}
```

An Applet is a small Internet-based program that has the Graphical User Interface (GUI), written in the Java programming language. Applets are designed to run inside a web browser or in

applet viewer to facilitate the user to animate the graphics, play sound, and design the GUI components such as text box, button, and radio button. When applet arrives on the client, it has limited access to resources, so that it can produce arbitary multimedia user interface and run complex computation without introducing the risk of viruses or breaching data integrity

Activity Details

To create an applet, we extend the –java.applet.Applet‖ class And by overriding the methods of java.awt.Applet, new functionality can be placed into web pages. Applets are compiled using javac compiler and it can be executed by using an appletviewer or by embedding the class file in the HTML (Hyper Text Markup Languege) file.

The java.applet package is the smallest package in Java API(Application Programming Interface). The Applet class is the only class in the package. The Applet class has many methods that are used to display images, play audio files etc but it has no main() method. Some of the methods include the following:

Init() : This method is used whenever initializations are needed for your applet. Applets can have a default constructor, but it is better to perform all initializations in the init method instead of the default constructor.

Start() : This method is automatically called after Java calls the init method. If this method is overwritten, code that needs to be executed every time that the user visits the browser page that contains this applet.

Stop() : This method is automatically called when the user moves off the page where the applet sits. If your applet doesn't perform animation, play audio files, or perform calculations in a thread, you don't usually need to use this method.

Destroy(): Java calls this method when the browser shuts down.

Applet Lifecycle

Every java Applet inherits a set of default behaviours from the Applet class. As a result, when an applet is loaded it undergoes a series of changes in its state. Following are the states in applets lifecycle.

Born or Initialisation state :

An applet begins its life when the web browser loads its classes and calls its init() method. This method is called exactly once in Applets lifecycle and is used to read applet parameters. Thus, in the init() method one should provide initialization code such as the initialization of variables.

Eg. public void init()

```
{
//initialisation
}
```

Running State:

Once the initialization is complete, the web browser will call the start() method in the applet. This method must called atleat once in the Applets lifecycle as the start() method can also

be called if the Applet is in —Stoped|| state. At this point the user can begin interacting with the applet.

Eg. public void start()

```
{
//Code
}
```

Stopped State:

The web browser will call the Applets stop() method, if the user moved to another web page while the applet was executing. So that the applet can take a breather while the user goes off and explores the web some more. The stop() method is called atleast once in Applets Lifecycle.

Eg. publc void stop()

```
{
//Code
}
```

Dead State:

Finally, if the user decides to quit the web browser, the web browser will free up system resources by killing the applet before it closes. To do so, it will call the applets destroy() method. One can override destroy() to perform one-time tasks upon program completion. for example, cleaning up threads which were started in the init() method.

Eg. public void destroy()

```
{
// Code
}
```

Note: If the user returns to the applet, the web browser will simply call the applet's start() method again and the user will be back into the program.

Display State :

Applet moves to the display state whenever it has to perform the output operations on the screen. This happens immediately after the applet enters into the running state. The paint() method is called to accomplish this task.

Eg. public void paint(Graphics g)

```
{
//Display Statements
}
```

How to create an applet

```
import java.awt.*;
import java.applet.*;
public class SimpleApplet extends Applet
{
public void paint(Graphics g)
{
g.drawString("My First Applet",40,40);
}
}
```

Save the file as SimpleApplet.java

Compile the file using javac SimpleApplet.java

In the first line we imports the Abstract Window Toolkit(AWT) classes

In the second line we import the Applet package, which contains the class —Applet.

The next line declares the class SimpleApplet. Inside simpleApplet, paint() method is declared. This method is defined by the AWT and must be overridden by the Applet. Method paint() is called each time that the applet must redisplay its output.

This paint() method has parameter of type — Graphics. This parameter contains the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

Inside paint() method is a call to drawstring(), which is a member of the Graphics class. This method output a String beginning at specified X, Y locations.

How to run your applet:

Use an applet viewer, such as the standard SDK tool, appletviewer. An applet viewer executes your applet in a window.

You are required to develop a sample frame window that illustrates the fundamentals of GUI programming. The sample frame window should have two buttons labeled CANCEL and OK. When you click the CANCEL button, the window's title is changed to You clicked CANCEL. Likewise, when you click the OK button, the window's title is changed to You clicked OK.

There are two key aspects involved in GUI programming. One is the placement

of GUI objects on the content pane of a frame, and the other is the handling of events generated by these GUI objects.

You can develop the sample program in two steps.

1. First define a JFrame subclass called MyFirstJButtonFrame to show how the two buttons labeled OK and CANCEL are placed on the frame.
2. Implement another subclass called MyFirstJButtonEvents to show how the button events are processed to change the frame's title.

Note that you are going to use buttons in your program. To use a button in your program, create an instance of the javax.swing.JButton class. Remember you are supposed to create two buttons and place them on the frame's content pane in the constructor. Name them cancelButton and okButton

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

…

```
JButton cancelButton, okButton;
cancelButton = new JButton("CANCEL");
okButton =     new JButton("OK");
```

Conclusion

Applets as previously described, are the small programs while applications are larger programs. Applets don't have the main method while in an application execution starts with the main method. Applets are designed just for handling the client site problems. while the java applications are designed to work with the client as well as server. Applications are designed to exists in a secure area. while the applets are typically used. Applications are not too small to embed into a html page so that the user can view the application in your browser. On the other hand applet have the accessibility criteria of the resources.

Java is often described as a Web programming language because of its use in writing programs called applets that run within a Web browser. That is, you need a Web browser to execute Java applets. Applets allow more dynamic and flexible dissemination of information on the Internet, and this feature alone makes Java an attractive language to learn.

Assessment

1. Display the message I Love Java by using JOptionPane.
2. Using JOptionPane input dialog, write a statement to input the person's firstname.
3. Using JOptionPane input dialog, write a statement to input the person's age(integer).

Activity 4.2 -GUI PROGRAMMING

Introduction

Graphical User Interface(GUI) is a type of user interface item that allows people to interact with programs in more ways than typing such as computers and many hand-held devices such as mobile phones. A GUI offers graphical icons, and visual indicators, as opposed to text-based interfaces. This helps to develop more efficient programs that are easy to work with. The user can interact with the application without any problem.

The GUI application is created in three steps. These are:

1. Add components to Container objects to make your GUI.
2. Setup event handlers for the user interaction with GUI.
3. Explicitly display the GUI for application.

GUI Components

It is visual object and the user interacts with this object via a mouse or a keyboard. Components included, can be actually seen on the screen, such as, buttons, labels etc. Any operation that is common to all GUI components are found in class Component. Different components are available in the Java AWT (Abstract Window Toolkit)package for developing user interface for your program.

A class library is provided by the Java programming language which is known as Abstract Window Toolkit (AWT). The Abstract Window Toolkit (AWT) contains several graphical widgets which can be added and positioned to the display area with a layout manager.

AWT is a powerful concept in JAVA. AWT is basically used to develop GUI applications. AWT is platform dependant. That means your .class file after the program compilation is platform independent but the look of your GUI application is platform dependant. AWT copies GUI component from local machines operating system. That means your applications look will differ in MAC operating system, as you have seen in WINDOWS operating system.

Examples of AWT components:

1. Labels:

This is the simplest component of Java Abstract Window Toolkit. This component is generally used to show the text or string in your application and label never perform any type of action. Syntax for defining labels:

```
Label label_name = new Label ("This is the label text.");
```

above code simply represents the text for the label.

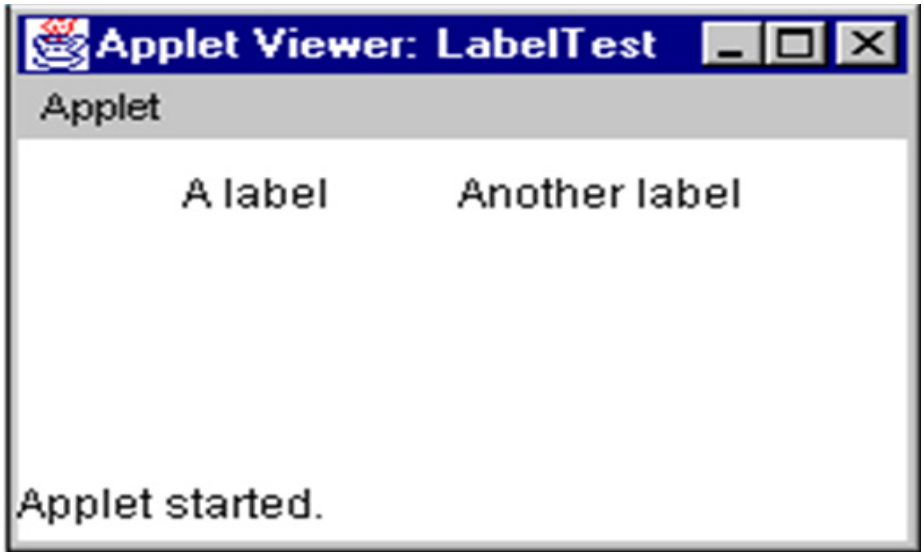
Label label_name = new Label ("This is the label text. || , Label.CENTER);

Justification of label can be left, right or centered. Above declaration used the center justification of the label using the Label.CENTER.

```
            import java.awt.*;
import java.applet.Applet;
public class LabelTest extends Applet
{
public void init()
{
add(new Label("A label"));
// right justify next label
add(new Label("Another label", Label.RIGHT));
}
}
```

Save the file as LabelTest. Java Compile the file using javac LabelTest.java On successful compilation, execute the file using appletviewer LabelTest.java

results:



Buttons:

This is the component of Java Abstract Window Toolkit which is used to trigger actions and other events required for your application. The syntax of defining the button is as follows:

Button button_name = new Button ("This is the label of the button.");

Example:

```
            import java.awt.*;
  import java.applet.Applet;
```

```
public class ButtonTest extends Applet
{
public void init()
{
Button button = new Button ("OK");
add (button);
}
}
```

results:

Activity Details

Using the two examples above, write a snippet code to create a checkbox and a radio button in your application.

Your results should appear as follows:

Conclusion

In this unit , you learnt how to create simple desktop applications using AWT package. However, AWT is not flexible enough for writing complex sophisticated applications, To create simple flexible and platform independent applications, we use swing covered in the next activity.

Assessment

Write an applet that displays a set of three concentric circles of different colours centered in the applet's drawing area.

Activity 4.3 -Swing

Introduction

Swing components facilitate efficient graphical user interface (GUI) development. These components are a collection of lightweight visual components. Swing components contain a pluggable look and feel. This allows all applications to run with the native look and feel on different platforms. PL&F allows applications to have the same behavior on various platforms. JFC contains operating systems neutral look and feel. Swing components do not contain peers. Swing components allow mixing AWT heavyweight and swing lightweight components in an application. The major difference between lightweight and heavyweight components is that lightweight components can have transparent pixels while heavyweight components are always opaque. Lightweight components can be non-regular while heavyweight components are always rectangular.

Examples of swing components include JFrame, JButton, JCheckbox etc.

A sample program to display two buttons, one text field and two labels:

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

class TextFrame2 extends JFrame implements ActionListener {

...

private JLabel prompt;

private JLabel image;

public static void main(String[] args) {

TextFrame2 frame = new TextFrame2();

frame.setVisible(true);

}

public TextFrame2() {

...

image = new JLabel(new ImageIcon("cat.gif"));

image.setSize(50, 50);
```

```
contentPane.add(image);

prompt = new JLabel();

prompt.setText("Please enter your name");

prompt.setSize(150, 25);

contentPane.add(prompt);

...

}

...

}
```

Activity Details

Save the above the file as TextFrame2.java

Compile using javax.TestFrame2.java

Execute the file using appletviewer TestFrame2.java

Conclusion

Swing was created to address the problems of AWT . We use swing to create basic applications which are more portable and allow the programmer to specify look and feel. These applications can change depending on the platform or else remain the same across all platforms.

Assessment

1. What part of a frame holds the graphical components?

A. content pane B. content provider

C. data frame D. window pane

2. What type of object determines where GUI components are placed in a container?

A. The layer organizer. B. The component manager.

C. The frame manager. D. The layout manager.

3. What method of a frame prevents (or allows) a user to change the size of the frame?

A. setResizable() B. setStretch()

C. clearSizable() D. clearUser()

4. How does FlowLayout() put components into the content frame?

A. By rows starting at the top, then left to right in each row.

B. Starts at the bottom, then right to left in each row.

C. Starts at the center, then spirals outward.

D. Puts the first component in the center, then squeezes the rest in around it.

5. What type of component is a JPanel()?

A. Container B. JFrame

C. JButton D. JWindow

6. What is the default layout manager for JPanel?

A. PanelLayout B. FrameLayout

C. FlowLayout D. BoxLayout

7. Which alignment directions may be used with BoxLayout?

A. Horizontal only. B. Horizontal and Vertical.

C. Left aligned and Right aligned. D. Centered and Justified.

8. Can a JPanel be placed inside another JPanel?

A. Yes.

B. No, only visible components may be placed in JPanels.

C. No, JPanel can't be nested.

D. Yes, but only one JPanel may be placed inside another.

9. Can a different layout manager type be picked for each JPanel ?

A. No, only one type of layout manager is used for the entire frame.

B. Yes, each container has its own layout manager, independent of any others.

C. No, all JFrames in a frame use the same layout manager.

D. Yes, but only if JPanels are nested.

10. How are components added to a JPanel?

A. With the set() method.

B. With the add() method.

C. With the put() method.

D. With the setComponent() method

Unit Summary

This unit covers applets, how to create applets and GUI programming applications. It also enables the learner to distinguish between AWT and swing GUI applications.

Unit Assessment

GUI applications

1. Explain why its important to import javax.swing and java.awt.event in the sample programs provided above.
2. Study the snippet code below and answer the questions that follow.
3. class ButtonListener implements ActionListener

{public void actionPerformed(ActionEvent evt)

4. Write a program implements a game:

There are three buttons in the frame. Two of the buttons cause the program to quit using System.exit(0); the remaining button changes the frame to green (a win!) The winning button is different each time the game is played.

The easy way to do this (although it seems unfair to the user) treats each button the same way. The actionPerformed() method does not check which button was clicked. When any button is clicked, the method picks a random integer from 0 to 2 and performs the "winning" action if the integer happens to be 0. Otherwise, it performs the "losing" action. NOTE: To the user, it seems like there is a "winning" button and two "losing" buttons. But, in fact, it does not matter which button was clicked.

Grading Scheme

15 marks for correct answers

Feedback

Refer to the notes.

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Course Assessment 1

Instructions

Answer the following questions:

QUESTION ONE(30 MARKS)

(a) Using examples, define the following terms (4 marks)

(i) Object

(ii) Class

(b) Explain three types of numeric data types used in Java. (6 marks)

(c) Java is portable. Explain this statement. Illustrate with an example. (3 marks)

(d) Describe three steps involved in creation of an object from a class. (3 marks)

(e) Write an object oriented Java program that reads in the values of a, b and c and computes and outputs the values of x1 and x2 in the equation.

Note: you must create and use an object in your program. (10 marks)

(f) State the function of the following operators. (4 marks)

i. =

ii. +=

iii. !

iv. ==

QUESTION TWO

(a) Give six reasons why Java is popular today. (6 marks)

(b) Consider a Lecturer class with two subclasses, fulltime and parttime lecturer. Write at least 3 constructors for each of the subclasses and 2 for the super class. (5 marks)

(c) Illustrate how you would create an object of the type parttime lecturer. (4 marks)

Grading Scheme

CAT 1 will contribute 10 % of final mark ie 100

Course Assesment 2

Instructions

Answer all the questions below

(a) Explain how modularity is implemented in Java. (2marks)

(b) Give five ways in which an interface is different from a class. (5 marks)

(c) Given the statement:

public static void main(String [] args)

Explain what is meant by the following:

(i) public

(ii) static

(iii) void

(iv) main

(v) String[]args (5 marks)

(d) Mention the application of the following inbuilt Java methods. (3 marks)

i. System.out.println()

ii. Double.parseDouble()

iii. JOptionPane.showMessageDialog()

e) Write a for loop that will print out all the multiples of 2 from 2 to 100 (i.e. 2 4 6…100) (5 Marks)

Grading Scheme

CAT 2 will contribute 10 % of final mark ie 100

Module Summary

In this module , you learnt how to write simple desk top apllications using Java, to design and implement flexible and reusable solutions using Object Oriented principles and to implement applications with event driven GUI. This knowledge can be applied in solving real world problems through software development in Java.

Course References

Introduction to object oriented programming with java , 5th edition Wu, C. Thomas, 2010

Java 2: The Complete Reference, Fifth Edition, Herbert Schildt, Tata McGraw Hill.

Introduction to Computer Science using Java by Bradley Kjell licensed under a Creative Commons Attribution-NonCommercial 3.0 Unported License.

The African Virtual University Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

The African Virtual University Regional Office in Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org