KAPLAN HIGHER EDUCATION ACADEMY

KAPLAN HIGHER EDUCATION INSTITUTE

Study Guide

Diploma Programme

Introduction to Programming using Java

v2.0

Message to Student

"I encourage you to be an active learner, questioning and extending what you know in order to become a contributing participant in the modern global economy."

Dear Kaplan Student,

Thank you for choosing Kaplan Singapore and entrusting your educational investment with us.

It is Kaplan Singapore's and the School of Diploma Studies' mission as your 'private education institution of choice' to commit to providing a fulfilling student journey and learning experience for your development. The underlying goal of Kaplan Singapore is to ensure that our students are able to respond the demands of the globalised economy set against a highly uncertain and complex environment.

In the 21st Century it is important that your educational experience provides with you with the skills and competencies on HOW to think and not just WHAT to think.

As such, the Kaplan Diplomas, which have a long history of being recognised as equivalent to the first year or more of undergraduate studies in high-quality international universities, aim to prepare you for work, further study, and lifelong learning.

Your role as a Kaplan student is to participate, think deeply on a topic and work with other students and your Lecturers to broaden your knowledge. I encourage you to be an active learner, questioning and extending what you know in order to become a contributing participant in the modern global economy.

However, only you can embody these attributes. You are the major determinant of your own academic success and as adult learners you will receive both the freedom and responsibility this entails. With that, I wish you well in your learning and look forward to seeing you succeed.

Krishna Rajulu
Academic Dean, Kaplan Singapore

Kaplan Desired Graduate Attributes

Through the reading of this module, Kaplan Singapore intends to:

- Instill in students the value of lifelong and self-directed learning by stimulating intellectual curiosity, creative and critical thinking and an awareness of cultural diversity;
- Assist students in developing professional attributes, ethical values, social skills and strategies that will nurture success in both their professional and personal lives;
- Foster integrity, commitment, responsibility and a sense of service to the community;
- Prepare students to meet the ever-changing needs of their communities both now and in the future; and
- Promote innovative and effective teaching.

Culminating from these institutional values and educational goals, Kaplan Singapore's Desired Graduate Attributes are:

Inquiry and criticality: Graduates will be able to critically collect, evaluate and apply information and data in order to make decisions in a wide variety of professional situations. This attribute is demonstrated when students:

- Undertake, evaluate and apply appropriate research, theories, concepts and tools to investigate problems and find solutions;
- Exercise critical thinking and independent judgement to assess situations and determine solutions; and
- Have an informed respect for the principles, methods, values and boundaries of their profession and the capacity to question these.

Ethicality and discernment: Graduates will be able to assess situations and respond in an ethically, socially and professionally responsible manner. This attributed is demonstrated when students:

- Act responsibly, ethically and with integrity in their profession;
- Hold personal values and beliefs and participate in the broad discussion of these values and beliefs while respecting the views of others;
- Understand the broad local and global economic, political, social and environmental systems and their impact as appropriate to their discipline and profession; and
- Acknowledge personal responsibility for their own judgments and behaviour

Ability to communicate well: Graduates will recognise the importance and value of communication in the learning and professional environment. This attributed is demonstrated when students:

- Create and present knowledge, arguments and ideas confidently and effectively using a variety of methods and technologies;
- Recognise the wide range of possible audiences for information and respond with communication strategies appropriate to those audiences; and
- Work collaboratively with people from diverse backgrounds and be aware of the different roles of team members and to function within that team.

Independent and reflective practitioner

- Graduates will be able to work independently and be self-directed learners with the capacity and motivation for continued professional learning and development; and
- They will be able to critically reflect on their own practice and evaluate and understand current capacity and further development needs

Embedded within the desired graduate attributes are the following skills:

- Conduct research.
- Analyse, organise and present data and information.
- Think and read critically.
- Make an oral presentation.
- Intellectual curiosity and awareness of culture and diversity.
- Develop professional ethos and practice that will foster success in career and life.
- Meet the ever changing needs of communities now and in the future.

Table of Contents

Message to Student	i
Kaplan Desired Graduate Attributes	ii
Table of Contents	iii
About this module	iv
Scheme of Work	v
Assessment Matters	vii
Topic 1 Introduction to Java Programming	1
Topic 2 Basic Computation	7
Topic 3 Control Structures Part I	17
Topic 4 Control Structures Part II	23
Topic 5 Classes and Methods Part I	31
Topic 6 Classes and Methods Part II	38
Topic 7 ARRAYS	46
Topic 8 Inheritance and Polymorphism	55
Topic 9 Files	65
Topic 10 Recursion	71

About this module

This unit is designed to develop the skills of problem solving and program design using the Java programming language. Students will also analyse the features of Object Oriented Programming Language and the definition and use of common classes.

It means to develop software (program) which contains instructions to tell a computer what activities to perform. Usually to solve problems.

There are multiple programming languages, each invented for a specific purpose. Hence there is no “best” language as each has its own strengths and weaknesses. Therefore, the key to know which language is appropriate to solve certain problems, would be best to master as many different languages as possible.

To begin your programming journey, start by learning one language! Once you have master one, you will find it easy to pick up other languages.

This module will introduce students to the use of Java language to perform basic computation, the concept of control structures, classes and methods, arrays, inheritance and polymorphism, and recursion.

On completion of this module, students will develop the skills of problem solving and program design using the Java programming language.

Module Learning Outcomes

Upon successful completion of this module, the student should be able to:

- Describe the java programming language and the structure of a simple Java application;
- Describe the relationship between Java and the World Wide Web;
- Create simple application using control structures;
- Create applications utilising java classes and methods in the design in order to improve code efficiency and expand functionality;
- Create Java applications utilising polymorphism, inheritance properties of OOP;
- Describe how recursion and I/O is processed in Java.

Overview of Learning Resources

Recommended reading:

JAVA: An introduction to Problem Solving & Programming. Walter Savitch & Frank, Carrano. Fifth Edition (2010). Pearson

Simply Java Programming: An Application-Drive Tutorial Approach. Deitel & Deitel. 2007 Prentice Hall.

Other sources:

See Proquest and Newslink databases linked to your Elearn LMS homepage. The National Library Board on North Bridge Road (databases are for Singaporean/PR only)

Scheme of Work

LESSON	TOPICS
1	01 Introduction to Java Programming • Programming Languages • Operating Systems • Java, the World Wide Web, and Beyond • The Java Language Specification, API, JDK, JRE, and IDE • Simple Java Program • Creating, Compiling, and Executing a Java Program • Programming Style and Documentation • Programming Errors
2	02 Basic Computation • Writing a Simple Program • Identifiers • Variables • Assignment Statements and Assignment Expressions • Named Constants • Naming Conventions • Data Types and Operations • Various Operators • Numeric Type Conversions • The String Type and methods
3	03 Control Structures Part I • Sequence • Selection • Boolean Data Type • if Statements • Nested if and Multi-Way if-else Statements • switch Statements • Conditional Operators
4	04 Control Structures Part II • Repetition • Loop Design Strategies • Counter controlled loop • Controlling a Loop with User Confirmation or a Sentinel Value • The while Loop • The do-while Loop • The for Loop • Nested Loop • Which Loop to Use? • Keywords break and continue
5	05 Classes and Methods Part I • Defining Classes for Objects • Static classes, Built in classes • Using Classes from the Java Library • Static Variables, Constants, and Methods • Passing Objects to Methods • The this Reference Revision Topic 1 - 5

6	06 Classes and Methods Part II • Introduction • Defining a Method • Calling a Method • Void vs. Value-Returning Methods • Passing Parameters by Values • Constructors • Method Abstraction and Stepwise Refinement • Overloaded methods and constructors
7	Assignment Brief, Discussion and Consultation
8	07 Arrays • Introduction • Array Basics • Variable-Length Argument Lists • Searching Arrays • Sorting Arrays • Introduction • Two-Dimensional Array Basics • Processing Two-Dimensional Arrays • Passing Two-Dimensional Arrays to Methods
9	08 Inheritance and Polymorphism • Inheritance • Super classes and Subclasses • Using the super Keyword • Overriding Methods • Polymorphism and Dynamic Binding • The Object's equals Method
10	09 Files • Introduction • How Is Text I/O Handled in Java? • Text I/O vs. Binary I/O
11	Assignment Draft Submission and Consultation
12	09 Files • Introduction • Case Study: Computing Factorials • Case Study: Computing Fibonacci Numbers
13	Revision Topics 6 - 10 Assignment Consultation (PT) Assignment Presentation (FT)
14	

Assessment Matters

Assessment Overview

Assessment 1: CA Quiz

Weightage: 20% (20 marks)
Date: To be confirmeed
Duration: 10 minutes per quiz
Test Format: 5 MCQs per topic

Assessment 2: Individual Assignment

Weightage: 80% (100 marks)
Date: Lesson Session 12 (FT) / Session 7 (PT)
Word Limit: Module Specific
Citation Format: APA
References: Module Specific

Important Policies

Penalties for Plagiarism

Plagiarism in any form is not tolerated by Kaplan Singapore. That said, direct quotations and general similarities of common terms and language mean the E-Learn LMS will often pick up every small similarity so the likelihood of a Turnitin Similarity report recording a result of 0% is unrealistic. After all, no technology is perfect and there is the need for some direct quotation (provided you reference using APA guidelines, of course) and to use commonly accepted terms and language.

TOP TIP:
The surest way to succeed is to ensure all work is correctly referenced. Keep a copy of the Kaplan Singapore Academic Works and APA Guide handy when you are typing your assignments and use it to guide you as to correct referencing, citation and other aspects of academic writing.

Penalties for late submissions

Kaplan Singapore prepares students for the realities of the workforce and further education by requiring students to meet deadlines and submit all work on time. As such, students are required to seek approval and penalties will be imposed on late assignment submissions in accordance with the table below and cited in the Programme Handbook:

No of days late	Penalty
1 – 5 days	10% deduction per day from the marks attained by students.
After 5 days	Assignments that are submitted more than 5 days after the due date will not be accepted and it will be deemed as "No Submission". Student will be required to re-module.

Assignments and Kaplan Learning Management System

Kaplan Singapore School of Diploma Studies requires you to submit Assignments through the Learning Management System (E-Learn LMS). When submitted, your assignment is checked for plagiarism by software called Turnitin linked to the E-Learn LMS. The software is intended to provide one more tool to improve the quality of academic writing and as such will be compulsory for use. It is important to note that this is merely one of many tools available to you and that final decisions about the quality of your work rest with your lecturer.

Assigment Submission: How to Use E-Learn LMS for Assignment Submission

1. You will be enrolled by the School of Diploma Studies Programme Management into the E-Learn LMS system only after your fee payment is confirmed.
2. You will be sent your USER NAME and PASSWORD via email.
3. Reset your password as prompted.
4. Enter the site at the following address: https://elearn-diploma.kaplan.com.sg
5. To submit assignment please refer to the LMS Manual

Please refer to your Student Handbook for more details on Penalties for Plagiarism, Misconduct, Examinations Rules and Regulations. Should you have any queries, please contact diploma.sg@kaplan.com

This page intentionally left blank

Study Guide

Topic 01 – Introduction to Java Programming

Lesson Learning Outcomes

- What is a programme and explain different types of Programming languages?
- Describe java specification like JRE, JVM and JDK?
- How to write and execute our first java programme?
- What are errors and explain different types of errors?
- Different types of comments in java programming?

What is a programme?

A programme is a step-by-step instruction given to a computer to perform a specific task. It is very similar to the English language like it is a way to communicate between person to person. The program is a way to communicate with humans to computers.

There are different types of programming languages available, and broadly they will be divided into two categories
1. High-Level Programming Language
2. Low Level or Machine Level Programming Language

High-Level Programming Language:

Any program that we write in a human-understandable format is called high-level programming language. To run it, we need a compiler or interpreter, which transforms it into a machine-understandable format.
Example: Java, C++, and Python

Low-Level Programming Language:

Machine language, or machine code, is the lowest level of computer languages. It contains binary code, often generated by compiling high-level source code to run it. We do not need a compiler or interpreter. We need an assembler.

When the CPU executes a program, it'll store program within the computer's main memory which is also referred as RAM. It consists of a sequence of addresses. Once the CPU has to access the program instruction or knowledge in a very specific address, it sends the address of that info as a signal to the memory. The main memory responds back by the info contained within that particular address.

History about JAVA

James Gosling developed Java at Sun Microsystems in 1995.It was created with the idea to write code one time and run it on any operating system, hence the term "**Write Once, Run Anywhere**."

Java can be described as a "simple, object-oriented, multi-threaded, garbage-collected, secure, robust, architecture-neutral, portable, high performance, dynamic language." Soon after its introduction, WWW browsers updated to include applets, known as small Java programs, in HTML documents. First through applets, and later through complete Internet applications, Java has become widespread on the Internet. Its built-in support for URLs and the HTTP protocol allows Java applications to access standard WWW servers. These Java features have improved the feasibility of the development of Internet hypermedia that does not rely on a WWW browser for a hypermedia client.

JAVA SPECIFICATIONS

Java Development Kit

JDK is a combination of all the tools and executables that are used to debug, compile and execute a Java Program. JDK is a platform dependent, and that is reason why we have separate installers for Mac, Windows and Unix systems.

Java Virtual Machine

JVM is the core of java programming language. At the point when we run a program, JVM is liable for changing over Byte code to the machine-specific code. JVM is adjustable, and we can utilize java choices to modify it, for instance, allotting maximum and minimum memory to JVM. JVM is virtual in light of the fact that it gives an interface that doesn't rely on the underlying operating system and machine hardware. This independence from hardware and operating system is what makes Java program write-once run-anywhere

Java Runtime Environment

JRE is the Implementation of JVM. It gives a stage to execute java programs. JRE comprises of JVM and java binaries and different classes to execute programs effectively. JRE doesn't contain any advancement devices like java compiler, debugger, and so forth. To execute any java program, we require JRE, however we needn't bother with JDK for running any java program.

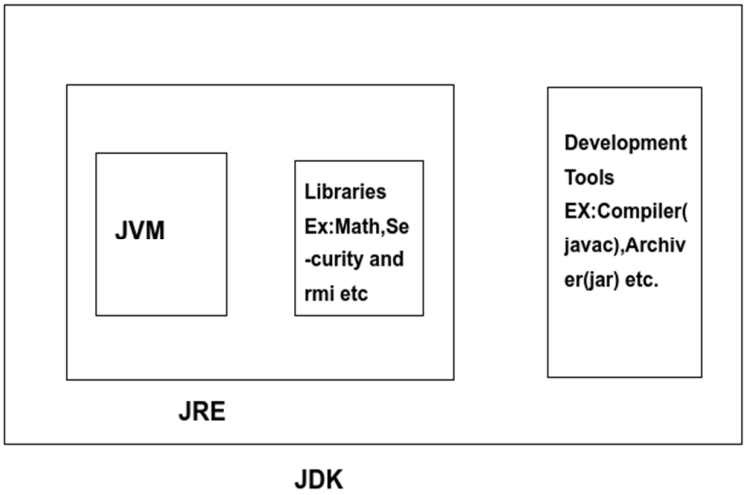
Application Program Interface

API contains predefined interfaces and classes for developing Java application programs. The Java language specification is still stable, but the API is still expanding time by time. At the Sun Java Website (java.sun.com), you can download and view the Java API's latest version.

Integrated Development Environment

IDE is a tool that can be used for creating, editing, debugging and compiling Java code. These tools additionally give dynamic and static assistance to helping any developer to create code.

Example: NetBeans, Notepad, Eclipse, etc.



JDK = JRE + Development Tools
JRE = JVM + Libraries

You can download the latest JDK together with NetBeans IDE from the link shown below:

https://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-3413139-esa.html

we will use NetBeans as IDE to write our programs

How to write, compile, and execute our first java program.

Step1. Go to NetBeans, and write the following code.

```
1. public class FristProgram
2.  {
3.          public static void main(String[] args)
4.          {
5.              System.out.println("Hello World");
6.          }
7. }
```

Save it as FirstProgram.java, remember the class name and the file name should be the same.

Step2. Run the file.

It will show the following output:

```
FirstProgram > main >
Output - JavaApplication4 (run)
run:
Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

Line 1: public class FristProgram

Here public is an access specifier which means this class can be visible to all, class is a keyword and FirstProgram is the name of the class

Line 3: public static void main(String[] args)

main() is a method which specifies the start of a program, void is return type of main method which means it will not return any value. If we declare any method as static it means that it does not require any object to execute that method. String[] args is an array of strings which is used for command line arguments.

Line 5: System.out.println("Hello World");

It can be considered as a print statement, anything written inside double quotes will be displayed on the console. System is a predefined class, out is the object of PrintStream class, println() is the method of PrintStream class.
We will learn more about access specifiers, classes and objects in next chapters.

ERRORS

There are three kinds of errors in the programming language. They are:

1. Syntax errors
2. Runtime errors
3. Logic errors.

Syntax errors or Compile time

These are errors where the compiler finds something wrong with the program, and these errors are identified at compile time, for example, using a variable that has not been declared. We will learn more about variables in the coming chapters.

Logic errors

Logic errors occur when our program compiles and runs, but it does not give the output that we are expecting. JVM does not help this kind of error because it does not know what our program is supposed to do.

Runtime errors

If there are no syntax errors, and if the errors occur at run time that stops the program execution, it is run time error. Division by zero is an example of run time error.

COMMENTS:

Comments make the program more readable. It is useful to provide more information about the programs.
There are two types of comments

1. Single-line comments which start with two forward slashes (//)
2. Multi-line comments which start with /* and end with */.

Java will ignore any text between /* and */.

REFERENCES

Herbert Schildt, Java: The Complete Reference,11th Edition, McGraw-Hill Education, ISBN: 9781260440249

Topic 02 – Basic Computation

Lesson Learning Outcomes

- What are identifiers and rules for identifiers?
- What are different types of variables available in java?
- What are named constants and why to using naming conventions?
- What are different data types available in java?
- What are different operators available in java?
- What are different string methods available in java?

IDENTIFIERS

Identifiers are the names given by a programmer to variables, classes and methods which help to distinguish one from the other.

SYNTAX

Syntax is a set of rules that should be followed by the programmer for proper execution of the code.

- Identifiers must have at least one character.
- Identifiers are case sensitive.
- Identifier can't be same as Reserved words
- The first character in a identifier should be an alphabet, underscore, or dollar sign.It should not be a digit.
- The rest of the characters can be any combinations of alphabets,digits, underscore, or dollar sign.

Examples of Valid Identifiers

Here are some valid identifiers:

- abc
- _abc
- a1b
- $abc
- abc123$

Examples of Invalid Identifiers

Here are some invalid Identifiers

- 5ab (Digit as first char is not allowed)
- ab-cd (dash is not allowed in identifiers)
- ab cd (spaces are not allowed)

KEYWORDS

Keywords are predefined identifiers by Java used for some specific purposes. Keywords cannot be used as variables. Various keywords present in java are

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	super
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	float	volatile

VAIRABLES

Variables are the names given to the memory location which stores a values of any type during the program execution.

Syntax:
datatype variable_name;
Here datatype represents type of value which is stored in the variable name like int, float etc, while writing variable names it should follow same rules as that of identifier.
In java Variables are divided into three types

1. Instance or object Variables
2. Static or class level Variables
3. Local Variables

Instance variables
Variables that are declared inside a class but outside methods or constructors are called instance variables.This varibales are also called as object varibales.Each object for that class has its own copy of each variables and thus, it will not effect the instance variable if one object changes the value of the variable.

Example

```
public class Car
{
String model_name;
int price;
}
```

Here model and price are instance variables of Car class.

Static variables
The variables that are declared with static keywords are called static variables.Static variables are also called as class variables.static means constant which mean the value remains constant for all the objects or instances created for that class.This variables are initialized only once. By using final keyword this variables can also be used as constants.

Example:

```
class Car
{
String model_name;
```

```
int price;
static int company_code=1001;
}
```

Here company_code is a static variable. Each object of Car class will share the company_code property.

Local variables

The variables which are declared inside method or blocks are called as local variables. The scope of local variables will be till the end of that block or method which means we cannot use those variabes outside the methods or blocks. Local variable reside in stack. Access modifiers are not used for local variable.

Example

```
Class Car{
String model_name;
int price;
float getDiscount(int temp)
{
double discount;
discount=temp*(25/100); // 25% discount to the total price
return discount;
}
}
```

Here discount is a local variable.

Assignment Statements and Assignment Expressions

```
class Car
{
int price;
int discount=10;
}
```

Here price is declared in the class but it is not assigned any value where as we have assigned a value of 10 to the discount variable using the assignment operator(=)

Syntax for assignment statements:

datatype variable = expression or value;

An ***expression*** involves computations of values, variables, and operators that finally evaluates to a value.

```
class Cal
{
int a = 10 , b = 5;
int sum;
sum = a + b ;
}
```

Here a+b is an expression which evaluates to a value that is assigned to the sum variable.

NAMED CONSTANTS

By names it specifies that the value once it gets assigned will not be changed during the program execution.

Syntax:
static final datatype name_of_identifier = constant_value;
final makes the identifier value unchangeable, static makes the identifier to be available without creating any instance to that class.

Example:
```
public class Area {
        static final double PI=3.14;
        public static void main(String[] args) {
        double radius=5,circle_area=0;
        System.out.println("Area of circle = "+ PI*radius*radius);
        }
}
```

Output:
```
compile:
run:
Area of circle = 78.5
BUILD SUCCESSFUL (total time: 1 second)
```

NAMING CONVENTIONS:
Naming conventions helps programs more readable by following general rules while naming variables, methods and classes etc.
Note: java is a case-sensitive language
This are some general rules followed in any programming language
1.While naming constants capitalise every letter in it.
2.Start with capital letter while writing class names.
3.Use underscore between words.
4.Use small letter while naming variables and methods.

Data types and operations

A data type is a set of data with values having predefined characteristics.
Example: Integer, floating point, character etc.
Data types are broadly divided into two types.
1. Primitive data types
2. Non-primitive data types

Primitive data types

There are eight primitive data types - byte, short, int, long, float, double, char and boolean.
byte, short, int, long involve whole numbers without any decimals where as float,double contains
decimal values.*boolean* contains TRUE or FALSE value. Char contains only one single character.

Example
```
public class Demo{
public static void main(String[] args)
{
int a=10;
float b=(float)10.50;
char c='a';
```

```
boolean d=false;
System.out.println("the int value is"+a);
System.out.println("the float value is"+b);
System.out.println("the char value is"+c);
System.out.println("the boolean value is"+d);
}
}
```

Output:

```
run:
the int value is 10
the float value is 10.5
the char value is a
the boolean value is false
BUILD SUCCESSFUL (total time: 1 second)
```

Range of numeric data types in Java

Type	Size	Range
byte	8 bits	-128 .. 127
short	16 bits	-32,768 .. 32,767
int	32 bits	-2,147,483,648 .. 2,147,483,647
long	64 bits	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807
float	32 bits	$3.40282347 \times 10^{38}$, $1.40239846 \times 10^{-45}$
double	64 bits	$1.7976931348623157 \times 10^{308}$, $4.9406564584124654 \times 10^{-324}$

Non-primitive or complex data types.
User defined data types are called as non-primitive data types.
Example: Arrays, classes etc.
We will learn more about non primitive data types in next chapters.

Numeric Type Conversions
Numeric type conversions means converting one data type to other data type
Example:
Float to int,double to int etc
There two type of conversions in java
1.Automatic or implicit type conversion
2. Explicit type conversion

Automatic or implicit type conversion
Java automatically converts one data type to other whenever required. In simple words converting lower data types to higher data types.
The following data type conversion is compatible
Byte – short – int – long – float – double

Example

```
class Conversion
{
  public static void main(String[] args)
  {
    int temp1 = 123;
    // automatic type conversion
    float temp2 = temp1;
    System.out.println("Integer value =  "+temp1);
    System.out.println("Float value = "+temp2);
  }
}
```

Output:

```
compile:
run:
Integer value =  123
Float value = 123.0
BUILD SUCCESSFUL (total time: 1 second)
```

Explicit type conversion
If the conversion is not done automatically then we need to explicitly
convert data using cast operator "()". In simple words converting higher data type to lower data type. There will be some loss in information during explicit type conversion.
The following data type conversion is compatible
Double – float – long – int – short – byte

Example

```
 class Conversion
{
   public static void main(String[] args)
   {
    double temp1 = 123.04;
     // explicit type conversion
     long temp2 = (long)temp1;
     System.out.println("double value =  "+temp1);
     System.out.println("long value = "+temp2);
   }
}
```

Output:

```
compile:
run:
double value =  123.04
long value = 123
BUILD SUCCESSFUL (total time: 1 second)
```

Operators

An operator is a symbol that helps the compiler to do a particular logical or mathematical operation.

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators

Arithmetic operators

Given: A=2, B=1

Operator	Description	Example
+	Addition, adds two values	A + B=3
-	Subtracttion,subtracts rightside value from leftside value	A - B=1
*	Multiplication,multiplices two values.	A * B = 2
/	Division,divides leftisde value by rightside value	A / B = 2
%	Modulo returns remainder after division.	A % B = 0
++	Increases the value by 1.	A++ = 3
--	Decreases the value by1.	B-- = 0

Relational Operators

Given: A=2, B=1

Operator	Description	Example
==	Returns true if two values are equal else false	(A == B) is false
!=	Returns true if two values are not equal else false	(A != B) is true.
>	Returns true if left side value is strictly greater than right side value else false.	(A > B) is true.
<	Returns true if left side value is strictly lesser than right side value else false.	(A < B) is false.
>=	Returns true if left side value is greater than or equal to right side value else false.	(A >= B) is true.
<=	Returns true if left side value is lesser than or equal to right side value else false.	(A <= B) is false.

Bitwise operators

Given: A=5 (101) and B=3 (011)

Operator	Description	Example
&	Returns bit by bit AND of two values	A & B = 1 //001
|	Returns bit by bit OR of two values	A | B = 7 //111
^	Returns bit by bit XOR of two values	A ^ B = 6 /110
~	Returns compliment of each bit	~A = -6(2's compliment)
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 1 = 10 //1010
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	B >> 1 = 1 //001

Logical Operators

Given A=0 and B=1

Operator	Description	Example
&&	Return true if both values are non zero else false	A && B= false
||	Return true if any of the two values is non zero else false	A || B=true
!	Return True if the value is zero else false	!A = true

Example

```
public class JavaApplication3 {
   public static void main(String[] args) {
     int a=2,b=1;
     System.out.println("a + b =  "+(a+b));
     System.out.println("a - b =  "+(a-b));
     System.out.println("a * b =  "+(a*b));
     System.out.println("a / b =  "+(a/b));
     System.out.println("a % b =  "+(a%b));
     System.out.println("a ++ =  "+(a++));
     System.out.println("b-- =  "+(b--));

     a=2;
     b=1;
     System.out.println("a == b =  "+(a==b));
     System.out.println("a != b =  "+(a!=b));
     System.out.println("a > b =  "+(a>b));
     System.out.println("a < b =  "+(a<b));
     System.out.println("a >= b =  "+(a>=b));
     System.out.println("a <= b =  "+(a<=b));

     a=5;
     b=3;
     System.out.println("a & b =  "+(a&b));
     System.out.println("a | b =  "+(a|b));
     System.out.println("a ^ b =  "+(a^b));
     System.out.println(" ~a =  "+(~a));
```

```
        System.out.println("a << 1 =  "+(a<<1));
        System.out.println("b>>1 =  "+(b>>1));
    }
}
```

Output:

```
compile:
run:
a + b =  3
a - b =  1
a * b =  2
a / b =  2
a % b =  0
a ++ =  2
b-- =  1
a == b =  false
a != b =  true
a > b =  true
a < b =  false
a >= b =  true
a <= b =  false
a & b =  1
a | b =  7
a ^ b =  6
 ~a =  -6
a << 1 =  10
b>>1 =  1
BUILD SUCCESSFUL (total time: 1 second)
```

String type and methods

In general string is a sequence of characters. But in Java, a string is an object that represents a sequence of characters. The class used to create string object is *java.lang.String* class .
There are two way to create a string objects in java

1. By using new keyword
   ex: String s = new String("Hello world");
2. By using string literal or double quotes
   ex: String s = "Hello world";

Java String Methods

1. Length(): It returns the count of the total characters present in the string.
2. compareTo(): It return 0 if two strings matches to each other, a positive number if first string is greater than other, a negative number if first string is lesser than second string.
3. concat():This methods concats/combines a specific string at the end of other string.
4. IsEmpty():It returns true if the string is empty otherwise false.
5. contains():It returs true if a specific string is found as a sub string in the given string.
6. toUpperCase():Converts each character in the string to uppercase.
7. toLowerCase(): Converts each character in the string to lowercase.
8. charAt():It returns character at i'th position in the given string.

Example:

```
 public static void main(String[] args)
{
String s="Hello world",temp="abcd";
System.out.println("length of string = "+s.length());
System.out.println("compare s==temp = "+s.compareTo(temp));
System.out.println("concat temp at the end of s = "+s.concat(temp));
System.out.println("Hello substr in s = "+s.contains("Hello"));
System.out.println("s to upper case = "+s.toUpperCase());
System.out.println("s to lower case = "+s.toLowerCase());
}
```

Output

```
compile:
run:
length of string = 11
length of string = -25
length of string = Hello worldabcd
length of string = true
length of string = HELLO WORLD
length of string = hello world
BUILD SUCCESSFUL (total time: 0 seconds)
```

Topic 03 – Control Structures Part I

Lesson Learning Outcomes

- What is control statement ad explain different types of control statements present in java?
- Explain briefly about IF,NESTED IF statements with examples?
- Explain about switch statement with an example?
- What is the difference between IF ELSE IF and switch statements?
- What is ternary operator and where it can be used?

Control statements are the statements that define the flow of programs. There are 3 types of control statements in Java: Selection, iteration, and jump statements.

Control Structures

Statements in Java can be either simple statements or compound statements. Simple statements, such as assignment statements and subroutine call statements, are the basic building blocks of a program. Compound statements, such as while loops and if statements, are used to organize simple statements into complex structures, which are called control structures because they control the order in which the statements are executed.
Control statements are the statements that define the flow of programs. There are 3 types of control statements in Java: Selection, iteration, and jump statements

Sequence
A sequence is one of the basic logic structures in computer programming. In a sequence structure, an action, or event, leads to the next ordered action in a predetermined order. The sequence can contain any number of actions, but no actions can be skipped in the sequence. In simple terms executing the program from top to bottom without skipping any line.

Selection
A selection (also called a decision) is also one of the basic logic structures in computer programming. In a selection structure, a question is asked, and depending on the answer, the program takes one of two courses of action, after which the program moves on to the next event.

Iteration
We will talk more about iteration and jump statements in next chapters.

Boolean Data Type

The Boolean data type has two possible values, either true or false.
Default value: false. They are usually used for true/false conditions.

Example:

```
 public static void main(String[] args)
{
        // TODO code application logic here
        boolean temp1=true;
        System.out.println(temp1);
        }
```

Output:
true

Simple IF
It is used to decide whether a particular block of code needs to be executed or not i.e. if a given condition is true then a block of statement is executed otherwise not.

Syntax:

```
if(condition)
{
  //block of code to be executed
}
```

Example:

```
public static void main(String[] args) {
        // TODO code application logic here
        int x=15;
        if(x>10)
        System.out.println("x is greater than 10");
        System.out.println("I am outside if block");
}
```

Output:
X is greater than 10
I am outside if block

IF… ELSE
The if-else statement has two blocks to codes, depending on the value of condition. If the condition is true, then if block executes otherwise else block executes.
Note: else statement can only be used with if.

Syntax

```
if (condition)
{
//block of code
}
else
{
//block of code
}
```

Example:

```
public static void main(String[] args) {
        // TODO code application logic here
        int x=5;
        if(x>10)
        System.out.println("x is greater than 10");
        else
        System.out.println("x is less than 10");
        System.out.println("I am outside if block");
}
```

Output:

X is less than 10
I am outside if block

NESTED IF

Nested, as the name suggests, something that is inside another. In Java also we are provided with the option of Nested if-else, where Nested if statement is an if statement inside another.
So you will observe multiple if block in this case. And its only when the outer if block is true then the inner if block is executed.

Syntax

```
if(condition)
{
        //code 1
    if(condition)
{
        //code 2
         }
}
```

Example:

```
public static void main(String[] args)
 {
        // TODO code application logic here
        int x=50;
        if(x>10)
                {
                        If (x>100)
                        {
                                System.out.println("x is greater than 10 and 100");
                        }
                        else
                        {
                                System.out.println("x is greater than 10 but less than 100");
                        }
                }
        else
        System.out.println("x is less than 10");

}
```

Output:

X is greater than 10 but less than 100

IF ELSE IF
if-else-if statement is used to check multiple conditions. In this statement we have only one "if" and one "else", however we can have multiple "else if". It is also known as if else if ladder

Syntax
```
if(Condition 1){
// Body of if statement
}
else if(Condition 2){
//Body of else if
}
else{
// Body of else statement
}
```

Note: In if-else-if statement, as soon as the condition is met, the corresponding set of statements gets executed, rest gets ignored. If none of the conditions is met then the statements inside else get executed.

Example:
```
public static void main(String[] args)
{
        // TODO code application logic here
        int x=3;
        if(x>10)
        System.out.println("x is value greater than 10");
        else if(x>8)
        System.out.println("x is value is greater than 8");
        else if(x>5)
        System.out.println("x value is greater than 5");
        Else
        System.out.println("x value is less than 5");
}
```

Output:
X value is less than 5

Switch Statement
A switch statement allows you to test the value of an expression and, depending on that value, to jump directly to some location within the switch statement. Only expressions of certain types can be used. The value of the expression can be one of the primitive integer types int, short, or byte. It can be the primitive char type. The positions that you can jump to are marked with case labels that take the form:

"case⟨constant⟩:".

This marks the position the computer jumps to when the expression evaluates to the given ⟨constant⟩. As the final case in a switch statement you can, optionally, use the label **"default:",** which provides a default jump point that is used when the value of the expression is not listed in any case label.A Switch Statement, as it is most often used, has the form:

Syntax

switch (⟨expression⟩)
{
case⟨constant-1⟩:⟨statements-1⟩
break;
case⟨constant-2⟩:⟨statements-2⟩
break;.. //
(more cases)
case⟨constant-N⟩:⟨statements-N⟩
break;
default: // optional default case⟨statements-(N+1)⟩
}

The break statements are technically optional. The effect of a break is to make the computer jump to the end of the switch statement. If you leave out the break statement, the computer will just forge ahead after completing one case and will execute the statements associated with the next case label. Note, by the way, that the constants in the case labels don't have to be in any particular order, as long as they are all different:

Example

```
public static void main(String args[]){
    int temp=10;
    switch(temp)
    {
        case 1:System.out.println("Case1 ");
            break;
        case 5:System.out.println("Case5 ");
            break;
        case 10:System.out.println("Case10 ");
            break;
        case 50:System.out.println("Case50 ");
            break;
        default:System.out.println("Default ");
    }
  }
```

Output:

Case10

Example2:

```
public static void main(String args[]){
    int temp=10;
    switch(temp)
    {
        case 1:System.out.println("Case1 ");
        case 5:System.out.println("Case5 ");
        case 10:System.out.println("Case10 ");
        case 50:System.out.println("Case50 ");
        default:System.out.println("Default ");
    }
  }
```

Output:
Case10
Case 50
Default

Conditional operator(?:)

This ternary operator or conditional operator can be used in if / else conditional statements, when the result is to be returned based upon one conditions.

Syntax:
Conditional expression? expression1 : expression2 ;
If the conditional expression is true , return expression1 else return expression2 if its false.

Example 1:

```
public static void main(String[] args)
  {
     int a = 20, b = 30, min;
     min = (a < b) ? a : b;
     System.out.println("Minimum is = " + min);
  }
```

Output:
Minimum is =20

Example 2:

```
public static void main(String[] args)
  {
     int a = 20, b = 30, max;
     max = (a > b) ? a : b;
     System.out.println("Maximum is = " + max);
  }
```

Output:
Maximum is =30

Topic 04 – Control Structures Part II

Learning Outcomes:

- What is repetition statement and discuss about loop design strategies.
- Explain briefly about different types of loops with examples.
- Explain difference between counter-controlled loop and sentinel value loop.
- With examples explain when to use which loop.
- Explain about break and continue keywords with examples.

Iteration or Repetition:

A repetition statement is used to execute a group or block of programming instructions multiple times.

Loop Design Strategies

There are several steps that we can follow when it comes to loop designs. To properly understand a loop, we must consider

- pre-condition
- post-condition
- loop variant
- loop invariant

We must properly define all four criteria to ensure a loop that is correct and terminates. Pre-condition must be properly defined and true before the loop executes. Post-condition must be true upon loop termination. Loop variant (exit condition) which controls the execution of a loop must be properly defined to ensure loop termination.

The most important aspect of a loop is the loop invariant. The loop invariant is a statement that is true before and after each iteration of the loop. While the values of variables may change, it is the truth of the loop invariant that does not vary. Many statements can be made that are true both before and after each iteration of the loop, but the loop invariant must relate the work that the loop does such that when the loop is finished the truth of the loop invariant implies the desired result.

There are two types of looping logics -

1) Sentinel Controlled Loop and
2) Counter Controlled Loop.

Sentinel-Controlled Loop

Sometimes, loop control may need to be based on the value of what we are processing. In this case, we would use a sentinel-controlled loop. Sentinel-controlled loop is sometimes called indefinite repetition because it is not known in advance how many times the loop will be executed. It is a repetition procedure for solving a problem by using a sentinel value (also called

a signal value, a dummy value or a flag value) to indicate "end of data entry". The sentinel value itself is not a part of the processed data.

An example of when we would use sentinel-controlled repetition is when we are processing data from a file and we do not know in advance when we would reach the end of the file.

Read and display the content of a file.

```
String line;
BufferedReader br = new BufferedReader(...);
while (line = br.readLine() != null)
  System.out.println(line);
```

Count-Controlled Loop

Count-controlled loop requires

- control variable (or loop counter)
- initial value of the control variable
- increment (or decrement) by which the control variable is modified each iteration through the loop
- condition that tests for the final value of the control variable

A count-controlled repetition will exit after running a certain number of times. The count is kept in a variable called an index or counter. When the index reaches a certain value (the loop bound) the loop will end.

Count-controlled repetition is often called definite repetition because the number of repetitions is known before the loop begins executing. When we do not know in advance the number of times we want to execute a statement, we cannot use count-controlled repetition. In such an instance, we would use sentinel-controlled repetition.

Display the character and its' ASCII value for all lower-case characters.

```
for (char x = 'a'; x <= 'z'; x++)
  System.out.println(x + " = " + (int) x);
```

Output:
Displays number from 66 to 92 because value of ‘a ‘is 66.

While statements

It repeatedly executes the block code , or keep on performing the given task , provided the condition remains “TRUE”. Meaning, as soon as the condition becomes “FALSE” it stops executing.

Syntax:

```
while ( expression )
 {
```

```
 //body of loop statements;
}
```

Example:

```
int count =1;
int product ;
System.out.println(“ Table of 2 “ );
while (count <=10 )
{
 product  = 2 * count ;
System.out.println(“ 2 X  “ + count + “=  “ + product );
count ++;
}
```

Output:

```
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20
```

Do - While statements

In the case of while loop, it executes only when the condition is true , but what if you want to execute the block code at least once , even if the condition is not met! In such cases we use do — while loop, it's almost the same as the while loop , the only difference is that the condition is written in the bottom , and the loop executes at least once no matter what!! In this case too, it stops to execute if the condition becomes “false”.

Syntax:

```
do {
//body of loop statements
} while (  condition ) ;
```

Example:

```
int val = 0;
int val1;
do
{
val1 = val +1;
System.out.println( “ Value Is “ + val1 );
} while ( val > 0 );
```

Output :

```
Value Is 1
```

For Statement

Syntax

```
for ( initialization ; termination ; increment )
{
 // body statements ;
}
```

(i) initialization : expression / condition to start the loop . It executes **ONLY ONCE** at the beginning of the loop.
(ii) termination : expression / condition to end/terminate the loop. This is evaluated every time the loop has to iterate , and if the expression evaluates to be “false” the loop terminates.
(iii) Increment : invoked after each iteration , gives the value by which the expression has to be incremented after each loop. Is optional , loop will run even if its not mentioned .

Example:

```
Int [ ]  num = { 10,20,30,40,50 } ;
System.out.print(“ The numbers are :  “);
for ( int i = 0 ; i < num.length ; i++)
{
System.out.print(num[i] + “  “ );
}
```

Output:
10 20 30 40 50

Nested Loops

If a loop exists inside the body of another loop, it's called a nested loop. Here's an example of the nested for loop.

```
for (int i = 1; i <= 5; ++i)
{
   // codes inside the body of outer loop
          for (int j = 1; j <=2; ++j)
          {
           // codes inside the body of both outer and inner loop
           }
   // codes inside the body of outer loop
}
```

Here, a for loop is inside the body another for loop.
It should be noted that you can put one type of loop inside the body of another type. For example, you can put a while loop inside the body of a for loop.

Example:

```
public static void main(String[] args)
{

    for (int i = 1; i <= 5; ++i) {
      System.out.println("Outer loop iteration " + i);
      for (int j = 1; j <=2; ++j) {
        System.out.println("i = " + i + "; j = " + j);
      }
    }
  }
```

Output:

```
Outer loop iteration 1
        i = 1 j = 1
        i = 1 j = 2
Outer loop iteration 2
        i = 1 j = 1
        i = 2 j = 2
Outer loop iteration 3
        i = 3 j = 1
        i = 3 j = 2
Outer loop iteration 4
        i = 4 j = 1
        i = 4 j = 2
Outer loop iteration 5
        i = 5 j = 1
        i = 5 j = 2
```

Which LOOP to use?

- **while:** The loop must repeat until a certain "condition" is met. If the "condition" is FALSE at the beginning of the loop, the loop is never executed.The "condition" may be determined by the user at the keyboard. The "condition" may be a numeric or an alphanumeric entry. This is a good, solid looping process with applications to numerous situations.
- **do-while:** Operates under the same concept as the while loop except that the do-while will always execute the body of the loop at least one time. (Do-while is an exit-condition loop. The condition is checked at the end of the loop.) This looping process is a good choice when you are asking a question, whose answer will determine if the loop is repeated.
- **for:** The loop is repeated a "specific" number of times, determined by the program or the user. The loop "counts" the number of times the body will be executed. This loop is a good choice when the number of repetitions is known, or can be supplied by the user.

JUMP STATEMENTS

This includes break statements, continue statements.

4.7.1 The break statements

As suggested by the name , break statements are used to terminate the execution of the code below and move out of the block code. After terminating , executes the statement next to the block statements , unless it's a labelled break statement , in which case it executes code from the point below the label.Its of two types, Labelled and unlabelled break statements.

(a) Unlabelled break statements:
Usually used with switch-case , but can be used in any conditional statement ,i.e., also used to terminate iteration statements too.

Example: Search for a number in an array

```
int[] num ={10,20,30,40,50};
int search = 40;
boolean foundIt = false;
for (int i=0 ; i < num.length ; i++)
{
if (num[i] ==search) {
foundIt = true;
break ;
}
}
If(foundIt) {
 System.out.println( search + " at index " + i);
}
else {
System.out.println( search + " not in the array ");
}
```

OUTPUT : 40 at index 3

Explanation*:* set Boolean foundIt to false at first , then run a for loop to find the number in array num. If its found , set foundIt to true , and break the statement. Then flow of control move to the statement immediately following the for loop block code. If foundIt is true , then print the index , else print not found!!

(b) Labelled break statements:
The break statement terminates the labelled statement ; the flow of control moves to the statement immediately following the labelled (terminated) statement.

Example: finding a number in 2D array

```
int search = 122;
int i=0 , j=0;
boolean foundIt = false;
int[][] num={ { 33,64,31,15},{122,45,6,12},{165,123,10,20}};
search: //label is followed by the " : "
for (; i<num.length;i++){
```

```
for ( j=0 ;j<num[i].length ; j++) {
if (num[i][j] == search) {
 foundIt = true;
break search ;
}   }}
if (foundIt){
System.out.println(search + “at “ +i + “ , “ +j );
}
else{
 System.out.println( search +” not found” );
}
```

OUTPUT:122 at 1,0

The continue statement

It is used to skip the current iteration of the for , while or do-while loop. What it actually does is , it also terminates the loop thus preventing it from executing the statements below it , but, the only difference is that , the flow of control remains with the loop ,i.e., the loop will run again from the top statement of its block code.

(i) Unlabelled continue statement:

Example: count the no of even numbers present in an array num

```
int[] num ={ 12,2,3,4,52,100,7,1,20,21};
int count =0;
for ( int i=0 ; i < num.length ; i++)
{
if ( (num[i] % 2) != 0 )
continue;
Count++;
}
System.out.println(“No of even numbers in the array are “ + count };
```

OUTPUT

No of even numbers in the array are 6

Explanation: In the above example, we have first created an array num and assigned some values to it. Initially the count is kept zero. When the for loop begins its execution , i=0 , will denote the first index or position in the array and num[i] will be equal to 12. At this point as per our if condition expression will be evaluated to check whether its even or not , if it is not even i.e., remainder on dividing 12 by 2 is not equal to 0 , then the execution of for loop body will terminate or stop , and again the flow of control will remain with the for loop only , and move to the top , so before the loop begins to execute again , “ i “ is incremented and the loop runs and checks the if condition (first statement here) , if again its not equal to 0 , execution stops , for loop will enter next iteration. Now , if the remainder turns out to be 0 , i.e., then if statement will not be executed here , and the flow of control moves to the following statement i.e., to count++ , thus incrementing the count. This goes on , and after the loop is terminated , the result is printed out .

(ii) Labelled continue statement:
The continue statement skips the current iteration of the outer loop marked with the given label.
Example: find the sub-string in the string

```
String main="look for the substring in me ";
String sb = "sub";
boolean found = false;
int max = main.length() - sb.length() ;
test:for (int i=0 ; i<=max;i++)
{
int n = sb.length();
 int j=i ;
int k=0;
while ( n --!=0 )
{
if (main.charAt(j++) != sb.charAt(k++))
continue test;
}
found = true;
break true;
}
System.out.println(found ? "found it " : "not found" );
```

OUTPUT:
found it

Topic 05 – Classes and Methods Part I

Learning Outcome:

- What are classes and objects?
- What is an static class and explain it with an example?
- Explain about builtin classes with example?
- Explain about this reference with example?
- How to pass objects in java?

Defining Classes for Objects

Object: Object is an entity that has state and behavior. In the real world, we can find many objects around us like cars, books, animals, etc, and all these objects have a state and behavior.

An object has three properties:

- **State:** It represents the data of an object.
- **Behavior:** It represents the functionality of an object.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user but it is used internally by the JVM to identify each object uniquely.

If we consider the car as a class, then its state is the model name, color, and behavior is driving and breaking. We can also define the object as an instance of a class.

Class: We need to define a class before creating objects. A class is a blueprint or template for the object. It represents a group of objects which have common properties. A class in Java can contain:

- Fields
- Methods
- Blocks
- Constructors
- Nested class and interface

Syntax:

```
class class_name {
   // fields;
   // methods;
}
```

In a real-world scenario, we will be working with different classes in multiple Java files. If we define multiple classes in a single Java source file, it is recommended to save the file name with the class name which has the main() method.

Example:

```
class Car
{
        // Fields
        int ModelNumber;
        String Color;

        // Methods
        void Drive() {
                System.out.println("Car is moving");
   }

        void Break() {
                System.out.println("Breaks applied");
   }
}

class Test
{
        public static void main(String[] args) {
                Car Ford = new Car();
                Ford.Drive();
        }
}
```

Output:
Car is moving

Here we have created a Car class that has two data members ‘ModelNumber’ and ‘Color’. we created the object of the Car class in Test Class by using the ‘new’ keyword and we are calling the ‘Drive’ method.

Initializing objects: Initializing an object means storing data into the object. There are 3 ways to initialize an object in Java.

1. By reference variable
2. By method
3. By constructor

Example:

```
class Car {
   String Color;
   // Initialize by constructor
   Car() {
      Color = "Red";
   }

   void ChangeColor(String NewColor) {
      Color = NewColor;
   }
}

class Test {
```

```
    public static void main(String[] args) {
        Car Ford = new Car();
        System.out.println(Ford.Color);

        // Initialize by reference variable
        Ford.Color = "Blue";
        System.out.println(Ford.Color);

        //Initialize by method
        Ford.ChangeColor("Green");
        System.out.println(Ford.Color);
    }
}
```

Output:
Red
Blue
Green

Static Classes

In java, we cannot use the static keyword with a class unless it is an inner class. A static class that is created inside a class is called a static nested class in java.

Properties:

- A static nested class can be instantiated without instantiating its outer class.
- A static nested class can access static data members of the outer class but it cannot access non-static members.

Syntax:

```
class MyOuter {
   static class Nested_Demo {
   }
}
```

Example:

```
class OuterClass {
    static private String Color = "Red";
    static class InnerClass {
        void msg() {
            // Accessing static member of outer class
            System.out.println("Color is " +Color);
        }
    }
}

class MainClass {
    public static void main(String[] args) {
        // Instantiating inner class with out instantiating outer class
        OuterClass.InnerClass Obj = new OuterClass.InnerClass();
        Obj.msg();
```

```
    }
}
```

Output :
Color is Red
Here we have created an ‘InnerClass’ nested inside an ‘OuterClass’ which has a method ‘msg()’ that accesses the static member of the outer class ‘Color’. If you observe the main class, we have created the inner class object without instantiating the outer class.

Static variables

The variables that are declared with static keywords are called static variables.Static variables are also called as class variables.static means constant which mean the value remains constant for all the objects or instances created for that class.This variables are initialized only once. By using final keyword this variables can also be used as constants.

Example:

```
class Car
{
String model_name;
int price;
static int company_code=1001;
}
```

Here company_code is a static variable. Each object of Car class will share the company_code property.

Built-in classes

Java provides libraries of predefined classes that are free to use. These libraries contain packages and classes to developers which help in reducing the effort of writing code.
In order to use a particular class or package from the library, we use the ‘import’ keyword

Syntax:

```
import package.name.Class;   // Import a single class
import package.name.*;   // Import the whole package
```

List of Library Classes in Java

Library classes	Purpose of the class
Java.io	Used for input and output functions.
Java.lang	Used for character and string operation.
Java.awt	Used for windows interface.

Java.util	Used for developing utility programming.
Java.applet	Used for the applet.
Java.net	Used for network communication.
Java.math	Used for various mathematical calculations like power, square root, etc.

Example:

```
import java.util.Arrays;
public class SortArray {
   public static void main(String[] args) {
      int[] arr = {13, 7, 6, 45, 21, 9, 101, 102};
      Arrays.sort(arr);
      System.out.printf("Sorted array : %s", Arrays.toString(arr));
   }
}
```

In the above example, we have imported the ‘Arrays’ class from java.util and then sorted an array of elements using the ‘sort’ method.

Output:

6 7 9 13 21 45 101 102

Passing Objects in java

In Java, it is always ‘pass by value’. It means whenever we send arguments to a method, values assigned to these arguments are passed and any changes to these values don’t reflect the original arguments. Since objects are reference variables, java does some interesting things.

In general, objects are reference variables that store the location of the created object. When we pass an object as an argument to methods, java follows the same procedure of pass by value, which creates another variable that stores the value of the argument which is the location of the passed object itself. Thus any changes that are implemented on the properties of the object inside the method will change the original object.

Example:

```
class Rectangle {
   int Length, Width;

   Rectangle(int l, int w) {
      Length = l;
      Width = w;
   }

   void ChangeValues(Rectangle R, int l, int w) {
      R.Length = l;
      R.Width = w;
   }
```

```
    void PrintArea(Rectangle R) {
        int Area = R.Length * R.Width;
        System.out.println("Area of rectangle: " +Area);
    }
}

class Test {
    public static void main(String[] args) {
        Rectangle R = new Rectangle(3, 4);

        //passing object to method
        R.ChangeValues(R, 4, 5);
        R.PrintArea(R);
    }
}
```

Output:
Area of rectangle is 20

this Keyword

This reference: The ‘this’ keyword refers to the current object in a method or constructor. The ‘this’ keyword is frequently used to differentiate between class attributes and parameters with the same name.

‘This’ keyword is used for

- Referencing current class instance variables.
- Invoking the current class constructor or method.
- Returning the current class instance.
- Passing as an argument to the constructor or method.

Example:

```
class Rectangle {
    int Length, Width;

    // invoking constructor
    Rectangle() {
        this(3, 4);
    }

    Rectangle(int Length, int Width) {
        //assigning parameter values to class attributes
        this.Length = Length;
        this.Width = Width;
    }

    void GetValues() {
        System.out.println("Length : " +Length);
        System.out.println("Width : " +Width);
    }
```

```
}

class Test1 {
   public static void main(String[] args) {
      Rectangle R = new Rectangle();
      R.GetValues();
   }
}
```

Output:
Length 3
Width 4

Topic 06 – Classes and Methods Part II

Learning Outcome:

- What are methods?
- How to declare and call methods?
- Explain the difference between void and value returning methods?
- Explain about different types of constructors with example?
- Explain about overloaded methods and constructors?

Methods

Java Method is a block of statements in or commands which are used to run a particular task and return the result or value to the caller, it can also perform a task without returning any value. They also provide the user to use a particular code again without retyping. A Java method must always be a part of a class in Java

Method Declaration
There are 6 components to declare a method in Java.

1. Access Specifier
An access specifier in Java defines the access type of the method, i.e. it defines in which part of the program it can be accessed.

- Public — In this type, the method is accessible in all classes in the application.
- Protected — Here, the method is accessible in the class in which it is defined and also in its subclasses.
- Private — Method is accessible only in the class it is defined in.
- Default — Method is accessible only within the same class and package in which it is defined.

2. Return Type
The return type defines the type of data that is returned by the method. It can return any kind of value like integer, float, double, etc and also it returns void when nothing is returned by the method.

3. Method Name
A method name is used to define the name of a method, it is the same as the naming of field types with a little difference in the convention. A method is invoked by its name.

4. Parameter list
The parameter list is a list with input parameters that are separated by commas, they are preceded by their data types within closed parenthesis, and also, if there are no parameters then there is an empty parenthesis ().

5. Method Body
The body of the Java method must be enclosed within braces, all the codes for the method are written in it. All the operations are performed inside a Java method.

6. Method Signature
A method signature consists of the method name and parameter list in Java i.e. number, type, and order of parameters.

Example:

```
public class BasicMethod {
static void staticmethod() {
System.out.println("This is static method. ");
}
public void nonstaticmethod() {
System.out.println("This is  non-static method. ");
}
public static void main(String[] args) {
staticmethod();
BasicMethod ob = new BasicMethod();
ob.nonstaticmethod();
}
}
```

Output:
This is static method
This is non static method

Calling a method

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**;**
The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method.
In the following example, myMethod() is used to print a text (the action), when it is called:

Example

```
public class MyClass {

   static void myMethod() {
      System.out.println("First Method");
   }

   public static void main(String[] args) {
      myMethod();
   }
}
```

Output:
First Method

Void Methods

If the return type is void then that will not return any value to the calling function

Example

```
public class MethodCall1 {

  public static void main(String[] args){
        int x = 1,y=2;
        addNumbers(x, y);
  }
  public static void addNumbers(int a, int b)
  {
        int res = a + b;
        System.out.println("value is"+res);
  }
}
```

Output:
Value is 3

Value Returning Methods

The methods can return values to the calling function the values can any of the data types like int, float, string or double etc. In the caller function we need to catch the value returned by the function with the same data type.

Example:

```
public class MethodCall1 {
  public static void main(String[] args){
        int x = 5;
        int y = 2;
        int z = 4;
        addNumbers(x, y);
        int res = addNumbers(y, z);
        System.out.print("The result of y + z is: " + res);
  }
  public static int addNumbers(int a, int b)
  {
        int res = a + b;
        return res;
  }
}
```

Output:
The result of y + z is 11

Passing Parameters by values
Information can be passed to methods as parameters. Parameters act as variables inside the method. Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.
The following example has a method that takes an int called **num** as parameter. When the method is called, we pass along a number, which is used inside the method to be multiplied by 2:

Example:
```
public class Math {
        static int multiplyBytwo(int number) {
        return number * 2;
        }
        public static void main(String[] args) {
        int result = multiplyBytwo(2);
        System.out.println("The output is: " + result);
        }
}
```

Output:
The output is 4

Constructors

A constructor is a special type of method in Java which is used for giving initial values to an object.

1. Every constructor is of the same name as that of its parent class
2. A constructor gets loaded after class loading
3. If we do not define a constructor for a class, then Java includes its own default constructor in the code.
4. There are two types of constructor
   Default constructor
   Parameterized Constructor

Default Constructor

A constructor that has no parameter is known as default constructor.

Example
```
        public class Student {
        // default constructor
        String name
        public Student() {
                this.name=”abc”
                }
        }
class demo{
        public static void main(String[] args)
        {
        Student s = new Student();
        System.out.println(s.name);
        }
}
```

Output:
abc

Parameterized Constructor

A constructor that has parameters is known as a parameterized constructor. If we want to initialize fields of the class with our own values, then we use a parameterized constructor.

Example

```
public class Student {
        // default constructor
        String name
        public Student(String temp) {
                this.name=temp;
                }
        }
class demo{
        public static void main(String[] args)
        {
        Student s = new Student(def);
        System.out.println(s.name);
        }
}
```

Output:
def

Method Abstraction and Stepwise Refinement

Method abstraction is achieved by separating the use of a method from its implementation. The user can use a method without knowing how it is implemented. The details of the implementation are encapsulated in the method and hidden from the user who invokes the method. This is also known as information hiding or encapsulation. If we decide to change the implementation, the user program will not be affected, provided that you do not change the method signature. We have already used the System.out.print method to display a string and the max method to find the maximum number. We know how to write the code to invoke these methods in your program, but as a user of these methods, you are not required to know how they are implemented.

The concept of method abstraction can be applied to the process of developing programs. When writing a large program, you can use the divide-and-conquer strategy, also known as stepwise refinement , to decompose it into subproblems. The subproblems can be further decomposed into smaller, more manageable problems.

Overloaded Methods

A class definition can provide multiple methods, including constructors, that have the same name but different signatures. The signatures defined for overloaded methods and constructors must be different from one another in at least one variation in the number, type, or mode of their parameters. However, note that a static method cannot overload an

instance method with an identical signature, because the method scope (determined by use of the STATIC option in the METHOD statement) does not count in the overloading.

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. If we have to perform only one operation, having the same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs.

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

Changing number of arguments
In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.
In this example, we are creating static methods so that we don't need to create instances for calling methods.

Example:

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Output:
22
33

Changing data type of arguments
In this example, we have created two methods that differ in data type. The first add method receives two integer arguments and second add method receives two double arguments.

Example:

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Output:
22
24.9

Constructor Overloading

Making multiple constructors for a single class, each having a different prototype is called Constructor overloading. This allows you to initialize the data members in multiple ways. Thus you can have more than 1 kind of object of the same class by passing different (number or datatype) of arguments during creation of the object.

Example

```
class area{
   //Data members-
    int side;
    int length;
    int breadth;
 public area(int s)// 1st constructor
   {
   side=s;
   }
   public area(int l,int b)// 2nd constructor
   {
    length=l;
    breadth=b;
   }
 int square_area()
   {
    return side*side;
    }
    int rectangle_area()
    {
    return length*breadth;
    }
}
class run
{
   public static void main(String args[])
    {
            area Square=new area(5); //object creation according to 1st constructor
            area Rectangle=new area(4,6);   // according to 2nd constructor
             System.out.println("Area of square="+Square.square_area());
            System.out.println("Area of rectangle="+Rectangle.rectangle_area());
            }
            }
```

Output:

Area of square=25
Area of rectangle=24

REFERENCE:

Herbert Schildt, Java: The Complete Reference,11th Edition, McGraw-Hill **Education, ISBN: 9781260440249**

Topic 07 – ARRAYS

Learning Outcome:

- What are arrays?
- How to declare and access arrays?
- How to search for an element in an array?
- How to sort the elements in an array?
- How to declare and access multidimensional arrays?
- How to pass arrays to methods?

Arrays

Java array is an object which contains elements of a similar data type. The elements of an array are stored in a contiguous memory location. Array is an index-based, the first element of the array is stored at the 0th index, and 2nd element is stored on 1st index and so on.

Example Instead of writing int a=1, int b=2, int c= 3 we can write int arr[]={1,2,3}.

How to declare an array

To use an array in a program, we must declare a variable to refer that array and specify it's data type.

datatype[] array;
or
datatype array[];

Creating new array

int array[]= new int[size];

To insert values in an array we must define its size. Arrays include values which are predefined or user defined. A program generally uses loop to take user defined values.

PREDEFINED
int array[]={1,2,3,4,5}

USER DEFINED
```
int array[]=new int[5];
for(int i=0;i<5;i++)
{
array[i]=x; /*x is the user defined value*/
}
```
The size of an array can also be user defined.
int array[]=new int [n]; /*n is the user defined value*/

How to Access Elements of an Arrays

Each and every element has its own corresponding index, which is used to access the array elements. The indexing of array element is done as 0, 1, 2……n-1. The first element of an array is at index 0 thus it is called zero element of an array. The final element has an index of n-1 if the size of array is n, it is done because there are n elements in an array and the first is indicated with 0 so last is indicated as n-1.

We can access the element of an array using the index number. Here is the syntax for accessing elements of an array,

// access array elements
array[index]

Example:

```
class Main {
 public static void main(String[] args) {

   // create an array
   int[] num = {1, 4, 15,12, 50};

   // access each array elements
   System.out.println("Accessing Elements of Array:");
   System.out.println("First Element: " + num[0]);
   System.out.println("Second Element: " + num[1]);
   System.out.println("Third Element: " + num[2]);
   System.out.println("Fourth Element: " + num[3]);
   System.out.println("Fifth Element: " + age[4]);
 }
}
```

Output:

```
Accessing Elements of Array:
First Element: 1
Second Element: 4
Third Element: 15
Fourth Element: 12
Fifth Element: 50
```

Variable length Argument Lists

Variable-length argument lists, makes it possible to write a method that accepts any number of arguments when it is called. For example, suppose we need to write a method named sum that can accept any number of int values and then return the sum of those values. We write the method as shown here:

```
public static int sum(int ... list)
{
   //write code here
}
```

int ... list

This statement declares list to be a variable length formal parameter. In fact, list is an array wherein each element is of type int, and the number of elements in list depends on the number of arguments passed.

Example:

```
public class VarArgs
{
  public static void main(String[] args)
  {
    System.out.println("The sum of 1 and 2 is " + sum(1, 2));
    System.out.println("The sum of 1, 2, and 3 is " + sum(1, 2, 3));
    System.out.println("The sum when no parameter is passed is "+ sum());
  }
  public static int sum(int... list)
  {
    int total = 0;
    // add all the values in list array
    for (int i = 0; i < list.length; i++)
    {
      total += list[i];
    }
    return total;
  }
}
```

Output:

```
The sum of 1 and 2 is 3
The sum of 1, 2, and 3 is 6
The sum when no parameter is passed is 0
```

Pass one dimensional arrays to methods

Just like other objects, arrays can be passed as parameters to methods. The following method takes a int array as an argument and prints the data stored in each element of array:

Example:

```
public class PassArray
{
  public static void main(String[] args)
  {
    final int SIZE = 3;
    int[] array = new int[SIZE];
   Scanner console = new Scanner(System.in);
    System.out.println("Enter " + n + " integers");
    for (int i = 0; i < n; i++)
    {
      array[i] = console.nextInt();
    }
    System.out.println("Numbers are :");
    printArray(array, SIZE);
  }
```

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

Output:
Enter 3 integers
1 2 3
Numbers are:
1 2 3

Searching Arrays

The simplest type of search is the sequential search. In the sequential search, each element of the array is compared to the key, in the order it appears in the array, until the desired element is found. If you are looking for an element that is near the front of the array, the sequential search will find it quickly. The more data that must be searched, the longer it will take to find the data that matches the key.

Algorithm:

- Step 1: Traverse the array
- Step 2: Match the key element with array element
- Step 3: If key element is found, return the index position of the array element
- Step 4: If key element is not found, return -1

Example:

```
import java.util.Scanner;
class LinearSearch
{
 public static void main(String args[])
 {
  int i, n, search, array[];
  Scanner in = new Scanner(System.in);
  System.out.println("Enter number of elements");
  n = in.nextInt();
  array = new int[n];
  System.out.println("Enter  " + n + " elements");
  for (int i= 0; i < n; i++)
  array[i] = in.nextInt();
  System.out.println("Enter value to search");
  search = in.nextInt();
  for (i = 0; i< n; i++)
  {
    if (array[i] == search)     /* Searching element is present */
    {
      System.out.println(search + " is present at location " + (i + 1) );
       break;
    }
```

```
  }
  if (i == n)  /* Element to search isn't present */
     System.out.println(search + " isn't present in array.");
 }
}
```

Output:
Enter number of elements
5
Enter 5 elements
10 11 12 13 14
Enter value to search
13
13 is present at location 4

Sorting Arrays

Sorting arranges data in a specific order. Computer data consists of records made up of one or more fields. To use data efficiently and perform various operations like searching, accessing, etc. it is advisable that this data is arranged in some specific order.

For Example, if there are numerous records of student data, then we could arrange this data depending on the student id or student name. This is termed as sorting. Hence sorting is essential to use the data more efficiently and easily.

Algorithm:

- STEP 1: START
- STEP 2: INITIALIZE array[]
- STEP 3: SET temp =0
- STEP 7: REPEAT STEP 8 to STEP 9 UNTIL i<array.length
  //for(i=0; i<array.length; i++)
- STEP 8: REPEAT STEP 9 UNTIL j<array.length
  //for(j=i+1;j<array.length;j++)
- STEP 9: if(array[i]>array[j]) then
  temp = array[i]
  array[i]=array[j]
  array[j]=temp
- STEP 11: PRINT "Elements of array sorted in ascending order"
- STEP 12: REPEAT STEP 13 UNTIL i<arr.length
  //for(i=0;i<arr.length;i++)
- STEP 13: PRINT arr[i]
- STEP 14: END

Program:

```
public class SortAsc {
   public static void main(String[] args) {

      //Initialize array
      int [] array = new int [] {15, 21, 80, 72, 1};
      int temp = 0;
      System.out.println("Elements of original array: ");
```

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

        //Sort the array in ascending order
        for (int i = 0; i < array.length; i++) {
            for (int j = i+1; j < array.length; j++) {
                if(arr[i] > array[j]) {
                    temp = array[i];
                    array[i] = array[j];
                    array[j] = temp;
                }
            }
        }
        //Displaying elements of array after sorting
        System.out.println("Elements of array sorted in ascending order: ");
        for (int i = 0; i < array.length; i++) {
            System.out.print(array[i] + " ");
        }
    }
}
```

Output:
Elements of original array:
15 21 80 72 1
Elements of array sorted in ascending order:
1 15 21 72 80

Multidimensional Arrays

The Java multidimensional arrays are arranged as an array of arrays i.e. each element of a multi-dimensional array is another array. The representation of the elements is in rows and columns. Thus, you can get a total number of elements in a multidimensional array by multiplying row size with column size.

So if you have a two-dimensional array of 2×3, then the total number of elements in this array = 2×3 = 5

We create a multidimensional array in Java by appending one set of square brackets ([]) per dimension you want to add.

Here is an example that creates a two-dimensional array:
int[][] intArray = new int[10][20];

This example creates a two-dimensional array of int elements. The array contains 10 elements in the first dimension, and 20 elements in the second dimension. In other words, this example creates an array of arrays of int elements. The array of arrays has space for 10 int arrays, and each int array has space for 20 int elements.

How to declare 2D array

The declaration of a 2D array is similar to that of a one dimensional array.

datatype[max_row][max_column] array;
or
datatype array[max_row][max_column];

Creating new array

int array=new int[max_row][max_column];
Similar to one dimensional array where values are either pre-defined or user defined. A program uses a loop to take user defined values.

PRE DEFINED
int arr[2][2]={1,2,3,4};

USER DEFINED

```
int arr []=new int[size][size]; /* size is user defined*/
for(int i=0;i<size;i++)
{
for(int j=0;j<size;j++)
{
arr[i][j]=x; /*where x is user defined*/
}
}
```

Each and every element in the 2D array has a set of corresponding indexes, which is used to access the array elements. The indexing of array elements is done as 0, 1, and 2……n-1 for I and similarly for j. The first element of an array is at index [0][0] thus it is called the zeroth element of an array. The final element has an index of [n-1].

Example:
Print elements of matrix

```
int[][] arr = { { 1, 2 }, { 3, 4 } };
for (int i=0;i<2;i++)
{
        for (int j=0;j<2;j++)
        {
        System.out.print(arr[i][j]+" ");
        }
System.out.println();
}
```

Output:
1 2
3 4

Pass 2-D arrays to Methods

Just like one-dimensional arrays, a two-dimensional array can also be passed to a method and it can also be returned from the method. The syntax is similar to one-dimensional arrays with an exception that an additional pair of square brackets is used.

Example: Transpose of a matrix

```
public class Transpose
{
        public static void main(String[] args)
        {
        int[] [] table= {{1,2,3},{4,5,6}};
            int[] [] result;
        System.out.println("Matrix Before Transpose : ");
        for(int i=0;i<table.length;i++)
                {
                for(int j=0;j<table[i].length;j++)
                System.out.print(table[i][j]+" ");
        System.out.println();
        }
        result=transpose(table);
        System.out.println("\nMatrix After Transpose :");
         for(int i=0;i<result.length;i++)
                {
                for(int j=0;j<result[i].length;j++)
                System.out.print(result[i][j]+" ");
                System.out.println();
                }
        }
        static int[][] transpose(int[][] a)
        {

                int[][] temp=new int[a[0].length][a.length];
                for(int i=0;i<a[0].length;i++)
                {
                for(int j=0;j<a.length;j++)
                        temp[i][j]= a[j][i];
                }
                   return temp;
        }
}
```

Output:

```
Matrix before transpose
1 2 3
4 5 6
Matrix after transpose
1 4
2 5
3 6
```

References

Herbert Schildt, Java: The Complete Reference, 11th Edition, McGraw-Hill Education, ISBN: 9781260440249.

Topic 08 – Inheritance and Polymorphism

Learning Outcomes

- What is inheritance and different types of inheritance?
- Explain about super keyword.
- What is polymorphism and different types of polymorphism?
- What is method overriding?
- Explain about object equals method.

Inheritance

In OOP, computer programs are designed in such a way where everything is an object that interacts with one another. Inheritance is an integral part of Java OOPs which lets the properties of one class be inherited by the other. It basically helps in reusing the code and establish a relationship between different classes.

As we know, a child inherits the properties of his parents. A similar concept is followed in Java, where we have two classes:

1. Parent class (Super or Base class)
2. Child class (Subclass or Derived class)

A class that inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

Syntax:
Now, to inherit a class we need to use **extends** keyword. In the below example, class Son is the child class and class DAD is the parent class. The class Son is inheriting the properties and methods of Mom class.

```
class Son extends DAD
{
//your code
}
```

Let’s see a small program and understand how it works. In this example, we have a base class Teacher and a subclass JavaTeacher. Since class JavaTeacher extends the properties from the base class, we need not declare these properties and methods in the subclass.

```
class Teacher{
String designation = "Teacher";
String collegeName = "abc";
void does(){
System.out.println("Teaching");
}
}
public class JavaTeacher extends Teacher{
String mainSubject = "java";
public static void main(String args[]){
JavaTeacher obj = new javaTeacher();
```

```
System.out.println(obj.collegeName);
System.out.println(obj.designation);
System.out.println(obj.mainSubject);
obj.does();
}
}
```

Output:
abc
Teacher
java
Teaching

Types of Inheritance

Single Inheritance
When a class extends to another class then it forms single inheritance. In the below example, we have two classes in which class A extends to class B that forms single inheritance.

Example:

```
class A{
   int a = 10;
   void show() {
      System.out.println("a = "+a);
   }
}
public class B extends A{
   public static void main(String[] args) {
   B b = new B();
   b.show();

   }
}
```

Output:
a=10
Here, we can notice that show() method is declared in class A, but using child class Demo object, we can call it. That shows the inheritance between these two classes.

Multilevel Inheritance
When a class extends to another class that also extends some other class forms a multilevel inheritance. For example a class C extends to class B that also extends to class A and all the data members and methods of class A and B are now accessible in class C.

Example:

```
class A{
   int a = 10;
   void show() {
      System.out.println("a = "+a);
   }
}
```

```
class B extends A{
   int b = 10;
   void showB() {
      System.out.println("b = "+b);
   }
}

public class C extends B{
public static void main(String[] args) {
   C c = new C();
   c.show();
   c.showB();
   }
}
```

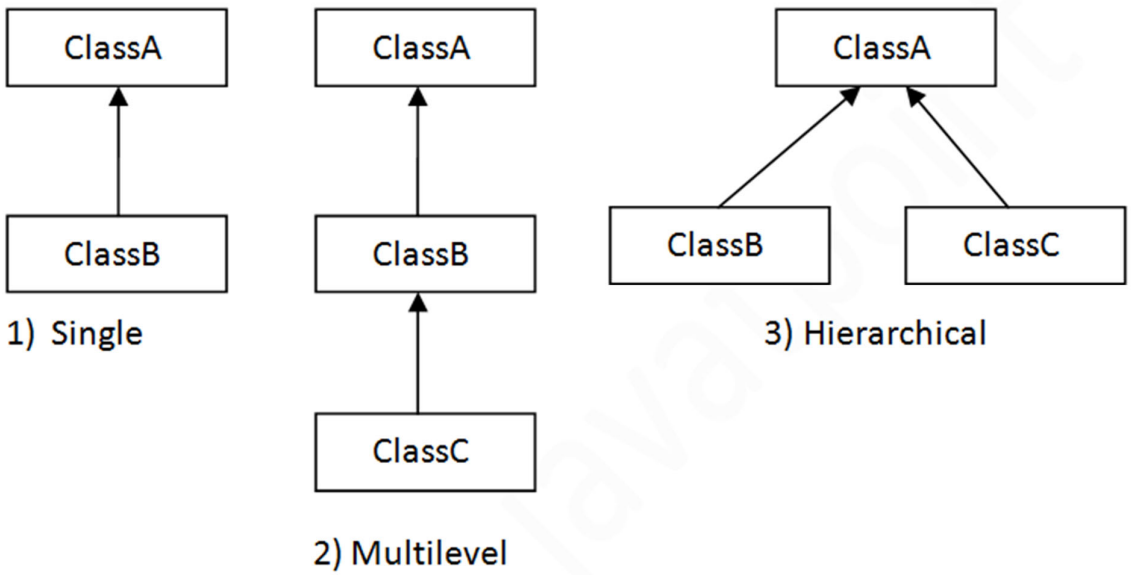
Output:
a=10 b=10

Hierarchical Inheritance
When a class is extended by two or more classes, it forms hierarchical inheritance. For example, class B extends to class A and class C also extends to class A in that case both B and C share properties of class A.

Example:

```
class A{
   int a = 10;
   void show() {
      System.out.println("a = "+a);
   }
}
class B extends A{
   int b = 10;
   void showB() {
      System.out.println("b = "+b);
   }
}
public class C extends A{
   public static void main(String[] args) {
      C c = new C();
      c.show();
      B b = new B();
      b.show();
   }
}
```

Output:
a = 10 a = 10



Super keyword

In Java, super keyword is used to refer to the immediate parent class of a child class. In other words super keyword is used by a subclass whenever it need to refer to its immediate super class

Example of Child class referring Parent class property using super keyword

In this example we will only focus on accessing the parent class property or variables.

```
class Parent
{
   String name;

}
public class Child extends Parent {
   String name;
   public void details()
   {
      super.name = "Parent";  //refers to parent class member
      name = "Child";
      System.out.println(super.name+" and "+name);
   }
   public static void main(String[] args)
   {
      Child cobj = new Child();
      cobj.details();
   }
}
```

Output:
Parent and Child

In this example we will only focus on accessing the parent class methods.

```
class Parent
{
   String name;
   public void details()
   {
     name = "Parent";
       System.out.println(name);
   }
}
public class Child extends Parent {
   String name;
   public void details()
   {
       super.details();   //calling Parent class details() method
       name = "Child";
       System.out.println(name);
   }
   public static void main(String[] args)
   {
       Child cobj = new Child();
       cobj.details();
   }
}
```

Output:
Parent Child

Method Overriding

Declaring a method in subclass which is already present in parent class is known as method overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class. In this case the method in parent class is called overridden method and the method in child class is called overriding method.

Example:
Lets take a simple example to understand this. We have two classes: A child class Boy and a parent class Human. The Boy class extends Human class. Both the classes have a common method void eat(). Boy class is giving its own implementation to the eat() method or in other words it is overriding the eat() method.

The purpose of Method Overriding is clear here. Child class wants to give its own implementation so that when it calls this method, it prints Boy is eating instead of Human is eating.

```
class Human{
  //Overridden method
  public void eat()
  {
    System.out.println("Human is eating");
  }
}
class Boy extends Human{
  //Overriding method
  public void eat(){
    System.out.println("Boy is eating");
  }
  public static void main( String args[]) {
    Boy obj = new Boy();
    //This will call the child class version of eat()
    obj.eat();
  }
}
```

Output:
Boy is eating

Polymorphism

Polymorphism contains two words: Poly and Morphism. Poly means many and morphism means behaviour.

The ability of an object to take on different forms is known as polymorphism. Polymorphism is the capability of a method to do different things based on the object that it is acting upon.

There are two types of polymorphism in java:
1. Static Polymorphism also known as compile time polymorphism
2. Dynamic Polymorphism also known as runtime polymorphism

Compile time Polymorphism (or Static polymorphism)

Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading is an example of compile time polymorphism.

Example of static Polymorphism:
Method overloading is one of the way java supports static polymorphism. Here we have two definitions of the same method add() which add method would be called is determined by the parameter list at the compile time. That is the reason this is also known as compile time polymorphism.

```
class SimpleCalculator
{
   int add(int a, int b)
   {
       return a+b;
   }
   int  add(int a, int b, int c)
   {
       return a+b+c;
   }
}
```

```
public class Demo
{
  public static void main(String args[])
  {
            SimpleCalculator obj = new SimpleCalculator();
      System.out.println(obj.add(10, 20));
      System.out.println(obj.add(10, 20, 30));
  }
}
```

Output:
30
60

Runtime Polymorphism (or Dynamic polymorphism)

It is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime, that's why it is called runtime polymorphism.

Example:
In this example we have two classes ABC and XYZ. ABC is a parent class and XYZ is a child class. The child class is overriding the method myMethod() of the parent class. In this example we have a child class object assigned to the parent class reference so in order to determine which method would be called, the type of the object would be determined at run-

time. It is the type of object that determines which version of the method would be called (not the type of reference).

```
class ABC{
  public void myMethod(){
        System.out.println("Overridden Method");
  }
}

public class XYZ extends ABC{

  public void myMethod(){
        System.out.println("Overriding Method");
  }
  public static void main(String args[]){
        ABC obj = new XYZ();
        obj.myMethod();
  }
}
```

Output:
Overriding Method
When an overridden method is called through a reference of the parent class, then type of the object determines which method is to be executed. Thus, this determination is made at run time.
Since both the classes, child class and parent class have the same method animalSound. Which version of the method(child class or parent class) will be called is determined at runtime by JVM.

Object Equals Method

Java Object class is the super class of all the Java classes. All Java classes implements the Object class by default.
The equals() method of the Object class compares the equality of two objects. The two objects will be equal if they share the same memory address.

Syntax:
public boolean equals(Object obj)

The method parses a reference object as a parameter. It returns true if the objects are equal, else returns false.

It is also possible that an object is equal to another given object, then the equals() method follow the equivalence relation to compare the objects.

- Reflexive: If x is a non-null reference, the calling of x.equals(x) must return true.
- Symmetric: If the two non-null references are x and y, x.equals(y) will return true if and only if y.equals(x) return true.
- Transitive: If the three non-null references are x, y, and z, x.equals(z) will also return true if x.equals(y) and y.equals(z) both returns true.
- Consistent: If the two non-null references are x and y, the multiple calling of x.equals(y) constantly returns either true or false. It does not provide any information used in the comparison.

- For any non-null reference x, x.equals(null) returns false.

In short, for any non-null reference say x and y, it returns true if and only if both references refer to the same object.

Example:

```
public class ObjectComparisonExample
{
public static void main(String[] args)
{
//creating constructor of the Double class
Double x = new Double(123.45);
//creating constructor of the Long class
Long y = new Long(12345);
//invoking the equals() method
System.out.println("Objects are not equal, hence it returns " + x.equals(y));
System.out.println("Objects are equal, hence it returns " + x.equals(123.45555));
}
}
```

Output:

```
Objects are not equal, hence it returns false
Objects are equal, hence it returns true
```

We have created constructor of the Double and Long class and pass corresponding values, as an argument that is stored in their objects, respectively.

After that, in the first println statement, we have invoked equals() method and parse an object y as a parameter that compares the object x and y. It returns false because x holds the double value and y holds the long value that is not equal.

Similarly, in the second println statement, we have invoked equals() method and parse the same value as in the constructor of the Double class. It returns true because the object of double class i.e. x holds the same value as we have passed in the equals() method.

References

Herbert Schildt, Java: The Complete Reference,11th Edition, McGraw-Hill Education, ISBN: 9781260440249

Topic 09 – Files

Learning Outcomes:

- What are files and why they are used in programing?
- How to read and write to files?
- What is stream and different types of streams available in java?

Introduction

Most Programs cannot accomplish their goals without accessing external data. This data is retrieved from an input source. The results of a program are sent to an output destination. The generic notion of an input source can abstract many different kinds of input: from a disk file, a keyboard, or a network socket. Likewise, an output destination may be a disk file, on the screen, or a network connection. These abstractions are a clean way to deal with input/output (I/O) without having difference between the keyboard and network.

Create a Java File Object

To create an object of File, we need to import the java.io.File package first. Once we import the package, here is how we can create objects of file.

```
// creates an object of File using the path

File file = new File(String pathName);
```

Here, we have created a file object named file. The object can be used to work with files and directories.

Note: In Java, creating a file object does not mean creating a file. Instead, a file object is an abstract representation of the file or directory pathname (specified in the parenthesis).

Create files

To create a new file, we can use the createNewFile() method. It returns true if a new file is created.False if the file already exists in the specified location.

Example:

```
// importing the File class
import java.io.File;
class Main {
public static void main(String[] args) {
// create a file object for the current location
File file = new File("newFile.txt");
try {
// trying to create a file based on the object
boolean value = file.createNewFile();
if (value) {
System.out.println("The new file is created.");
```

```
}
else {
System.out.println("The file already exists.");
}
}
catch(Exception e) {
e.getStackTrace();
}
}
}
```

In the above example, we have created a file object named file. The file object is linked with the specified file path.

```
File file = new File("newFile.txt");
```

Here, we have used the file object to create the new file with the specified path.
If newFile.txt doesn't exist in the current location, the file is created and this message is shown.
The new file is created.

However, if newFile.txt already exists, we will see this message.
The file already exists.

Read files

To read data from the file, we can use subclasses of either InputStream or Reader.

Example:
Suppose we have a file named input.txt with the following content.
SAMPLE CONTENT.

```
// importing the FileReader class
import java.io.FileReader;
class Main {
public static void main(String[] args) {
char[] array = new char[100];
try {
// Creates a reader using the FileReader
FileReader input = new FileReader("input.txt");
// Reads characters
input.read(array);
System.out.println("Data in the file:");
System.out.println(array);
// Closes the reader
input.close();
}
catch(Exception e) {
e.getStackTrace();
}
}
}
```

Output:
Data in the file:
SAMPLE CONTENT

In the above example, we have used created an object of FileReader named input. It is now linked with the input.txt file.

FileReader input = new FileReader("input.txt");

To read the data from the input.txt file, we have used the read() method of FileReader.
Java write to files

Write to Files

To write data to the file, we can use subclasses of either OutputStream or Writer.

Example:

```
// importing the FileWriter class
import java.io.FileWriter;
class Main {
public static void main(String args[]) {
String data = "SAMPLE CONTENT";
try {
// Creates a Writer using FileWriter
FileWriter output = new FileWriter("output.txt");
// Writes string to the file
output.write(data);
System.out.println("Data is written to the file.");
// Closes the writer
output.close();
}
catch (Exception e) {
e.getStackTrace();
}
}
}
```

Output:
Data is written to the file.
Output.txt

SAMPLE CONTENT

In the above example, we have created a writer using the FileWriter class. The writer is linked with the output.txt file.

FileWriter output = new FileWriter("output.txt");

To write data to the file, we have used the write() method.

Here when we run the program, the output.txt file is filled with the following content.
This is the data in the output file.

Java uses the concept of a stream to make I/O operations on a file. So let's now understand what a Stream is in Java.

What is a Stream?

In Java, Stream is a sequence of data which can be of two types.

1. **Byte Stream**
   This mainly incorporates byte data. When an input is provided and executed with byte data, then it is called the file handling process with a byte stream.

2. **Character Stream**
   Character Stream is a stream which incorporates characters. Processing of input data with character is called the file handling process with a character stream.

Byte Streams
Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams, the most frequently used classes are, FileInputStream and FileOutputStream. Following is an example which makes use of these two classes to copy an input file into an output file −

Example:

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

Input.txt
Sample content.

Output.txt
Sample content

Character Streams

Java Byte streams are used to perform input and output of 8-bit bytes, whereas Java Character streams are used to perform input and output for 16-bit unicode.

Though there are many classes related to character streams but the most frequently used classes are,FileReader and FileWriter.Though internally FileReaderuses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

We can rewrite the above example, which makes the use of these two classes to copy an input file (having unicode characters) into an output file –

Example:

```
import java.io.*;
public class CopyFile {
  public static void main(String args[]) throws IOException {
    FileReader in = null;
    FileWriter out = null;

    try {
      in = new FileReader("input.txt");
      out = new FileWriter("output.txt");
      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
      }
    }
finally {
      if (in != null) {
        in.close();
      }
      if (out != null) {
        out.close();
      }
    }
  }
}
```

Input.txt
Sample content

Output.txt
Sample content

References

Herbert Schildt, Java: The Complete Reference, 11th Edition, McGraw-Hill Education, ISBN: 9781260440249

Topic 10 – Recursion

Learning Outcomes:

- What is recursion and why does it work?
- Difference between iterative and recursive algorithms?
- How to compute factorials using recursion?
- How to compute fibonacci sequence using recursion?

Introduction

Sometimes a problem is too difficult or too complex to solve because it is too big. If the problem can be broken down into smaller versions of itself, we may be able to find a way to solve one of these smaller versions and then be able to build up to a solution to the entire problem. This is the idea behind recursion

Recursion is a problem solving technique which involves breaking a problem into smaller instances of the same problem (also called sub problems) until we get a small enough sub problem that has a trivial solution. We can say that recursion is “defining a problem in terms of itself” as it involves a function calling itself with a base case to terminate the infinite loop.

For example, we can define the operation "find your way home" as:

1. If you are at home, stop moving.
2. Take one step toward home.
3. "Find your way home".

Here the solution to finding your way home is two steps (three steps). First, we don't go home if we are already home. Secondly, we do a very simple action that makes our situation simpler to solve. Finally, we redo the entire algorithm

Why Recursion Works

In a recursive algorithm, the computer "remembers" every previous state of the problem. This information is "held" by the computer on the "activation stack" (i.e., inside of each function's workspace).

Every function has its own workspace per call of the function.

A recursive algorithm always has two parts:

- **Base case**, which is the simplest, smallest instance of the problem, that can’t be decomposed any further. Base cases often correspond to emptiness – the empty string, the empty list, the empty set, the empty tree, zero, etc.
- **Recursive step**, which decomposes a larger instance of the problem into one or simpler or smaller instances that can be solved by recursive calls, and then recombines the results of those sub problems to produce the solution to the original problem.

It’s important for the recursive step to transform the problem instance into something smaller, otherwise the recursion may never end. If every recursive step shrinks the problem, and the base case lies at the bottom, then the recursion is guaranteed to be finite.

A recursive implementation may have more than one base case, or more than one recursive step.

Case Study 1: Computing Factorials

Factorial Program in Java: Factorial of n is the product of all positive descending integers. Factorial of n is denoted by n! For example:

1. 4! = 4*3*2*1 = 24
2. 5! = 5*4*3*2*1 = 120

Let's see the factorial Program using loop in java.

```
class FactorialExample{
 public static void main(String args[]){
  int i,fact=1;
  int number=5;//It is the number to calculate factorial
  for(i=1;i<=number;i++){
      fact=fact*i;
  }
  System.out.println("Factorial of "+number+" is: "+fact);
 }
}
```

Output:
Factorial of 5 is: 120

Factorial using recursion

```
public static long factorial(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n-1);
  }
}
public static void main(String[] args) {
    long x = factorial(4);
  System.out.println("Factorial of 4"+" is: "+x);
}
```

Output:
Factorial of 4 is: 24

In the recursive implementation the base case is n = 0, where we compute and return the result immediately: 0! is defined to be 1. The recursive step is n > 0, where we compute the result with the help of a recursive call to obtain (n-1)! Then complete the computation by multiplying by n.

Case Study 2 Fibonacci

The Fibonacci sequence gets its name from Italian mathematician, Leonardo Fibonacci. He introduced this series in Western Europe with his book Liber Abaci back in 1202. The Indian mathematics scene had seen the Fibonacci magic way back in 200 BCE, as evidenced by the works of Pingala. This representation of numbers also occupies a special place in the spheres of coding and computing.

It is a sequence of non-negative integers whose nth term, i.e., F_n is given by sum of last 2 terms, i.e., $F_{n-1}+F_{n-2}$. Base cases for this recursive relation is $F_0 = 0$ and $F_1 = 1$.

The Fibonacci sequence is the series of numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 …

The next number is found by adding up the two numbers before it:

- 2 is calculated by adding the two numbers preceding it (1+1),
- 3 is calculated by adding the two numbers preceding it (1+2),
- 5 is (2+3), and so on..

Here is an example of Fibonacci series: 0, 1, 1, 2, 3, 5, 8, and 13….etc.

In the above example, 0 and 1 are the first two terms of the series. These two terms are printed directly. The third term is made by adding the first two terms. In this case 0 and 1. So, we get 0+1=1. Hence 1 is printed as the third term. The next term is generated by using the second and third term and not using the first term.

Fibonacci using loop

```
class FibonacciExample1{
public static void main(String args[])
{
 int n1=0,n2=1,n3,i,count=10;
 System.out.print(n1+" "+n2);//printing 0 and 1

 for(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  System.out.print(" "+n3);
  n1=n2;
  n2=n3;
 }
}}
```

Output:
0 1 1 2 3 5 8 13 21 34

Fibonacci Using Recursion

```
public class FibonacciCalc{
   public static int fibonacciRecursion(int n){
   if(n == 0){
          return 0;
   }
   if(n == 1 || n == 2){
                  return 1;
          }
```

```
    return fibonacciRecursion(n-2) + fibonacciRecursion(n-1);
    }
public static void main(String args[]) {
    int maxNumber = 10;
    System.out.print("Fibonacci Series of "+maxNumber+" numbers: ");
    for(int i = 0; i < maxNumber; i++){
                    System.out.print(fibonacciRecursion(i) +" ");
            }
    }
}
```

Output:

Fibonacci Series of 10 numbers: 0 1 1 2 3 5 8 13 21 34

1. Take an input number. Checks for 0, 1, 2 and returns 0, 1, 1 accordingly because the Fibonacci sequence starts with 0, 1, and 1.
2. When input n is >=3, the function will call itself recursively. The call is done two times. Let's see an example for input of 4.
   fibonacciRecursion (4)
   It will recursively call fibonacciRecursion function for values 2 and 3
   fibonacciRecursion (2) \\ call for value 0 and 1
   fibonacciRecursion (0) = 0
   fibonacciRecursion (1) = 1
   fibonacciRecursion (3) \\ It will call for 1 and 2
   fibonacciRecursion (1) = 1
   fibonacciRecursion (2) \\ It will call for 0 and 1
   fibonacciRecursion (0) = 0
   fibonacciRecursion (1) = 1

Now result is added 0+1+1+0+1=3

References

Herbert Schildt, Java: The Complete Reference, 11th Edition, McGraw-HillEducation, ISBN: 9781260440249

Kaplan City Campus @ Wilkie Edge | 8 Wilkie Road, Level 2, Singapore 228095
Kaplan City Campus @ PoMo | 1 Selegie Rd, Level 7, Singapore 188306

6733 1877 info.sg@kaplan.com

KaplanSingapore

Assessment Criteria – Introduction to Programming using Java: Individual Assignment

Weightage : 80% (100 marks)
Submission Deadline : Session 12 (FT) / Session 7 (PT)

ASSESSMENT SUMMARY

Question 1 (25 Marks)

Write a program that takes a command-line argument N and prints an N-by-N triangular pattern like the one below.

```
* * * * * *
. * * * * *
. . * * * *
. . . * * *
. . . . * *
. . . . . *
```

Question 2 (25 Marks)

Implement a class called Account which defines a modified banking account that can be created with a given balance and account name. It must be able to perform the following operations in the public interface:

- Accepts an amount to deposit the balance
- Withdraws an amount from the balance
- Returns its balance
- Returns its name
- Displays its name and balance

Write a Java application called AccountSystem to simulate an instance of the class. You need to use deposit and withdrawal methods in this program.

Output - AccountSystem (run)

```
run:
Please Enter Account Name : Lucas Chong
Please Enter Your Account Balance : 5000
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1. Account Status
2. Withdrawal
3. Deposit
99. Exit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Please Select Above Option : 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Dear Lucas Chong,
**************************
Current Balance : 5000.0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Account Status
2. Withdrawal
3. Deposit
99. Exit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Please Select Above Option : 2
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Please enter amount you wish to withdraw :500
*******************************
your current bank balance is : 4500.0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Account Status
2. Withdrawal
3. Deposit
99. Exit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Please Select Above Option : 99
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Thank you for banking with us
BUILD SUCCESSFUL (total time: 35 seconds)
```

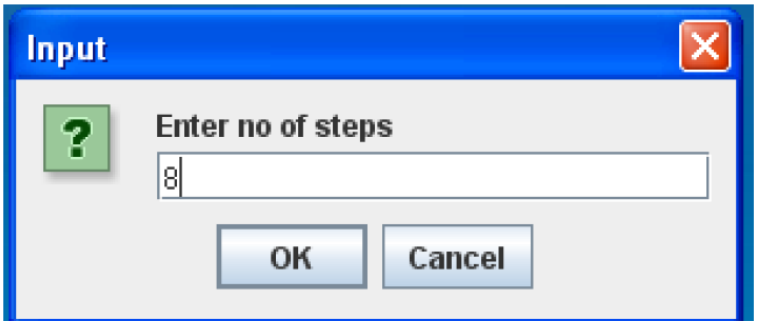
Output

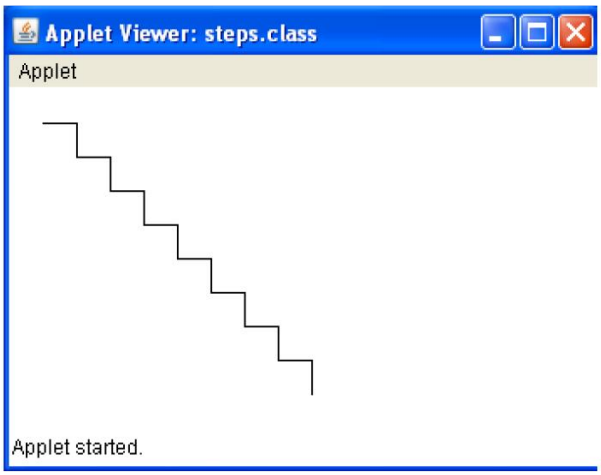
AccountSystem (run) | AccountSystem (run) #3

```
run:
Please Enter Account Name : Test 2
Please Enter Your Account Balance : -500
Please enter a valid account balance : 500
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1. Account Status
2. Withdrawal
3. Deposit
99. Exit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Please Select Above Option : 3
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Please enter amount you wish to deposit :-999
*******************************
Invalid amount entered
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Account Status
2. Withdrawal
3. Deposit
99. Exit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Please Select Above Option : 3
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Please enter amount you wish to deposit :99999
*******************************
your new balance is : 100499.0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Account Status
2. Withdrawal
3. Deposit
99. Exit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Please Select Above Option :
```

Question 3 **(25 Marks)**

Write a program to create the applet as shown below. The applet uses a paint method that draws steps based on an integer the user inputs via input dialog.





Question 4 **(25 Marks)**

Write a program called GradesAverage, which prompts user for the number of students, reads it from the keyboard, and saves it in an int variable called numStudents. It then prompts user for the grades of each of the students and saves them in an int array called grades. Your program shall check that the grade is between 0 and 100.

A sample session is as follow:

```
Enter the number of students: 3

Enter the grade for student 1: 55

Enter the grade for student 2: 108

Invalid grade, try again...

Enter the grade for student 2: 56

Enter the grade for student 3: 57

The average is 56.0
```

Marking Guide

Grading

The assignment will be graded according to:

Description	**Mark Allocated**	**Mark Awarded**
Question 1 - Java programming code - Code explanation - Testing (3 test cases with valid and invalid inputs) - Code optimization - Code efficiency	**25**	
Question 2 - Java programming code - Code explanation - Testing (3 test cases with valid and invalid inputs) - Code optimization - Code efficiency	**25**	
Question 3 - Java programming code - Code explanation - Testing (3 test cases with valid and invalid inputs) - Code optimization - Code efficiency	**25**	
Question 4 - Java programming code - Code explanation - Testing (3 test cases with valid and invalid inputs) - Code optimization - Code efficiency	**25**	
Total:	**100**	

Instruction

Your project should include the following:

1. Cover page
2. Project Document

Note: Project document should include the following:

- Explanation of overall java code
- Java Source Code with comments and screen shots of the console input and output

Requirements

- You need to create a folder for your Project Assignment
- All codes and output must be clearly documented and included in the Word Document
- You must use good indentations and comments
- You must implement this project by yourself.

---End of Assessment---

KAPLAN

Introduction to Programming using Java

Topic 01
Introduction to Java Programming

Learning Outcomes

- What is a programme and explain different types of Programming languages?
- Describe java specification like JRE,JVM and JDK etc?
- How to write and execute our first java programme?
- What are errors and explain different types of errors?
- Different types of comments in java programming?

KAPLAN

WHAT IS PROGRAMME?

A programme is a step-by-step instruction given to a computer to perform a specific task.

Programming languages are divided into two categories

- High-Level Programming Language
- Low Level or Machine Level Programming Language

HIGH-LEVEL PROGRAMMING LANGUAGE

- Any programme that we write in a human-understandable format is called high-level programming language.
- To run high-level programming language, we need a compiler or interpreter, which transforms it into a machine-understandable format.
- Example: Java, C++, and Python

LOW-LEVEL PROGRAMMING LANGUAGE

- Machine language, or machine code, is the lowest level of computer languages.
- It contains binary code, often generated by compiling high-level source code to run it.
- We do not need a compiler or interpreter. We need an assembler.

KAPLAN

JAVA SPECIFICATIONS

- Java Development Kit
- Java Virtual Machine
- Java Runtime Environment
- Application Program Interface
- Integrated Development Environment

KAPLAN

JAVA DEVELOPMENT KIT

- JDK is a combination of all the tools and executables that are used to debug, compile and execute a Java Program.
- JDK is a platform dependent.
- That is reason for having separate installers for Mac, Windows and Unix systems.

KAPLAN

JAVA VIRTUAL MACHINE

- JVM is the core of java programming language.
- At the point when we run a programme, JVM is liable for changing over Byte code to the machine-specific code.
- JVM is adjustable, and we can utilize java choices to modify it, for instance, allotting maximum and minimum memory to JVM.
- JVM is virtual because it gives an interface that doesn't rely on the underlying operating system and machine hardware.
- This independence from hardware and operating system is what makes Java programme write-once run-anywhere

KAPLAN

JAVA RUNTIME ENVIRONMENT

- JRE is the Implementation of JVM.
- It gives a stage to execute java programmes.
- JRE comprises of JVM and java binaries and different classes to execute programmes effectively.
- JRE doesn't contain any advancement devices like java compiler, debugger, and so forth.
- To execute any java program, we require JRE, however we needn't bother with JDK for running any java programme.

KAPLAN

APPLICATION PROGRAM INTERFACE

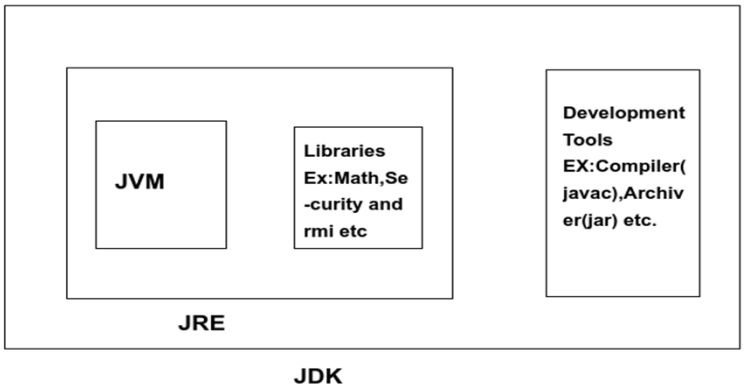
- API contains predefined interfaces and classes for developing Java application programmes.
- The Java language specification is still stable, but the API is still expanding time by time.
- At the Sun Java Website (java.sun.com), you can download and view the Java API's latest version.

KAPLAN

INTEGRATED DEVELOPMENT ENVIRONMENT

- IDE is a tool that can be used for creating, editing, debugging and compiling Java code.
- These tools additionally give dynamic and static assistance to helping any developer to create code.
- Example: NetBeans, Notepad, Eclipse

INTEGRATED DEVELOPMENT ENVIRONMENT



JDK = JRE + Development Tools
JRE = JVM + Libraries

FIRST JAVA PROGRAM

```
1. public class FristProgram
2. {
3.         public static void main(String[] args)
4.         {
5.                 System.out.println("Hello World");
6.         }
7. }
```

Output:

Hello World

ERRORS

- **Syntax or compile time errors**
  These are errors where the compiler finds something wrong with the program.
- **Run time errors**
  If there are no syntax errors, and if the errors occur at run time that stops the programme execution, it is run time error.
- **Logical errors**
  Logic errors occur when our programme compiles and runs, but it does not give the output that we are expecting.

COMMENTS

- Comments make the programme more readable.
- There are two types of comments;
  1. Single-line comments which start with two forward slashes (//)
  2. Multi-line comments which start with /* and end with */.

KAPLAN

Introduction to Programming using Java

Topic 02
Basic Computation

Learning Outcomes

- What are identifiers and rules for identifiers?
- What are different types of variables available in java?
- What are named constants and why to using naming conventions?
- What are different data types available in java?
- What are different operators available in java?
- What are different string methods available in java?

KAPLAN

IDENTIFIERS

- Identifiers are the names given by a programmer to variables, classes and methods.
- Identifiers must have at least one character.
- Identifiers are case sensitive.
- Identifier can't be same as Reserved words.
- The first character in a identifier should be an alphabet, underscore, or dollar sign. It should not be a digit.

KAPLAN

IDENTIFIERS

Examples of Valid Identifiers

- abc _abc
- a1b
- $abcabc123$

Examples of Invalid Identifiers

- 5ab (Digit as first char is not allowed)
- ab-cd (dash is not allowed in identifiers)
- ab cd (spaces are not allowed)

KAPLAN

VARIABLES

Variables are the names given to the memory location which stores a values of any type during the program execution.

Syntax:

datatype variable_name;

In java Variables are divided into three types

- Instance or object Variables
- Static or class level Variables
- Local Variables

KAPLAN

ASSIGNMENT STATEMENTS

```
class Car{
int price;
int discount=10;
}
```

Here price is declared in the class but it is not assigned any value where as we have assigned a value of 10 to the discout variable using the assignment operator(=)

Syntax for assignment statements :

datatype variable = expression or value;

KAPLAN

EXPLICIT TYPE CONVERSION

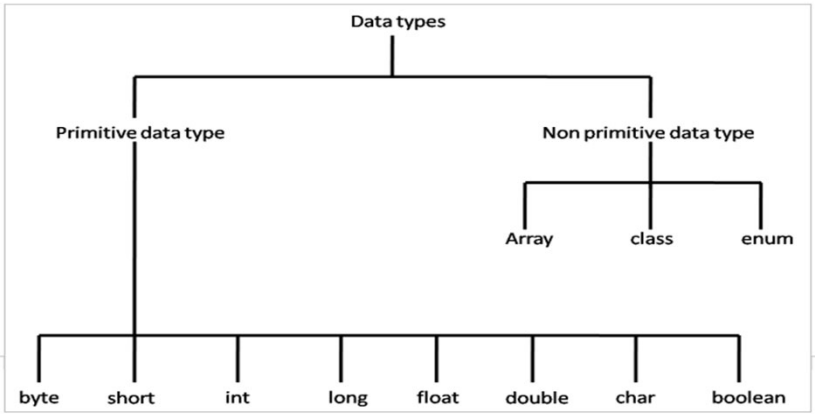
If the conversion is not done automatically then we need to explicitly convert data using cast operator “()”.

In simple words converting higher data type to lower data type.

There will be some loss in information during explicit type conversion.

The following data type conversion is compatible
Double – float – long – int – short – byte

DATA TYPES



DATA TYPE RANGES

Type	Size	Range
byte	8 bits	-128 .. 127
short	16 bits	-32,768 .. 32,767
int	32 bits	-2,147,483,648 .. 2,147,483,647
long	64 bits	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807
float	32 bits	$3.40282347 \times 10^{38}$, $1.40239846 \times 10^{-45}$
double	64 bits	$1.7976931348623157 \times 10^{308}$, $4.9406564584124654 \times 10^{-324}$

NUMERIC TYPE CONVERSIONS

Numeric type conversions means converting one data type to other data type

Example:

Float to int,double to int etc

There two type of conversions in java;

1. Automatic or implicit type conversion
2. Explicit type conversion

AUTOMATIC OR IMPLICIT TYPE CONVERSION

Java automatically converts one data type to other whenever required.

In simple words converting lower data types to higher data types.

The following data type conversion is compatible

Byte – short – int – long – float – double

EXPLICIT TYPE CONVERSION

If the conversion is not done automatically then we need to explicitly convert data using cast operator “()”.

In simple words converting higher data type to lower data type.

There will be some loss in information during explicit type conversion.

The following data type conversion is compatible
Double – float – long – int – short – byte

OPERATORS

Arithmetic operators

Lets consider A=2,B=1

Operator	Description	Example
+	Addition, adds two values	A + B=3
-	Subtracttion,subtracts rightside value from leftside value	A - B=1
*	Multiplication,multiplices two values.	A * B = 2
/	Division,divides leftisde value by rightside value	A / B = 2
%	Modulo returns remainder after division.	A % B = 0
++	Increases the value by 1.	A++ = 3
--	Decreases the value by1.	B-- = 0

KAPLAN

OPERATORS

Relational Operators

Lets consider A=2,B=1

Operator	Description	Example
==	Returns true if two values are equal else false	(A == B) is false
!=	Returns true if two values are not equal else false	(A != B) is true.
>	Returns true if left side value is strictly greater than right side value else false.	(A > B) is true.
<	Returns true if left side value is strictly lesser than right side value else false.	(A < B) is false.
>=	Returns true if left side value is greater than or equal to right side value else false.	(A >= B) is true.
<=	Returns true if left side value is lesser than or equal to right side value else false.	(A <= B) is false.

KAPLAN

OPERATORS

Bitwise operators

Lest consider A=5 //101 and B=3 //011

Operator	Description	Example
&	Returns bit by bit AND of two values	A & B = 1 //001
|	Returns bit by bit OR of two values	A | B = 7 //111
^	Returns bit by bit XOR of two values	A ^ B = 6 /110
~	Returns compliment of each bit	~A = -6(2's compliment)
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 1 = 10 //1010
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	B >> 1 = 1 //001

KAPLAN

OPERATORS

Logical Operators

Lets consider A=0 and B=1

Operator	Description	Example
&&	Return true if both values are non zero else false	A && B= false
||	Return true if any of the two values is non zero else false	A || B=true
!	Return True if the value is zero else false	!A = true

KAPLAN

STRINGS

A string is a sequence of characters.

But in Java, a string is an object that represents a sequence of characters. The class used to create string object is java.lang.String class .

There are two way to create a string objects in java;

1. By using new keyword
   ex: String s = new String("Hello world");

2. By using string literal or double quotes
   ex: String s = "Hello world";

STRING METHODS

1. Length(): It returns the count of the total characters present in the string.
2. compareTo(): It return 0 if two strings matches to each other, a positive number if first string is greater than other, a negative number if first string is lesser than second string.
3. concat():This methods concats/combines a specific string at the end of other string.
4. IsEmpty():It returns true if the string is empty otherwise false.

STRING METHODS

5. contains():It returs true if a specific string is found as a sub string in the given string.
6. toUpperCase():Converts each character in the string to uppercase.
7. toLowerCase(): Converts each character in the string to lowercase.
8. charAt():It returns character at i’th position in the given string

KAPLAN

Introduction to Programming using Java

Topic 03
Control Structures Part I

Learning Outcomes

- What is an control statement and explain different types of control statements present in java?
- Explain in breifly ablout IF,NESTED IF statements with examples?
- Explain about switch statement with an example?
- What is the differece between IF ELSE IF and switch statement in java?
- What is an ternary operator ans where it can be used?

KAPLAN

CONTROL STATEMENT

Control statements are the statements that define the flow of programs.

There are 3 types of control statements in Java:

- Selection
- Iteration
- Jump

SELECTION

A selection (also called a decision) is also one of the basic logic structures in computer programming.

In a selection structure, a question is asked, and depending on the answer, the program takes one of two courses of action, after which the program moves on to the next event.

BOOLEAN DATA TYPE

The boolean data type has two possible values, either true or false.

Default value: false.They are usually used for true/false conditions.

Example:

```
 public static void main(String[] args) {
        // TODO code application logic here
        boolean temp1=true;
        System.out.println(temp1);
        }
```

Output:

true

IF

It is used to decide whether a particular block of code needs to be executed or not i.e if a given condition is true then a block of statement is executed otherwise not.

Syntax:

```
if(condition)
{
  //block of code to be executed
}
```

IF

Example:

```
public static void main(String[] args) {
        // TODO code application logic here
        int x=15;
        if(x>10)
        System.out.println("x is greater than 10");
        System.out.println("I am outside if block");
}
```

Output:

X is greater than 10
I am outside if block

IF ELSE

The if-else statement has two blocks to codes, depending on the value of condition.
If the condition is true, then if block executes otherwise else block executes.

Note: else statement can only be used with if.

Syntax

```
if (condition)
{
//block of code
}
else
{
//block of code
}
```

IF ELSE

Example:

```
public static void main(String[] args) {
        // TODO code application logic here
        int x=5;
        if(x>10)
        System.out.println("x is greater than 10");
        else
        System.out.println("x is less than 10");
        System.out.println("I am outside if block");
}
```

Output:

X is less than 10
I am outside if block

NESTED IF

Nested, as the name suggests, something that is inside another.
In Java also we are provided with the option of Nested if-else, where Nested if statement is an if statement inside another.

Syntax

```
if(condition)
{
        //code 1
    if(condition)
{
        //code 2
    }
}
```

NESTED IF

Example:

```
public static void main(String[] args) {
        int x=50;
        if(x>10) {
                If (x>100) {
                        System.out.println("x is greater than 10 and 100");
                }
                else{
                        System.out.println("x is greater than 10 but less than 100");
                }
        }
        else
        System.out.println("x is less than 10");

}
```

Output:

X is greater than 10 but less than 100

IF ELSE IF

if-else-if statement is used to check multiple conditions.
In this statement we have only one "if" and one "else", however we can have multiple "else if". It is also known as if else if ladder

Syntax

```
if(Condition 1){
// Body of if statement
}
else if(Condition 2){
//Body of else if
}
else{
// Body of else statement
}
```

SWTICH STATEMENT

A switch statement allows you to test the value of an expression and, depending on that value, to jump directly to some location within the switch statement.

Only expressions of certain types can be used.

The value of the expression can be one of the primitive integer types int,short, orbyte.

It can be the primitive char type.

The positions that you can jump to are marked with case labels that take the form:
“case〈constant〉:”.

SWITCH STATEMENT

Syntax

```
switch (〈expression〉)
{
case〈constant-1〉:〈statements-1〉
break;
case〈constant-2〉:〈statements-2〉
break;.. //
(more cases)
case〈constant-N〉:〈statements-N〉
break;
default: // optional default case〈statements-(N+1)〉
}
```

TERNARY OPERATOR (?:)

This ternary operator or conditional operator can be used in if / else conditional statements , when the result is to be returned based upon one conditions.

Syntax:
Conditional expression ? expression1 : expression2 ;
If the conditional expression is true , return expression1 else return expression2 if its false.

TERNARY OPERATOR (?:)

```
public static void main(String[] args)
   {
      int a = 20, b = 30, min;
      min = (a < b) ? a : b;
      System.out.println("Minimum is = " + min);
   }
```

Output:
Minimum is =20

KAPLAN

Introduction to Programming using Java

Topic 04
Control Structures Part II

Learning Outcomes

- What is an repetition statement and discuss about loop design strategies?
- Explain in briefly about While,Do-while and for loops with examples?
- Explain difference between counter controlled loop and sentinal value loop?
- With examples explain when to use which loop?
- Explain about break and continue keywords with examples?

KAPLAN

LOOP DESIGN STRATEGIES

Repetition:

A repetition statement is used to execute a group or block of programming instructions multiple times.

- pre-condition
- post-condition
- loop variant
- loop invariant

SENTINEL-CONTROLLED LOOP

Sentinel-controlled loop is sometimes called indefinite repetition because it is not known in advance how many times the loop will be executed.

It is a repetition procedure for solving a problem by using a sentinel value (also called a signal value, a dummy value or a flag value) to indicate "end of data entry".

The sentinel value itself is not a part of the processed data.

SENTINEL-CONTROLLED LOOP

Read and display the content of a file.

```
String line;
BufferedReader br = new BufferedReader(...);
while (line = br.readLine() != null)
   System.out.println(line);
```

COUNT-CONTROLLED LOOP

Count-controlled loop requires;

- control variable (or loop counter)
- initial value of the control variable
- increment (or decrement) by which the control variable is modified each iteration through the loop
- condition that tests for the final value of the control variable

COUNT-CONTROLLED LOOP

Display the character and its' ASCII value for all lower case characters.

```
for (char x = 'a'; x <= 'z'; x++)
   System.out.println(x + " = " + (int) x);
```

Output:

Displays number from 66 to 92 because value of ‘a ‘is 66.

WHILE LOOP

It repeatedly executes the block code , or keep on performing the given task , provided the condition remains “TRUE”. Meaning, as soon as the condition becomes “FALSE” it stops executing.

Syntax:

```
while ( expression )
 {
 //body of loop statements;
}
```

WHILE LOOP

Write table of 2

```
int count =1;
int product ;
System.out.println(“ Table of 2 “ );
while (count <=10 )
{
 product  = 2 * count ;
System.out.println(“ 2 X  “ + count + “=  “ + product );
count ++;
}
```

DO-WHILE LOOP

it executes only when the condition is true , but what if you want to execute the block code at least once , even if the condition is not met!

In such cases we use do — while loop , it's almost the same as the while loop , the only difference is that the condition is written in the bottom , and the loop executes at least once no matter what!!

In this case too , it stops to execute if the condition becomes “false”.

Syntax:

```
do {
//body of loop statements
} while (  condition ) ;
```

DO-WHILE LOOP

```
int val = 0;
int val1;
do
{
val1 = val +1;
System.out.println( " Value Is " + val1 );
} while ( val > 0 );
```

Output :

Value Is 1

FOR LOOP

Syntax

```
for ( initialization ; termination ; increment )
{
 // body statements ;
}
```

(i) initialization : expression / condition to start the loop . It executes **ONLY ONCE** at the beginning of the loop.

(ii) termination : expression / condition to end/terminate the loop. This is evaluated every time the loop has to iterate , and if the expression evaluates to be "false" the loop terminates.

(iii) Increment : invoked after each iteration , gives the value by which the expression has to be incremented after each loop. Is optional , loop will run even if its not mentioned .

FOR LOOP

```
Int [ ]  num = { 10,20,30,40,50 } ;

System.out.print(“ The numbers are :  “);

for ( int i = 0 ; i < num.length ; i++)

{

System.out.print(num[i] + “  “ );

}
```

Output:

10 20 30 40 50

NESTED LOOPS

If a loop exists inside the body of another loop, it's called a nested loop.

```
public static void main(String[] args) {

    for (int i = 1; i <= 5; ++i) {

      System.out.println("Outer loop iteration " + i);

      for (int j = 1; j <=2; ++j) {
        System.out.println("i = " + i + "; j = " + j);
      }
    }
  }
```

NESTED LOOPS

Output:

```
Outer loop iteration 1
        i = 1 j = 1
        i = 1 j = 2
Outer loop iteration 2
        i = 1 j = 1
        i = 2 j = 2
Outer loop iteration 3
        i = 3 j = 1
        i = 3 j = 2
Outer loop iteration 4
        i = 4 j = 1
        i = 4 j = 2
Outer loop iteration 5
        i = 5 j = 1
        i = 5 j = 2
```

JUMP STATEMENTS

The break statement

As suggested by the name , break statements are used to terminate the execution of the code below and move out of the block code.

1. Unlabelled break
2. Labelled break

UNLABELLED BREAK

```
int[] num ={10,20,30,40,50};
int search = 40;
boolean foundIt = false;
for (int i=0 ; i < num.length ; i++)
{
if (num[i] ==search) {
foundIt = true;
break ;
}
}
If(foundIt) {
 System.out.println( search + " at index " + i);
}
else {
System.out.println( search + " not in the array ");
}
```

OUTPUT:40 at index 3

LABELLED BREAK

(From Left to Right)

```
int search = 122;
int i=0 , j=0;
boolean foundIt = false;
int[][] num={ {
33,64,31,15},{122,45,6,12},{165,123,10,20}
};
search: //label is followed by the " : "
for (; i<num.length;i++){
for ( j=0 ;j<num[i].length ; j++) {
if (num[i][j] == search) {
 foundIt = true;
break search ;
}   }}
if (foundIt){
System.out.println(search + "at " +i
+ " , " +j );
}
else{
 System.out.println( search +" not
found" );
}
```

OUTPUT:122 at 1,0

THE CONTINUE STATEMENT

The continue statement

It is used to skip the current iteration of the for , while or do-while loop. What it actually does is , it also terminates the loop thus preventing it from executing the statements below it , but, the only difference is that , the flow of control remains with the loop ,i.e., the loop will run again from the top statement of its block code.

1. Unlabelled Continue
2. Labelled Continue

UNLABELLED CONTINUE

```
int[] num ={ 12,2,3,4,52,100,7,1,20,21};
int count =0;
for ( int i=0 ; i < num.length ; i++)
{
if ( (num[i] % 2) != 0 )
continue;
Count++;
}
System.out.println("No of even numbers in the array are " + count };
```

OUTPUT

No of even numbers in the array are 6

LABELLED CONTINUE

(From Left to Right)

```
String main="look for the substring in me ";
String sb = "sub";
boolean found = false;
int max = main.length() - sb.length() ;
test:for (int i=0 ; i<=max;i++)
{
int n = sb.length();
 int j=i ;
int k=0;
while ( n --!=0 )
{
if (main.charAt(j++) != sb.charAt(k++))
continue test;
}
found = true;
break true;
}
System.out.println(found ? "found it " : "not found" );
```

OUTPUT:

found it

KAPLAN

Introduction to Programming using Java

Topic 05
Classes and objects Part I

Learning Outcomes

- What are classes and objects?
- What is an static class and explain it with an example?
- Explain about builtin classes with example?
- Explain about this reference with example?
- How to pass objects in java?

KAPLAN

CLASS

A class is a blueprint or template for the object. It represents a group of objects which have common properties. A class in Java can contain:

- Fields
- Methods
- Blocks
- Constructors
- Nested class and interface

Syntax:

```
class class_name {
   // fields;
   // methods;
}
```

OBJECT

Object is an entity that has state and behavior. In the real world, we can find many objects around us like cars, books, animals, etc, and all these objects have a state and behavior.

An object has three properties:

• State	It represents the data of an object.
• Behaviour	It represents the functionality of an object.
• Identity	An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user but it is used internally by the jVM to identify each object uniquely.

INITIALIZING OBJECTS

Initializing an object means storing data into the object.
There are 3 ways to initialize an object in Java.

1. By reference variable
2. By method
3. By constructor

KAPLAN

INITIALIZING OBJECTS

```
class Car {
    String Color;
    // Initialize by constructor
    Car() {
        Color = "Red";
    }

    void ChangeColor(String NewColor) {
        Color = NewColor;
    }
}
```

KAPLAN

INITIALIZING OBJECTS

```
class Test {
   public static void main(String[] args) {
     //Initialize br constructor
       Car Ford = new Car();
       System.out.println(Ford.Color);

       // Initialize by reference variable
       Ford.Color = "Blue";
       System.out.println(Ford.Color);

       //Initialize by method
       Ford.ChangeColor("Green");
       System.out.println(Ford.Color);
   }
}
```

INITIALIZING OBJECTS

Output:

- Red
- Blue
- Green

STATIC CLASS

In java, we cannot use the static keyword with a class unless it is an inner class. A static class that is created inside a class is called a static nested class in java.

Properties:

- A static nested class can be instantiated without instantiating its outer class.
- A static nested class can access static data members of the outer class but it cannot access non-static members.

KAPLAN

STATIC CLASS

```
class OuterClass {
   static private String Color = "Red";
   static class InnerClass {
      void msg() {
         // Accessing static member of outer class
         System.out.println("Color is " +Color);
      }
   }
}

class MainClass {
   public static void main(String[] args) {
      // Instantiating inner class with out instantiating outer class
      OuterClass.InnerClass Obj = new OuterClass.InnerClass();
      Obj.msg();
   }
}
```

Output :
Color is Red

KAPLAN

BUILTIN CLASSES

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

KAPLAN

BUILTIN CLASSES

Example:

```
import java.util.Arrays;
public class SortArray {
   public static void main(String[] args) {
      int[] arr = {13, 7, 6, 45, 21, 9, 101, 102};
      Arrays.sort(arr);
      System.out.printf("Sorted array : %s", Arrays.toString(arr));
   }
}
```

In the above example, we have imported the ‘Arrays’ class from java.util and then sorted an array of elements using the ‘sort’ method.

Output:

6 7 9 13 21 45 101 102

KAPLAN

PASSING OBJECTS

In Java, it is always ‘pass by value’. It means whenever we send arguments to a method, values assigned to these arguments are passed and any changes to these values don’t reflect the original arguments.

In general, objects are reference variables that store the location of the created object. When we pass an object as an argument to methods, java follows the same procedure of pass by value, which creates another variable that stores the value of the argument which is the location of the passed object itself. Thus any changes that are implemented on the properties of the object inside the method will change the original object.

KAPLAN

PASSING OBJECTS

```
class Rectangle {
   int Length, Width;

   Rectangle(int l, int w) {
      Length = l;
      Width = w;
   }

   void ChangeValues(Rectangle R, int l, int w) {
      R.Length = l;
      R.Width = w;
   }
   void PrintArea(Rectangle R) {
      int Area = R.Length * R.Width;
      System.out.println("Area of rectangle: " +Area);
   }
}
```

KAPLAN

PASSING OBJECTS

```
class Test {
    public static void main(String[] args) {
        Rectangle R = new Rectangle(3, 4);

        //passing object to method
        R.ChangeValues(R, 4, 5);
        R.PrintArea(R);
    }
}
```

Output:
Area of rectangle is 20

THIS REFERENCE

The ‘this’ keyword refers to the current object in a method or constructor. The ‘this’ keyword is frequently used to differentiate between class attributes and parameters with the same name.

‘This’ keyword is used for

- Referencing current class instance variables.
- Invoking the current class constructor or method.
- Returning the current class instance.
- Passing as an argument to the constructor or method.

KAPLAN

Introduction to Programming using Java

Topic 06
Classes and objects Part II

Learning Outcomes

1. What are methods?
2. How to declare and call methods?
3. Explain the difference between void and value returning methods?
4. Explain about different types of constructors with example?
5. Explain about overloaded methods and constructors?

KAPLAN

METHODS

Java Method is a block of statements in or commands which are used to run a particular task and return the result or value to the caller, it can also perform a task without returning any value.

They also provide the user to use a particular code again without retyping.

A Java method must always be a part of a class in Java.

KAPLAN

METHODS

1. Access Specifier

An access specifier in Java defines the access type of the method, i.e. it defines in which part of the program it can be accessed.

1. Public	In this type, the method is accessible in all classes in the application.
2. Protected	Here, the method is accessible in the classes in which it is defined and also in its subclasses.
3. Private	Method is accessible only in the class it is defined in.
4. Default	Method is accessible only within the same class and package in which it is defined.

KAPLAN

METHODS

2. Return Type

The return type defines the type of data that is returned by the method. It can return any kind of value like integer, float, double, etc and also it returns void when nothing is returned by the method.

3. Method Name

A method name is used to define the name of a method, it is the same as the naming of field types with a little difference in the convention. A method is invoked by its name.

KAPLAN

METHODS

4. Parameter list

The parameter list is a list with input parameters that are separated by commas, they are preceded by their data types within closed parenthesis, and also, if there are no parameters then there is an empty parenthesis ().

5. Method Body

The body of the Java method must be enclosed within braces, all the codes for the method are written in it. All the operations are performed inside a Java method.

KAPLAN

METHODS

6 Method Signature

A method signature consists of the method name and parameter list in Java i.e. number, type, and order of parameters.

Example:

```
public class BasicMethod {
static void staticmethod() {
System.out.println("This is static method. ");
}
public void nonstaticmethod() {
System.out.println("This is  non-static method. ");
}
```

METHODS

```
public static void main(String[] args) {
staticmethod();
BasicMethod ob = new BasicMethod();
ob.nonstaticmethod();
}
}
```

Output:

This is static method

This is non static method

METHODS

Calling a method

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**;**

Example

```
public class MyClass {
    static void myMethod() {
        System.out.println("First Method");
    }
    public static void main(String[] args) {
        myMethod();
    }
}
```

VOID METHODS

If the return type is void then that will not return any value to the calling function

Example

```
public class MethodCall1 {
  public static void main(String[] args){
            int x = 1,y=2;
            addNumbers(x, y);
  }
  public static void addNumbers(int a, int b)
  {
            int res = a + b;
            System.out.println("value is"+res);
  }}
```

Output:

Value is 3

VALUE RETURNING METHODS

```
public class MethodCall1 {
  public static void main(String[] args){
        int x = 5, y = 2,z = 4;
        addNumbers(x, y);
        int res = addNumbers(y, z);
        System.out.print("The result of y + z is: " + res);
  }
  public static int addNumbers(int a, int b)
  {
        int res = a + b;
        return res;
  }}
```

Output:

The result of y + z is 11

CONSTRUCTORS

A constructor is a special type of method in Java which is used for giving initial values to an object.

1. Every constructor is of the same name as that of its parent class
2. A constructor gets loaded after class loading
3. If we do not define a constructor for a class, then Java includes its own default constructor in the code.

There are two types of constructor;

1. Default constructor
2. Parameterized Constructor

DEFAULT CONSTRUCTORS

A constructor that has no parameter is known as default constructor.

Example

```
public class Student {
        // default constructor
        String name
        public Student()
{
                this.name=”abc”
                }
}
```

KAPLAN

DEFAULT CONSTRUCTORS

```
class demo{
        public static void main(String[] args){
        Student s = new Student();
        System.out.println(s.name);
        }
}
```

Output:

abc

KAPLAN

PARAMETERIZED CONSTRUCTORS

A constructor that has parameters is known as a parameterized constructor. If we want to initialize fields of the class with our own values, then we use a parameterized constructor.

Example

```
public class Student {
        // default constructor
        String name
        public Student(String temp) {
                this.name=temp;
                }
        }
```

PARAMETERIZED CONSTRUCTORS

```
class demo{
        public static void main(String[] args)
        {
        Student s = new Student(def);
        System.out.println(s.name);
        }
}
```

Output:

```
def
```

METHOD ABSTRACTION & STEPWISE REFINEMENT

Method abstraction is achieved by separating the use of a method from its implementation.

The user can use a method without knowing how it is implemented. The details of the implementation are encapsulater in the method and hidden from the user who invokes the method. This is also known as information hiding or encapsulation.

If we decide to change the implementation, the user program will not be affected, provided that you do not change the method signature.

We have already used the System.out.print method to display a string and the max method to find the maximum number. We know how to write the code to invoke these methods in your program, but as a user of these methods, you are not required to know how they are implemented.

KAPLAN

METHOD ABSTRACTION & STEPWISE REFINEMENT

The concept of method abstraction can be applied to the process of developing programs.

When writing a large program, you can use the divide-and-conquer strategy, also known as stepwise refinement , to decompose it into subproblems.

The subproblems can be further decomposed into smaller, more manageable problems.

KAPLAN

OVERLOADED METHODS

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

KAPLAN

CHANGING NUMBER OF ARGUMENTS

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Output:

22
33

KAPLAN

CHANGING DATA TYPE OF ARGUMENTS

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Output:

22
24.9

CONSTRUCTOR OVERLOADING

Making multiple constructors for a single class, each having a different prototype is called Constructor overloading.This allows you to initialize the data members in multiple ways. Thus you can have more than 1 kind of object of the same class by passing different (number or datatype) of arguments during creation of the object.

Example

```
class area{
  //Data members-
   int side;
   int length;
   int breadth;
 public area(int s)// 1st constructor
  {
  side=s;
  }
```

CONSTRUCTOR OVERLOADING

```
public area(int l,int b)// 2nd constructor
  {
   length=l;
   breadth=b;
  }
 int square_area()
  {
   return side*side;
   }
   int rectangle_area()
   {
   return length*breadth;
   }
}
```

CONSTRUCTOR OVERLOADING

```
class run
{
  public static void main(String args[])
   {
          area Square=new area(5); //object creation according to 1st constructor
          area Rectangle=new area(4,6);   // according to 2nd constructor
           System.out.println("Area of square="+Square.square_area());
          System.out.println("Area of rectangle="+Rectangle.rectangle_area());
          }
          }
```

Output:

Area of square=25

Area of rectangle=24

KAPLAN

Introduction to Programming using Java

Topic 07
Arrays

Learning Outcomes

1. What are arrays?
2. How to declare and access arrays?
3. How to search for an element in an array?
4. How to sort the elements in an array?
5. How to declare and access multidimensional arrays?
6. How to pass arrays to methods?

KAPLAN

ARRAYS

Array is an object which contains elements of a similar data type.

The elements of an array are stored in a contiguous memory location.

Array is an index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Example

Instead of writing int a=1, int b=2, int c= 3 we can write int arr[]={1,2,3}.

HOW TO DECLARE/ACCESS ARRAYS

datatype[] array;

or

datatype array[];

Creating new array

int array[]= new int[size];

To insert values in an array we must define its size. Arrays include values which are predefined or user defined. A program generally uses loop to take user defined values.

HOW TO DECLARE/ACCESS ARRAYS

PREDEFINED

```
int array[]={1,2,3,4,5}
```

USER DEFINED

```
int array[]=new int[5];

for(int i=0;i<5;i++)
{
array[i]=x; /*x is the user defined value*/
}
```

The size of an array can also be user defined.

```
int array[]=new int [n]; /*n is the user defined value*/
```

HOW TO DECLARE/ACCESS ARRAYS

We can access the element of an array using the index number. Here is the syntax for accessing elements of an array,

// access array elements
array[index]

Example:

```
class Main {
 public static void main(String[] args) {

   // create an array
   int[] num = {1, 4, 15,12, 50};

   // access each array elements
   System.out.println("Accessing Elements of Array:");
```

HOW TO DECLARE/ACCESS ARRAYS

```
System.out.println("First Element: " + num[0]);
   System.out.println("Second Element: " + num[1]);
   System.out.println("Third Element: " + num[2]);
   System.out.println("Fourth Element: " + num[3]);
   System.out.println("Fifth Element: " + age[4]);
 }
}
```

Output

```
Accessing Elements of Array:
First Element: 1
Second Element: 4
Third Element: 15
Fourth Element: 12
Fifth Element: 50
```

VARIABLE LENGTH ARGUMENTS

Variable-length argument lists, makes it possible to write a method that accepts any number of arguments when it is called.

Example

```
public class VarArgs
{
   public static void main(String[] args)
   {
      System.out.println("The sum of 1 and 2 is " + sum(1, 2));
      System.out.println("The sum of 1, 2, and 3 is " + sum(1, 2, 3));
      System.out.println("The sum when no parameter is passed is "+ sum());
   }
```

VARIABLE LENGTH ARGUMENTS

```
public static int sum(int... list)
  {
    int total = 0;
    // add all the values in list array
    for (int i = 0; i < list.length; i++)
    {
      total += list[i];
    }
    return total;
  }
}
```

Output :

The sum of 1 and 2 is 3

The sum of 1, 2, and 3 is 6

The sum when no parameter is passed is 0

SEARCHING

Algorithm:

- Step 1: Traverse the array
- Step 2: Match the key element with array element
- Step 3: If key element is found, return the index position of the array element
- Step 4: If key element is not found, return -1

Example:

```
import java.util.Scanner;
class LinearSearch
{
  public static void main(String args[])
  {
    int i, n, search, array[];
    Scanner in = new Scanner(System.in);
```

SEARCHING

```
System.out.println("Enter number of elements");
   n = in.nextInt();
   array = new int[n];
   System.out.println("Enter  " + n + " elements");
   for (int i= 0; i < n; i++)
   array[i] = in.nextInt();
   System.out.println("Enter value to search");
   search = in.nextInt();
   for (i = 0; i< n; i++)
   {
     if (array[i] == search)     /* Searching element is present */
     {
        System.out.println(search + " is present at location " + (i + 1)
);
         break;
     }
   }
```

SEARCHING

```
if (i == n)  /* Element to search isn't present */
     System.out.println(search + " isn't present in
array.");
  }
}
```

Output:

```
Enter number of elements
5
Enter 5 elements
10 11 12 13 14
Enter value to search
13
13 is present at location 4
```

SORTING

Algorithm

- STEP 1: START
- STEP 2: INITIALIZE array[]
- STEP 3: SET temp =0
- STEP 7: REPEAT STEP 8 to STEP 9 UNTIL i<array.length
  //for(i=0; i<array.length; i++)
- STEP 8: REPEAT STEP 9 UNTIL j<array.length
  //for(j=i+1;j<array.length;j++)
- STEP 9: if(array[i]>array[j]) then
  temp = array[i]
  array[i]=array[j]
  array[j]=temp
- STEP 11: PRINT "Elements of array sorted in ascending order"
- STEP 12: REPEAT STEP 13 UNTIL i<arr.length
  //for(i=0;i<arr.length;i++)
- STEP 13: PRINT arr[i]
- STEP 14: END

SORTING

```
public class SortAsc
{
   public static void main(String[] args)
 {
      //Initialize array
      int [] array = new int [] {15, 21, 80, 72, 1};
      int temp = 0;
      System.out.println("Elements of original array: ");
      for (int i = 0; i < array.length; i++)
     {
         System.out.print(array[i] + " ");
      }
```

SORTING

```
//Sort the array in ascending order
    for (int i = 0; i < array.length; i++) {
        for (int j = i+1; j < array.length; j++) {
            if(arr[i] > array[j]) {
                temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            } } }
    //Displaying elements of array after sorting
    System.out.println("Elements of array sorted in
ascending order: ");
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }   }  }
```

SORTING

Output:

Elements of original array:

15 21 80 72 1

Elements of array sorted in ascending order:

1 15 21 72 80

MULTI-DIMENSIONAL ARRAYS

The Java multidimensional arrays are arranged as an array of arrays i.e. each element of a multi-dimensional array is another array. The representation of the elements is in rows and columns. Thus, you can get a total number of elements in a multidimensional array by multiplying row size with column size.

So if you have a two-dimensional array of 2×3, then the total number of elements in this array = 2×3 = 5

We create a multidimensional array in Java by appending one set of square brackets ([]) per dimension you want to add.

Here is an example that creates a two-dimensional array:
int[][] intArray = new int[10][20];

HOW TO DECLAR/ACCESS MULTI-DIMENSIONAL ARRAYS

How to declare 2D array

The declaration of a 2D array is similar to that of a one dimensional array.

datatype[max_row][max_column] array;

or

datatype array[max_row][max_column];

Creating new array

int array=new int[max_row][max_column];

Similar to one dimensional array where values are either pre-defined or user defined.
A program uses a loop to take user defined values.

PRE DEFINED

int arr[2][2]={1,2,3,4};

HOW TO DECLARE/ACCESS MULTI-DIMENSIONAL ARRAYS

USER DEFINED

```
int arr [][]=new int[size][size]; /* size is user defined*/
for(int i=0;i<size;i++)
{
for(int j=0;j<size;j++)
{
arr[i][j]=x; /*where x is user defined*/
}
}
```

Each and every element in the 2D array has a set of corresponding indexes , which is used to access the array elements. The indexing of array elements is done as 0,1,2……n-1 for i and similarly for j. The first element of an array is at index [0][0] thus it is called the zeroth element of an array. The final element has an index of [n-1][n-1].

HOW TO DECLARE/ACCESS MULTI DIMENSIONAL ARRAYS

EXAMPLE

Print elements of matrix

```
int[][] arr = { { 1, 2 }, { 3, 4 } };
for (int i=0;i<2;i++)
{
    for (int j=0;j<2;j++)
    {
    System.out.print(arr[i][j]+" ");
    }
System.out.println();
}
```

Output:

```
1 2
3 4
```

HOW TO PASS MULTI-DIMENSIONAL ARRAYS TO METHODS

Transpose of a matrix

```
public class Transpose
{
        public static void main(String[] args)
        {
        int[] [] table= {{1,2,3},{4,5,6}};
            int[] [] result;
        System.out.println("Matrix Before Transpose : ");
        for(int i=0;i<table.length;i++)
        {
                for(int j=0;j<table[i].length;j++)
                System.out.print(table[i][j]+" ");
        System.out.println();
        }
```

HOW TO PASS MULTI-DIMENSIONAL ARRAYS TO METHODS

```
result=transpose(table);
        System.out.println("\nMatrix After Transpose :");
         for(int i=0;i<result.length;i++)
         {
                for(int j=0;j<result[i].length;j++)
                System.out.print(result[i][j]+" ");
                System.out.println();
         }
         }
        static int[][] transpose(int[][] a)
         {

                int[][] temp=new int[a[0].length][a.length];
```

HOW TO PASS MULTI-DIMENSIONAL ARRAYS TO METHODS

```
for(int i=0;i<a[0].length;i++)
        {
        for(int j=0;j<a.length;j++)
                temp[i][j]= a[j][i];
        }
          return temp;
    } }
```

Output:

Matrix before transpose
1 2 3
4 5 6
Matrix after transpose
1 4
2 5
3 6

KAPLAN

Introduction to Programming using Java

Topic 08
Inheritance and Polymorphism

Learning Outcomes

1. What is inheritance and different types of inheritance ?
2. Explain about super keyword ?
3. What is polymorphism and different types of polymorphism?
4. What is method overriding?
5. Explain about object equals method?

KAPLAN

INHERITANCE

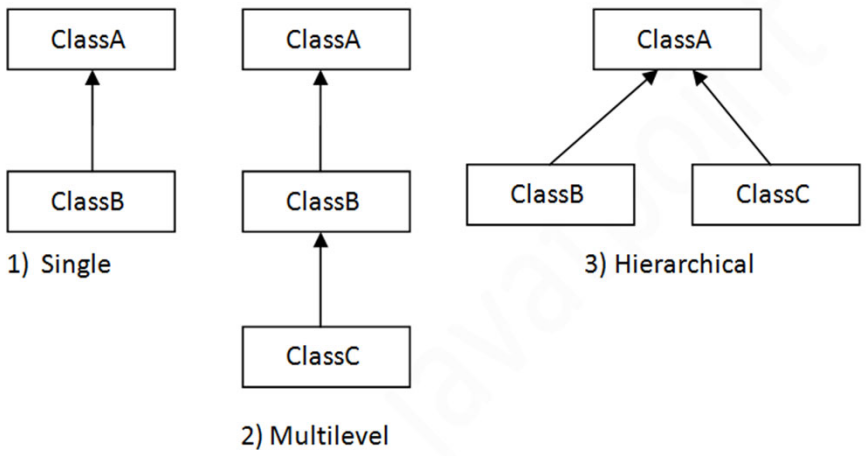
Inheritance is an integral part of Java OOPs which lets the properties of one class be inherited by the other. It basically helps in reusing the code and establish a relationship between different classes.

1. Parent class (Super or Base class)

2. Child class (Subclass or Derived class)

A class that inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

TYPES OF INHERITANCE



SINGLE INHERITANCE

```
class A{
    int a = 10;
    void show() {
        System.out.println("a = "+a);
    }
}
public class B extends A{
    public static void main(String[] args) {
    B b = new B();
    b.show();

    }
}
```

Output:

a=10

MULTILEVEL INHERITANCE

```
class A{
    int a = 10;
    void show() {
        System.out.println("a = "+a);
    }
}

class B extends A{
    int b = 10;
    void showB() {
        System.out.println("b = "+b);
    }
}
```

MULTILEVEL INHERITANCE

```
public class C extends B{
public static void main(String[] args) {
    C c = new C();
    c.show();
    c.showB();
    }
}
```

Output:

a=10 b=10

HIERARCHICAL INHERITANCE

```
class A{
    int a = 10;
    void show() {
        System.out.println("a = "+a);
    }
}
class B extends A{
    int b = 10;
    void showB() {
        System.out.println("b = "+b);
    }
}
public class C extends A{
    public static void main(String[]
args) {
        C c = new C();
        c.show();
        B b = new B();
        b.show();
    }
}
```

Output:

a = 10 a = 10

SUPER KEYWORD

In Java, super keyword is used to refer to the immediate parent class of a child class.
In this example we will only focus on accessing the parent class property or variables.

```
class Parent
{
   String name;

}
public class Child extends Parent {
   String name;
   public void details()
   {
      super.name = "Parent";  //refers to parent class member
      name = "Child";
      System.out.println(super.name+" and "+name);
   }
```

SUPER KEYWORD

```
public static void main(String[] args)
   {
      Child cobj = new Child();
      cobj.details();
   }
}
```

Output:

Parent and Child

SUPER KEYWORD

In this example we will only focus on accessing the parent class methods.

```
class Parent
{
   String name;
   public void details()
   {
     name = "Parent";
       System.out.println(name);
   }
}
public class Child extends Parent {
   String name;
   public void details()
   {
```

SUPER KEYWORD

```
 super.details();//calling Parent class details() method
       name = "Child";
       System.out.println(name);
   }
   public static void main(String[] args)
   {
       Child cobj = new Child();
       cobj.details();
   }
}
```

Output:

Parent Child

METHOD OVERRIDING

Declaring a method in subclass which is already present in parent class is known as method overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class.

Example

```
class Human{
  //Overridden method
  public void eat()
  {
    System.out.println("Human is eating");
  }
}
```

METHOD OVERRIDING

```
class Boy extends Human{
  //Overriding method
  public void eat(){
    System.out.println("Boy is eating");
  }
  public static void main( String args[]) {
    Boy obj = new Boy();
    //This will call the child class version of eat()
    obj.eat();
  }
}
```

Output:

Boy is eating

POLYMORPHISM

Polymorphism contains two words : Poly and Morphism . Poly means many and morphism means behaviour. The ability of an object to take on different forms is known as polymorphism.

Polymorphism is the capability of a method to do different things based on the object that it is acting upon.

There are two types of polymorphism in java:

1) Static Polymorphism also known as compile time polymorphism.

2) Dynamic Polymorphism also known as runtime polymorphism.

STATIC POLYMORPHISM

Polymorphism that is resolved during compiler time is known as static polymorphism.
Method overloading is an example of compile time polymorphism.

```
class SimpleCalculator
{
   int add(int a, int b)
   {
       return a+b;
   }
   int  add(int a, int b, int c)
   {
       return a+b+c;
   }
}
```

STATIC POLYMORPHISM

```
public class Demo
{
   public static void main(String args[])
   {
            SimpleCalculator obj = new SimpleCalculator();
      System.out.println(obj.add(10, 20));
      System.out.println(obj.add(10, 20, 30));
   }
}
```

Output:

30
60

DYNAMIC POLYMORPHISM

It is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime, that's why it is called runtime polymorphism.

Example

```
class ABC{
  public void myMethod(){
        System.out.println("Overridden Method");
  }
}
public class XYZ extends ABC{

  public void myMethod(){
        System.out.println("Overriding Method");
  }
```

DYNAMIC POLYMORPHISM

```
  public static void main(String args[]){
        ABC obj = new XYZ();
        obj.myMethod();
  }
}
```

Output:

Overriding Method

KAPLAN

OBJECT EQUALS METHOD

Java Object class is the super class of all the Java classes. All Java classes implements the Object class by default.

The equals() method of the Object class compares the equality of two objects. The two objects will be equal if they share the same memory address.

Syntax:
public boolean equals(Object obj)
The method parses a reference object as a parameter. It returns true if the objects are equal, else returns false.

It is also possible that an object is equal to another given object, then the equals() method follow the equivalence relation to compare the objects.

KAPLAN

OBJECT EQUALS METHOD

- Reflexive: If x is a non-null reference, the calling of x.equals(x) must return true.
- Symmetric: If the two non-null references are x and y, x.equals(y) will return true if and only if y.equals(x) return true.
- Transitive: If the three non-null references are x, y, and z, x.equals(z) will also return true if x.equals(y) and y.equals(z) both returns true.
- Consistent: If the two non-null references are x and y, the multiple calling of x.equals(y) constantly returns either true or false. It does not provide any information used in the comparison.
- For any non-null reference x, x.equals(null) returns false.

In short, for any non-null reference say x and y, it returns true if and only if both references refer to the same object.

OBJECT EQUALS METHOD

Example

```
public class ObjectComparisonExample
{
public static void main(String[] args)
{
//creating constructor of the Double class
Double x = new Double(123.45);
//creating constructor of the Long class
Long y = new Long(12345);
//invoking the equals() method
System.out.println("Objects are not equal, hence it returns " +
x.equals(y));
System.out.println("Objects are equal, hence it returns " +
```

OBJECT EQUALS METHOD

```
x.equals(123.45));
}
}
```

Output:

Objects are not equal, hence it returns false

Objects are equal, hence it returns true

KAPLAN

Introduction to Programming using Java

Topic 09
FILES

Learning Outcomes

1. What are files and why they are used in programing?
2. How to read and write to files?
3. What is stream and different types of streams available in java?

KAPLAN

FILES

Most Programs cannot accomplish their goals without accessing external data. This data is retrieved from an input source.

The results of a program are sent to an output destination. The generic notion of an input source can abstract many different kinds of input : from a disk file, a keyboard, or a network socket.

Likewise, an output destination may be a disk file, on the screen, or a network connection. These abstractions are a clean way to deal with input/output (I/O) without having difference between the keyboard and network.

CREATE A FILE

To create an object of File, we need to import the java.io.File package first. Once we import the package, here is how we can create objects of file.

```
// creates an object of File using the path
File file = new File(String pathName);
```

Here, we have created a file object named file. The object can be used to work with files and directories.
Note: In Java, creating a file object does not mean creating a file. Instead, a file object is an abstract representation of the file or directory pathname (specified in the parenthesis).

CREATE A FILE

```
// importing the File class
import java.io.File;
class Main {
public static void main(String[] args) {
// create a file object for the current location
File file = new File("newFile.txt");
try {
// trying to create a file based on the object
boolean value = file.createNewFile();
if (value) {
System.out.println("The new file is created.");
}
else {
System.out.println("The file already exists.");
}
```

CREATE A FILE

```
}
catch(Exception e) {
e.getStackTrace();
}}}
```

We have created a file object named file. The file object is linked with the specified file path.
File file = new File("newFile.txt");
Here, we have used the file object to create the new file with the specified path.
If newFile.txt doesn't exist in the current location, the file is created and this message is shown.
The new file is created.
However, if newFile.txt already exists, we will see this message.
The file already exists.

READ A FILE

```
// importing the FileReader class
import java.io.FileReader;
class Main {
public static void main(String[] args) {
char[] array = new char[100];
try {
// Creates a reader using the FileReader
FileReader input = new FileReader("input.txt");
// Reads characters
input.read(array);
System.out.println("Data in the file:");
System.out.println(array);
// Closes the reader
input.close();
}
```

READ A FILE

```
catch(Exception e) {
e.getStackTrace();
}
}
}
```

Output

Data in the file:
SAMPLE CONTENT

WRITE TO FILE

```
// importing the FileWriter class
import java.io.FileWriter;
class Main {
public static void main(String args[]) {
String data = "SAMPLE CONTENT";
try {
// Creates a Writer using FileWriter
FileWriter output = new FileWriter("output.txt");
// Writes string to the file
output.write(data);
System.out.println("Data is written to the file.");
// Closes the writer
output.close();
}
```

WRITE TO FILE

```
catch (Exception e) {
e.getStackTrace();
}
}
}
```

Output
Data is written to the file.
Output.txt
SAMPLE CONTENT

STREAM

Java uses the concept of a stream to make I/O operations on a file.

In Java, Stream is a sequence of data which can be of two types.

1. Byte Stream

This mainly incorporates byte data. When an input is provided and executed with byte data, then it is called the file handling process with a byte stream.

2. Character Stream

Character Stream is a stream which incorporates characters. Processing of input data with character is called the file handling process with a character stream.

BYTE STREAM

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

BYTE STREAM

```
finally {
        if (in != null) {
           in.close();
        }
        if (out != null) {
           out.close();
        }
     }
   }
}
```

Input.txt

Sample content.

Output.txt

Sample content

CHARACTER STREAM

```
import java.io.*;
public class CopyFile {
   public static void main(String args[]) throws IOException {
      FileReader in = null;
      FileWriter out = null;

      try {
         in = new FileReader("input.txt");
         out = new FileWriter("output.txt");
         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }
```

CHARACTER STREAM

```
finally {
        if (in != null) {
           in.close();
        }
        if (out != null) {
           out.close();
        }
     }
  }
}
```

Input.txt
Sample content
Output.txt
Sample content

KAPLAN

Introduction to Programming using Java

Topic 10
RECURSION

Learning Outcomes

1. What is recursion and why does it work?
2. Difference between iterative and recursive algorithms?
3. How to compute factorials using recursion ?
4. How to compute fibonacci sequence using recursion?

KAPLAN

RECURSION

Recursion is a problem solving technique which involves breaking a problem into smaller instances of the same problem (also called as subproblems) until we get small enough subproblem that has a trivial solution. We can say that recursion is “defining a problem in terms of itself” as it involves a function calling itself with a base case to terminate the infinite loop.
For example, we can define the operation "find your way home" as:

1. If you are at home, stop moving.
2. Take one step toward home.
3. "find your way home".

Here the solution to finding your way home is two steps (three steps). First, we don't go home if we are already home. Secondly, we do a very simple action that makes our situation simpler to solve. Finally, we redo the entire algorithm

WHY RECURSION WORKS

In a recursive algorithm, the computer "remembers" every previous state of the problem. This information is "held" by the computer on the "activation stack" (i.e., inside of each function's workspace).

Every function has its own workspace per call of the function.

A recursive algorithm always has two parts:

- **Base case**, which is the simplest, smallest instance of the problem, that can’t be decomposed any further. Base cases often correspond to emptiness – the empty string, the empty list, the empty set, the empty tree, zero, etc.
- **Recursive step**, which decomposes a larger instance of the problem into one or more simpler or smaller instances that can be solved by recursive calls, and then recombines the results of those subproblems to produce the solution to the original problem.

FACTORIAL USING LOOPS

Factorial Program in Java: Factorial of n is the product of all positive descending integers. Factorial of n is denoted by n!. For example:

1. 4! = 4*3*2*1 = 24
2. 5! = 5*4*3*2*1 = 120

```
class FactorialExample{
 public static void main(String args[]){
  int i,fact=1,number=5;//It is the number to calculate factorial
  for(i=1;i<=number;i++){
      fact=fact*i;
  }
  System.out.println("Factorial of "+number+" is: "+fact);
 }  }
```

Output:

Factorial of 5 is: 120

FACTORIAL USING RECURSION

```
public static long factorial(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n-1);
  }
}
public static void main(String[] args) {
    long x = factorial(4);
  System.out.println("Factorial of 4"+" is: "+x);
}
```

Output:

Factorial of 4 is: 24

FACTORIAL USING RECURSION

In the recursive implementation the base case is n = 0, where we compute and return the result immediately: 0! is defined to be 1. The recursive step is n > 0, where we compute the result with the help of a recursive call to obtain (n-1)!, then complete the computation by multiplying by n.

FIBONACCI

The Fibonacci sequence gets its name from Italian mathematician, Leonardo Fibonacci. He introduced this series in Western Europe with his book Liber Abaci back in 1202. The Indian mathematics scene had seen the Fibonacci magic way back in 200 BCE, as evidenced by the works of Pingala. This representation of numbers also occupies a special place in the spheres of coding and computing.

It is a sequence of non-negative integers whose nth term, i.e., F_n is given by sum of last 2 terms, i.e., $F_{n-1}+F_{n-2}$. Base cases for this recursive relation is $F_0 = 0$ and $F_1 = 1$.

FIBONACCI

The Fibonacci Sequence is the series of numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

The next number is found by adding up the two numbers before it:

- 2 is calculated by adding the two numbers preceding it (1+1),
- 3 is calculated by adding the two numbers preceding it (1+2),
- 5 is (2+3), and so on..

Here is an example of Fibonacci series: 0,1,1,2,3,5,8,13….etc.

FIBONACCI USING LOOPS

```
class FibonacciExample1{
public static void main(String args[])
{
 int n1=0,n2=1,n3,i,count=10;
 System.out.print(n1+" "+n2);//printing 0 and 1
 for(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are
already //printed
 {
  n3=n1+n2;
  System.out.print(" "+n3);
  n1=n2;
  n2=n3;
 } }}
```

Output:

0 1 1 2 3 5 8 13 21 34

Fibonacci using Recursion

```
public class FibonacciCalc{
   public static int fibonacciRecursion(int n){
   if(n == 0){
            return 0;
   }
   if(n == 1 || n == 2){
                        return 1;
            }
   return fibonacciRecursion(n-2) + fibonacciRecursion(n-1);
   }
public static void main(String args[]) {
   int maxNumber = 10;
   System.out.print("Fibonacci Series of "+maxNumber+" numbers: ");
   for(int i = 0; i < maxNumber; i++){
                        System.out.print(fibonacciRecursion(i) +" ");
            }
```

FIBONACCI USING RECURSION

}}

Output:

Fibonacci Series of 10 numbers: 0 1 1 2 3 5 8 13 21 34

1. Take an input number. Checks for 0, 1, 2 and returns 0, 1, 1 accordingly because the Fibonacci sequence starts with 0, 1, 1.
2. When input n is >=3, The function will call itself recursively. The call is done two times. Let's see an example for input of 4.

fibonacciRecursion (4)

It will recursively call fibonacciRecursion function for values 2 and 3

fibonacciRecursion (2) \\ call for value 0 and 1

fibonacciRecursion (0) = 0

fibonacciRecursion (1) = 1

fibonacciRecursion (3) \\ It will call for 1 and 2

fibonacciRecursion (1) = 1

FIBONACCI USING RECURSION

fibonacciRecursion (2) \\ It will call for 0 and 1

fibonacciRecursion (0) = 0

fibonacciRecursion (1) = 1

Now result is added 0+1+1+0+1=3