CSC 233: Object Oriented Programming using JAVA
(2 Credits)

Published by the Centre for Open and Distance Learning,
University of Ilorin, Nigeria

E-mail: codl@unilorin.edu.ng
Website: https://codl.unilorin.edu.ng

Course Development Team

Content Authoring

Dr. AbdulraheemMuyideen

Language Editor

Bankole Ogechi Ijeoma

Instructional Design

Koledafe Sunday Olawale
Jibril Mohammed
Adeniyi Joy

From the Vice Chancellor

Courseware development for instructional use by the Centre for Open and Distance Learning (CODL) has been achieved through the dedication of authors and the team involved in quality assurance based on the core values of the University of Ilorin. The availability, relevance and use of the courseware cannot be timelier than now that the whole world has to bring online education to the front burner. A necessary equipping for addressing some of the weaknesses of regular classroom teaching and learning has thus been achieved in this effort.

This basic course material is available in different electronic modes to ease access and use for the students. They are available on the University's website for download to students and others who have interest in learning from the contents. This is UNILORIN CODL's way of extending knowledge and promoting skills acquisition as open source to those who are interested. As expected, graduates of the University of Ilorin are equipped with requisite skills and competencies for excellence in life. That same expectation applies to all users of these learning materials.

Needless to say, that availability and delivery of the courseware to achieve expected CODL goals are of essence. Ultimate attention is paid to quality and excellence in these complementary processes of teaching and learning. Students are confident that they have the best available to them in every sense.

It is hoped that students will make the best use of these valuable course materials.

Professor S. A. Abdulkareem
Vice Chancellor

Foreword

Courseware remains the nerve centre of Open and Distance Learning. Whereas some institutions and tutors depend entirely on Open Educational Resources (OER), CODL at the University of Ilorin considers it necessary to develop its own materials. Rich as OERs are and widely as they are deployed for supporting online education, adding to them in content and quality by individuals and institutions guarantees progress. Doing it in-house as we have done at the University of Ilorin has brought the best out of the Course Development Team across Faculties in the University. Credit must be given to the team for prompt completion and delivery of assigned tasks in spite of their very busy schedules.

The development of the courseware is similar in many ways to the experience of a pregnant woman eagerly looking forward to the D-day when she will put to bed. It is customary that families waiting for the arrival of a new baby usually do so with high hopes. This is the apt description of the eagerness of the University of Ilorin in seeing that the centre for open and distance learning [CODL] takes off.

The Vice-Chancellor, Prof. Sulyman Age Abdulkareem, deserves every accolade for committing huge financial and material resources to the centre. This commitment, no doubt, boosted the efforts of the team. Careful attention to quality standards, ODL compliance and UNILORIN CODL House Style brought the best out from the course development team. Responses to quality assurance with respect to writing, subject matter content, language and instructional design by authors, reviewers, editors and designers, though painstaking, have yielded the course materials now made available primarily to CODL students as open resources.

Aiming at a parity of standards and esteem with regular university programmes is usually an expectation from students on open and distance education programmes. The reason being that stakeholders hold the view that graduates of face-to-face teaching and learning are superior to those exposed to online education. CODL has the dual-mode mandate. This implies a combination of face-to-face with open and distance education. It is in the light of this that our centre has developed its courseware to combine the strength of both modes to bring out the best from the students. CODL students, other categories of students of the University of Ilorin and similar institutions will find the courseware to be their most dependable companion for the acquisition of knowledge, skills and competences in their respective courses and programmes.

Activities, assessments, assignments, exercises, reports, discussions and projects amongst others at various points in the courseware are targeted at achieving the objectives of teaching and learning. The courseware is interactive and directly points the attention of students and users to key issues helpful to their particular learning. Students' understanding has been viewed as a necessary ingredient at every point. Each course has also been broken into modules and their component units in sequential order.

At this juncture, I must commend past directors of this great centre for their painstaking efforts at ensuring that it sees the light of the day. Prof. M. O. Yusuf, Prof. A. A. Fajonyomi and Prof. H. O. Owolabi shall always be remembered for doing their best during their respective tenures. May God continually be pleased with them, Aameen.

Bashiru, A. Omipidan
Director, CODL

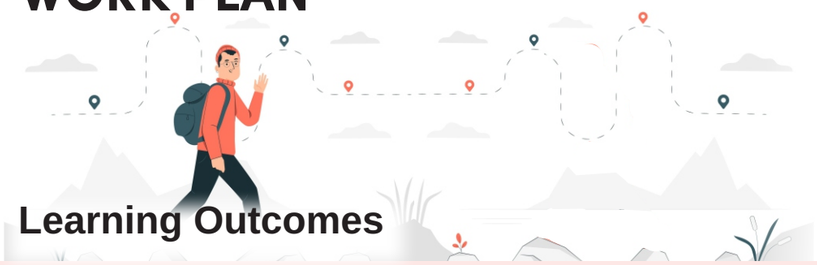
Course Guide

Here in this course, CSC 233, which is a 2 units credit course with six modules of 21 study units. I will be providing you with the principles of object-oriented programming which includes Data Abstraction concerning data and behaviour of object, Hiding and Encapsulation of Data. The accessibility level of objects to achieve data abstraction will be discussed. I will also further in this course explain in details the principle of inheritance and polymorphism. I will look into class, object and methods. Relationship between classes and class Hierarchy will also be discussed which will further describe Abstract class and class library. Object-Oriented Design Approach/Object Modeling is examined. Identification of class which include class Attribute and methods in the problem statement is explained. All discussions on the topic will be illustrated using java language. Fundamentals of java language are introduced at the begin for easy understanding of the concepts of object-oriented programming. Finally, Java operators and expressions are discussed

Course Goal

The goal of this course is to introduce you to concepts of Object Oriented Programming (OOP) using java programming language to solve real-life computational problems.

WORK PLAN



Learning Outcomes

At the end of this course, you should be able to:

i. highlight the fundamentals of java language;
ii. discuss the basic concepts used in OOP;
iii. apply OOP concept in problem-solving;
iv. identify class and object in problem statement;
v. describe object-oriented analysis and design;
vi. describe the various benefits provided by OOP; and
vii. explain the programming applications of OOP.

Week 01 → Week 02 → Week 03

Course Infomation

This is a compulsory course for students in the Departments of Chemistry, Industrial Chemistry, Geology, Mathematics, Computer Science. You are expected to participate in all the course activities and have minimum of 75% attendance to be able to write the final examination

Module 3

Class and Object

UNIT 1: Class

UNIT 2: Object

Course Guide

Module 1

Basic Programming in Java

UNIT 1: Identifiers

UNIT 2: Control Statement

UNIT 3: Single Dimensional Array

UNIT 4: Multi-Dimensional Array

Module 2

Methods

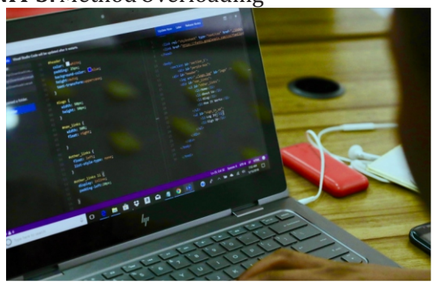
UNIT 1: Defining Method

UNIT 2: Calling Method

UNIT 3: Passing Parameter

UNIT 4: Method Overriding

UNIT 5: Method Overloading



Module 4

Basic concepts of Object-Oriented Programming

UNIT 1: Data Abstraction

UNIT 2: Data Hiding

UNIT 3: Encapsulation

UNIT 4: Inheritance

UNIT 5: Polymorphism

UNIT 6: Object Modeling

Module 5

Java Operators and Expressions

UNIT 1: Operators

UNIT 2: Expression

Course Requirements

Requirements for success

The CODL Programme is designed for learners who are absent from the lecturer in time and space. Therefore, you should refer to your Student Handbook, available on the website and in hard copy form, to get information on the procedure of distance/e-learning. You can contact the CODL helpdesk which is available 24/7 for every of your enquiry.

Visit CODL virtual classroom on http://codllms.unilorin.edu.ng. Then, log in with your credentials and click on CSC233. Download and read through the unit of instruction for each week before the scheduled time of interaction with the course tutor/facilitator. You should also download and watch the relevant video and listen to the podcast so that you will understand and follow the course facilitator.

At the scheduled time, you are expected to log in to the classroom for interaction.

Self-assessment component of the courseware is available as exercises to help you learn and master the content you have gone through.

You are to answer the Tutor Marked Assignment (TMA) for each unit and submit for assessment.

Embedded Support Devices

Support menus for guide and references

Throughout your interaction with this course material, you will notice some set of icons used for easier navigation of this course materials. We advise that you familiarize yourself with each of these icons as they will help you in no small ways in achieving success and easy completion of this course. Find in the table below, the complete icon set and their meaning.

 **Introduction**	 **Learning Outcomes**	 **Main Content**

 **Summary**	 **Tutor Marked Assignment**	 **Self Assessment**
 **Web Resources**	 **Downloadable Resources**	 **Discuss with Colleagues**
 **References**	 **Futher Reading**	 **Self Exploration**

Grading and Assessment

TMA

CA

Exam

Total

Module 1
Basic Programming in Java

Units

Identifiers

Control Statement

Single Dimensional Array

Multi-Dimensional Array

UNIT 1

Identifiers

Introduction

In this unit, you will be introduced to java identifiers, the purpose of identifiers and their types. You will also learn how to declare an identifier and the various data types and the range of value associated with each type.

Learning Outcomes

At the end of this unit, you should be able to:

i. create a valid identifier with the appropriate rule;
ii. state the scope of the identifier created;
iii. list five types of data type;
iv. declare a variable; and
v. assign value to a variable.

Main Content

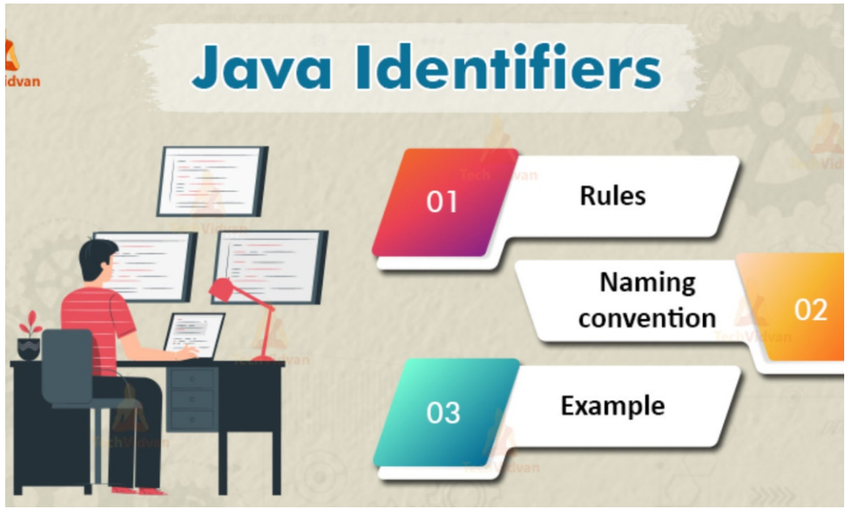
Identifiers

4 mins

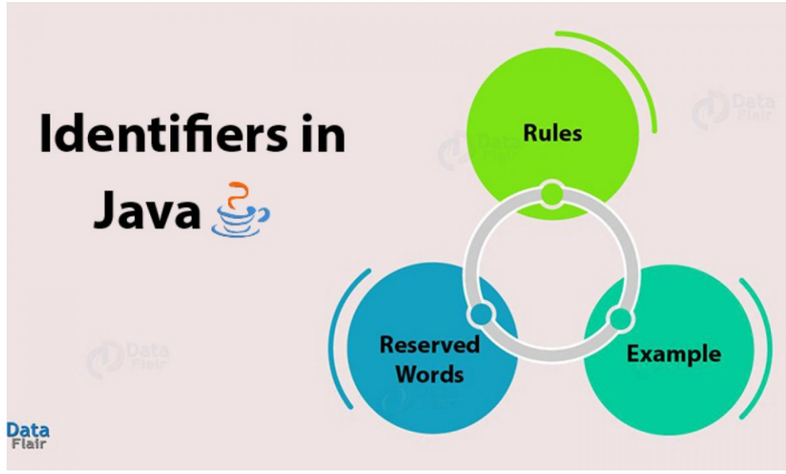
I want you to know that identifiers are the names of variables, methods, classes, packages and interfaces. They are named references created in the memory to identify individual items needed in programming. It will interest you to know they are names of items needed in programming. Very importantly, you must note that identifiers MUST obey the following rules:

i. An identifier is a sequence of characters that consists of letters, digits, underscores(_) and dollar signs ($).
ii. An identifier must start with a letter, an underscore (_) or a dollar sign ($). It cannotstart with a digit.
iii. An identifier cannot be a reserved word. Reserved words are words that have special meaning to the programming language you are using.
iv. An identifier cannot be true, false, or null.
v. An identifier can be of any length.

If you follow the above rules stated, you can infer that the examples of valid identifier names are $2, Compute Area, area, radius, and show Message Dialog; they are legal identifiers. On the other hand, if you follow the rules, you will see that 2A and d+4 are not legal identifiers, why? because they do not follow the rules. Java is case sensitive; hence identifier: My name and my name are not the same.



01 | ARPANET java identifier
source| Vidvan tech



01 | Identifiers in java
source| Tech vidvan.com

Variables

2 mins

Now let's talk about Variables. Variables are identifiers used to store values in the memory to be used later in a program. They are referred as variables because their values can be changed. In a program, variables can have a particular data type depending on the data it is going to hold. Variable can have a numerical value, string which comprises of letters and digits or other data types we shall discuss later.

To use a variable, you should know that it must be declared by telling the compiler its name as well as what type of data it can store. It is of important note for you to know that what the variable declaration does is to inform the compiler to allocate appropriate memory space for the variable based on its data type.
I'll show you below what the syntax for declaring a variable is

```
data type variableName;
```

Let's take a look at some examples of variable declarations in java

```
intcount;          // Declare count to be an integer variable;
```

```
doubleradius;   // Declare radius to be a double variable;
stringsurname;          // Declare surname to be a string variable;
```

In the examples we just considered above, the count is declared as an integer type, what this should mean to you is that it will store only integer values. In the same vein, the radius will store double data type, and the surname will store the string data type.

Assignment Statements and Assignment Expressions

1 min

An assignment statement is used to store a value into a variable. Now, let us break that down, value can be assigned to a variable by using an assignment statement. In Java, the equal sign (=) is used as the assignment operator. The syntax for assignment statements is as follows:

```
variable = expression;
```

taking a look at the above syntax, you would see that an assignment expression represents a computation involving values, variables, and operators which indicates the relation between them. Let's consider these examples below, consider the following code:

```
int x = 20;                          // Assign 20 to variable x
double radius = 1.5;                 // Assign 1.5 to variable radius
x = 5 * (3 / 2) + 3 * 2;             // Assign the value of the
expression to x
x = y + 1;                           // Assign the addition of y and
1 to x
area = radius * radius * 3.14159;    // Compute area
```

A variable can also be used in an expression. For example,

```
x = x + 1;                           // Assign the value of x + 1 to x
```

In this assignment statement, you should note that the result of x + 1 is assigned to x. If x is 1 before the statement is executed, then it becomes 2 after the statement is executed.
To assign a value to a variable, the variable name must be on the left of the assignment operator. Thus, 1 = x would be very wrong.

Constants

1 min

Like it was mentioned earlier when we were talking about variables, the value may change in the course of execution. However, you should be aware that some valuesare needed to remain unchanged. Such variable is called constant variable, which represents permanent data that never changes. If a variable is used frequently, you don't want to keep typing the value every time the value is needed in the program, then such variable is declared as constant. The syntax for declaring a constant is:

final data type *CONSTANTNAME* = *VALUE*;

Try to understand that a constant must be declared and initialized in the same statement. The word **final** in the above syntax is a Java keyword for declaring a constant. For example, you can declare constant as follows:

```
final double PI = 3.14159;          // Assign the value 3.14159 to
variable PI
```

By convention, constants are named in uppercase: PI, not pi or Pi.

Numeric Data Types and Operations

1 min

SAQ 4

You should be aware that Data types inform the program of the different sizes and values that can be stored in the variable. I want you to understand that data type in java is of two types. The Primitive data type and Non-primitive data type.

Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.
Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.
You should note that Every primitive data type has a range of values. The compiler allocates memory space for each variable or constant according to its data type. Java provides eight primitive data types for numeric values, characters, and Boolean values. I want you to know that this unit introduces numeric data types.

Numerical Data Type and its Storage size

Data Type	Storage Size
Byte	8-bit sign
Short	16-bit sign
Int	32-bit sign
Long	64-bit sign
Float	32-bit IEEE
Double	64-bit IEEE

Do you know that Java uses four types for integers data types which arebyte, short, int, and long? It will interest you to know that you need tochoose an appropriate data type for your variable base on what you need it for. For example, if you need a variable for a value within a range of byte, declare the variable as a byte. Java uses two types for floating-point numbers: float and double. The double type is twice as big as the float. So, the double is known as double-precision, float as single precision.Normally you should use the double type because it is more accurate than the float type.

Numerical Operators

2 mins

I want you to bear in mind that the numerical operators in java include the standard arithmetic operators such as addition (+), subtraction (–), multiplication (*), division (/) and the remainder (%). I want you to also know that in java, the result of integer division is an integer. It is important you note this example, the fractional part of 7/2 is truncated to be 3 and not3.5, and –7 / 2 yields -3, not–3.5. To perform regular mathematical division, one of the operands must be a floating-point number. For example, 7.0 / 2 yields 3.5 or 7/2.0 also yields 3.5.

Have it at the back of your mind that the % operator yields the remainder after division. The left-hand operand is the dividend and the right-hand operand is the divisor. Therefore, 7 % 3 yields 1, 12 % 4 yields 0, 26 % 8yields 2, and 20 % 13 yields 7.

03 | operators in java
source| edureka.co

Shorthand Operators

2 mins

Try to understand that very often the current value of a variable is used, modified, and then reassigned back to the same variable such as when there is a need to increment a variable value. Such is achieved in java by shorthand operator. For example, the following statement adds the current value of i with 8 and assigns the result back to i:

04 | java script shorthand operator
source| brandon more

```
i = i + 8;          // Assign the value i + 8  to variable i
```

The above example can be represented in Java as

```
i += 8;          // Assign the value i + 8  to variable I. same thing as above
```

The += is called the addition assignment operator.

String Type

2 mins

I want you to know that the char type represents only one character. To represent a string of characters, use the datatype called String. For example, the following code declares the message to be a string with value “Welcome to Java”.

```
Stringmsg = "Welcome to this Workshop ";
```

For now, we discuss only how to declare a String variable, how to assign a string value to the variable and to concatenate strings.Two strings can be joined or concatenated using the plus sign (+) operator called concatenation operator.

05 | Java string functioning in Java with Example
source| Edureka

Naming Conventions

2 mins

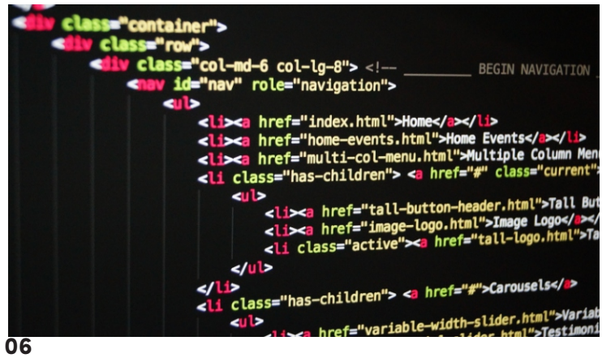
You should always bear in mind that it is important that you choose descriptive names with straightforward meanings for the variables, constants, classes, and methods in your program. Names are case sensitive in java. Below are the conventions for naming variables, methods, and classes in java.

i. Use lowercase for variables and methods. If a name consists of several words, oncatenatethem into one, making the first-word lowercase and capitalizing the first letter of each subsequent word—for example, the variables radius and area and the method showInputDialog.

ii. Capitalize the first letter of each word in a class name—for example, the class namesComputeArea, Math, and JOptionPane.

iii. Capitalize every letter in a constant, and use underscores between words—for example, the constants PI and MAX_VALUE.

I want you to bearin mind that It is important to follow the naming conventions to make programs easy to read. It is necessary not to choose class names that are already used in the Java library. For example, you know Mathclass is defined in Java, you should not name your class Math. You should also avoid using abbreviations for identifiers. Using complete words is more descriptive. For example,matricNumber is better than matNum.



06 | css naming conventions that will save you hours of debugging
source| freecodecamp

06

Summary

In this unit, I discussed identifiers, naming the identifiers and the data type of the identifier declared. Various data type operators were also discussed. Constant variable and its importance were explained. Finally, standard naming convention in java was also discussed.

In this unit, you have lerant:

i. Identifiers;

ii. Naming the identifiers;

iii. Data type of the identifier;

iv. Data type operators;

v. Constant variable; and

vi. Naming convention.

Self-Assessment Questions

1. create an identifier using the identifier naming rule convention
2. identify valid identifier name from the list
   a++, --a, 4#R, $4, #44, apps, class, public, int, x, y, radius
3. identify invalid identifier name from the list
   a++, --a, 4#R, $4, #44, apps, class, public, int, x, y, radius
4. list five types of data type
5. declare an integer variable named miles with initial value 100;
6. Declare a double constant named KONSTATNT with value 10.9;

Tutor Marked Assessment

- Identify valid identifier name from the list and give reasons why they are valid
  a++, --a, 4#R, $4, #44, apps, class, public, int, x, y, radius
- Identify invalid identifier name from the list and state the reason why they are not valid.
  a++, --a, 4#R, $4, #44, apps, class, public, int, x, y, radius
- list five types of data type

Further Reading

- Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.
- Eckel, B. (2003). Thinking in JAVA: Prentice Hall Professional.
- Keogh, J., Giannini, M., & Rinaldi, W. (2004). OOP demystified: McGraw-Hill, Inc.
- Lambert, K., & Osborne, M. (2006). Fundamentals of Java: AP* Computer Science Essentials for the A & AB Exams: Course Technology Press.
- Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.

References

- Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.
- Eckel, B. (2003). Thinking in JAVA: Prentice Hall Professional.
- Keogh, J., Giannini, M., & Rinaldi, W. (2004). OOP demystified: McGraw-Hill, Inc.
- Lambert, K., & Osborne, M. (2006). Fundamentals of Java: AP* Computer Science Essentials for the A & AB Exams: Course Technology Press.
- Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.

UNIT 2

Control Statement

Introduction

As a way of introduction to this unit, you will be introduced to control statements, Boolean expressions, Boolean data types, values, comparison operators and expressions.You will also learn conditional statement and selection statements.

Learning Outcomes

At the end of this unit, you should be able to:

1. Discuss the Boolean data type
2. Use IF Statement
3. Declare the Boolean data type
4. Demonstrate Switch Statement
5. Control program with For loop

Main Content

Boolean Data Type

1 min

SAQ 1 & 2

I want you to know that in order to compare two values, such as whether a value is greater than 0, equal to 0 or less than 0, Java provides six comparison operators known as relational operators which results into Boolean value. I want you to bear in mind that a variable that holds a Boolean value is known as a Boolean variable. The boolean data type is used to declare Boolean variables. A boolean variable can hold one of the two values: true and false. The table 1.2.1 below describes to you the relational operators

Relational Operator

Operator	Name
<	Less than
<=	Less than or Equal to
>	Greater than
>=	Greater or Equal to
==	Equal to
!=	Not Equal to

Table 1.2.1:

The result of the comparison is a Boolean value is either true or false. For example, the following statement declares a Boolean variable and assigns a true value to the variable.

```
boolean lights On = true;          // Assign the value true to boolean variable lightOn
```

You should note that True and False are literals, just like a number such as 10. True and False are reserved words and you cannot use them as identifiers in your program.

01

01 | css naming conventions that will save you hours of debugging

source|

Boolean data type and its exmples in java programming

IF Statement

4 mins

SAQ 3

This unit introduces to you selection statements. Java has several types of selection statements which are listed below:

i. one-way if statements
ii. two-way if statements
iii. nested if statements
iv. switch statements,
v. conditional expressions.

i. one-way if statements

I want you to try to understand that a one-way if statement executes an action if and only if the condition is true. The syntax for a one-way if statement is shown below

```
if (boolean-expression)
{//Set of code to execute if the condition is true}
```

The execution flow chart for one way if statement is shown below.

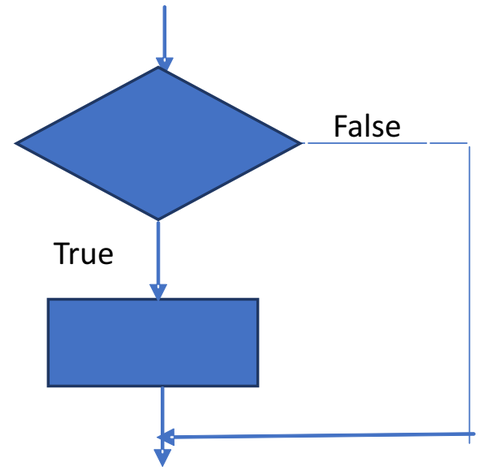


Figure 1.2.1: One way If Statement

SAQ 4

It will intrest you to know that If the boolean expression evaluates to true, the statements in the block are executed. As an example, see the following code:

```
if (radius >= 0)
{
area = radius * radius * PI;
```

```
System.out.println("The area for the circle of radius " +
radius + " is " + area);
}
```

Also try to note that If the value of the radius is greater than or equal to 0, then the area is computed, and the result is displayed; otherwise, the two statements in the block will not be executed. The boolean-expression is enclosed in parentheses () and statements to execute if the condition is true are in the brace bracket {}.

ii. Two-way IF Statements

Don't forget, A one-way if statement takes an action if the specified condition is true. If the condition is not true, then nothing is done. But what if you want to take alternative actions when the condition is not true? You can use a two-way if statement for such condition. The actions that a two-way if statement specifies differ based on whether the condition is true or false.

Here is the syntax for a two-way if statement:

```
if (boolean-expression)
{//Code to execute if condition is true;}
else{//Code to execute if condition is true or false;}
```

Two way if statement is also called IF-Else Statement.

The execution flow chart for two way if statement is shown below.

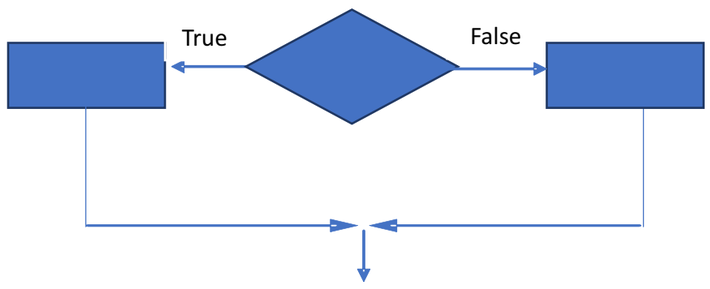


Figure 1.2.2: Two way If Statement condition

If the boolean expression evaluates to true, the statement(s) for the true case is executed; otherwise, the statement(s) for the false case is executed.

For example, consider the following code:

```
if (radius >= 0)
{area = radius * radius * PI;System.out.println ("The area for the
circle of radius " +radius + " is " + area);}else {System.out.println
("Negative Value is Not Allow");}
```

If radius >= 0 is true, the area is computed and displayed; if it is false, the message"NegativeValue is Not Allow" is printed.

As usual, the braces can be omitted if there is only one statement within them. The braces enclosing the System.out.print ("Negative Value is Not Allow ") statement can, therefore, be omitted in the preceding example.

iii. Nested IF Statements

I want you to know that the statement following true condition can be any valid Java statement, including another if or if ... else statement. The inner if statement is said to be nested inside the outer if statement. The inner if statement can contain another if statement; in fact, there is no limit to the depth of the nesting.

For example, the following is a nested if statement:

```
if (m >n){if (a >b)System.out.println ("i and j are greater than
k");}else System.out.println("i is less than or equal to k");
```

The if (a>b) statement is nested inside the if (m>n) statement.

The nested if statement can be used to implement multiple alternatives.

iv. Switch Statements

I want you to aware that there is no limit to using the nested if statement. However, you should know thatoveruse of nested if statements make a program difficult to read and tracing error in such a case can be impossible. Java introduces a switch statement to handle multiple conditions efficiently and better than nested if statement does.

Here I will like to present to you the full syntax for the switch statement:

```
switch (switch-expression)
{
case value1: statement(s)1;
break;
```

```
case value2: statement(s)2;
break; ...

case valueN: statement(s)N;
break;
default: statement(s)-for-default;
}
```

The switch statement follows these rules:

a. The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

b. The value1, value2 to valueN must have the same data type as the value of the switch-expression. Note that value1 and valueN are constant expressions, meaning that they cannot contain variables, such as 1 + x.

c. When the value in a case statement matches the value of the switch-expression, the statements starting from this case are executed until either a break statement or the end of the switch statement is reached.

d. The keyword break is optional. The break statement immediately ends the switch statement.

e. The optional default case can be used to perform actions when none of the specified cases matche the switch-expression.

f. The case statements are checked in sequential order, but the order of the cases (including the default case) does not matter. However, it is good programming style to follow the logical sequence of the cases and place the default case at the end.

Try to understand that once a case is matched, the statements starting from the matched case are executed until a break statement, or the end of the switch statement is reached. This is referred to as fall-through behaviour.

v. Conditional Expressions

It is important you know that the conditional expressions are in a completely different style, with no explicit if in the statement.

The syntax for conditional expression is given below:

```
boolean-expression ?expression1 : expression2;
```

The result of this conditional expression is expression1 is executed if boolean-expression is true; otherwise, the result is expression2 is executed.

Suppose you want to assign the larger number between variable num1 and num2 to the max. You can simply write a statement using the conditional expression:

```
max = (num1 > num2) ? num1 : num2;
```


For another example that I will want you to take note of, the following statement displays the message “num is even” if num is even, and otherwise displays “num is odd.”

```
System.out.println((num % 2 == 0) ? "num is even" : "num is odd");
```

Loops

3 mins

SAQ 7,8 & 9

You should know that Java provides a powerful construct called a loop that controls how many times an operation or a sequence of operations is performed repeatedly in succession. Using a loop statement, the computer can be programmed to print a string a hundred times without having to code the print statements a hundred times. Loops are constructs that control repeated executions of a block of statements. I want you to have it at the back of your mind that the concept of looping is fundamental to programming. It will interest you to know that Java provides three types of loop statements:

i. while loops
ii. do-while loops
iii. for loops

i. while loops

I want you be aware that the syntax for the while loop is as given as

```
while (loop-continuation-condition)
{// Loop body
Statement(s);
```

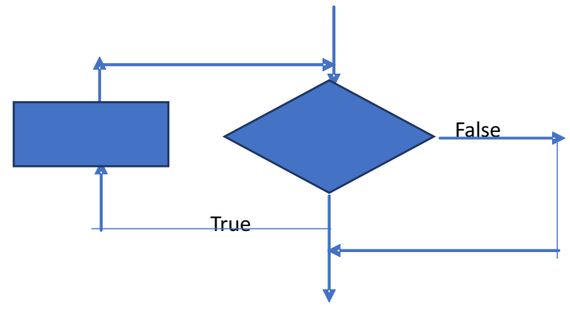


Figure1.2.3: while-loop flow chart

Figure1.2.3 shows the while-loop flow chart. You should be aware that it is the part of the loop that contains the statements to be repeated which is called the loop body. A one-time execution of a loop body is referred to as an iteration of the loop. I want you have it at the back of your mind that Each loop contains a loop-continuation-condition, a Boolean expression that controls the execution of the body. It is evaluated each time to determine if the loop body is executed. If its evaluation is true, the loop body is executed; if its evaluation is false, the entire loop terminates, and the program control turns to the statement that follows the while loop. Suppose we have the while statement given as

```
int count = 0;
while (count <50)
{System.out.println("Welcome to Java!");
count++}
```

It is important you know that the loop-continuation-condition is (count <50) and the loop body contains two statements System.out.println ("Welcome to Java!"); and count++; In this example, you should know exactly how many times the loop body needs to be executed. So, a control variable count is used to count the number of executions. This type of loop is known as a counter-controlled loop.

ii. do-while loops

Try to understand that the do-while loop is a variation of the while loop. Its syntax is given, as shown below:

```
do
{// Loop body; Statement(s);} while (loop-continuation-condition);
```

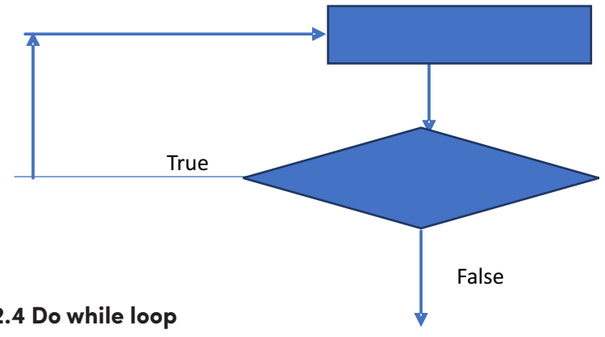


Figure1.2.4 Do while loop

Its very vital for you to know that the loop body is executed first. Then the loop-continuation-condition is evaluated. If the evaluation is true, the loop body is executed again; if it is false, the

do-while loop terminates. The difference between a while loop and a do-while loop is the order in which the loop-continuation-condition is evaluated and the loop body executed.

The while loop and the do-while loop have the same expressive power. Sometimes one is a more convenient choice than the other.

iii. for loops

In general, you should bear in mind that the syntax of a for loop is as given as:

```
for(initial-action;loop-continuation-condition;action-after-each-iteration)
{// Loop body; Statement(s);}
```

The flow chart of the for loop is in figure 1.2.5.

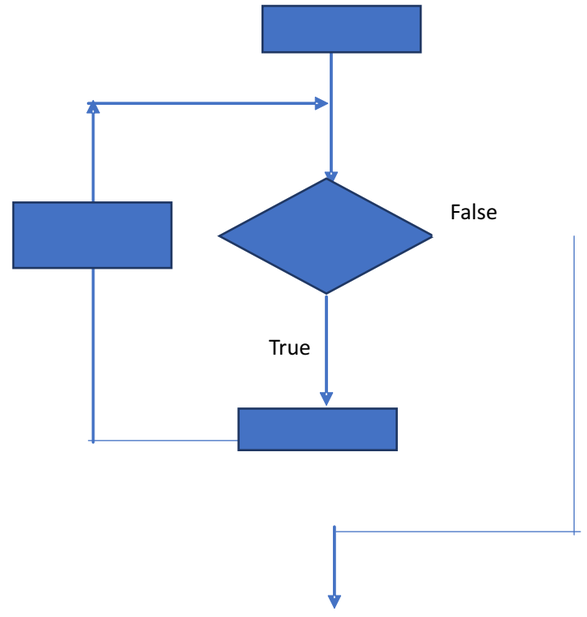


Figure 1.2.5: For loop

I want you to understand that the for-loop statement starts with the keyword for, followed by the control structure of the loop in an enclosed pair of parentheses. This control structure consists of initial-action,loop-continuation-condition, and action-after-each-iteration separated by semicolons. The control structure is followed by the loop body enclosed inside braces bracket.

Let me say that A for loop generally uses a variable to control how many times the loop body is executed and when to terminate the loop. This variable is referred to as a control variable. The initial action often initializes a control variable, the action-after-each-iteration usually increases or decreases the control variable, and the loop-continuation-condition tests whether the control variable has reached a termination value. For example, the following for loop prints Welcome to Java! ten times:

```
int i;
for (i = 0; i< 10; i++)
{System.out.println ("Welcome to Java!");}
```

A for loop performs an initial action once, then repeatedly executes the statements in the loop body, and acts as an iteration when the loop continuation-condition evaluates to true.

While Loop?

1 min

I want you to know that the while loop and for loop are the types of loops called pretest loops since before the loop body is executed, the continuation condition is checked first while the do-while loop is called a posttest loop since after the loop body is executed, then the condition is checked. You should try to understand that the three forms of loop statements, while, do-while, and for, are equivalent since the three loops can be used to achieve the desired loop.
You can use any of the loop statements that is most comfortable for you. In general, a for loop may be used if the number of repetitions is known in advance, the while loop may be used if the number of repetitions is not fixed, as in the case of reading the numbers until the input is 0. A do-while loop can be used to replace a while loop if the loop body has to be executed before the continuation condition is tested.

For-Each Loop

1 min

You should be aware that Java provides a convenient for loop, known as a for-each loop or enhanced for loop, which use to traverse the array sequentially without using an index variable. For example, the following code displays all the elements in the array my List:

```
for (double u: myList)
{System.out.println(u);}
```

You can read the code as “for each element u in my List do the following.” Note that the variable u, must be declared the same type as the elements in my List.
In general, the syntax for a for-each loop is

```
for (elementType element: arrayRefVar)
{// Process the element}
```

You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

Summary

In this unit, I believe Control statements and loop were discussed. I also believe The use of if control statement and types of IF statements were explained. The control statements can be used to control the flow of the program in the appropriate direction you want. Similarly, iteration of some statements in a loop was examined. Using the loop, iteration can be achieved in programming.
In this unit, you have learnt to:

i. the Boolean data type;
ii. use IF Statement;
iii. use Two way IF statement;
iv. use Nested IF Statement;
v. apply Switch Statement;
vi. use a conditional expression to decide;
vii. control program with For loop;
viii. apply Do while loop;
ix. apply While loop;
x. use Nested loop.

Self-Assessment Questions

1. State the values a boolean variable store
2. declare a variable of Boolean data type
3. use IF Statement to find if a number is greater than zero
4. use Two way IF statement to decide if a number is positive or negative.
5. demonstrate Switch Statement
6. using conditional expression determine when a given value is greater than ten
7. With forloop print 1 to 10;
8. Apply the do-while loop to compute the sum of the first 10 integers.
9. Give the syntax of While loop

the following control statements :

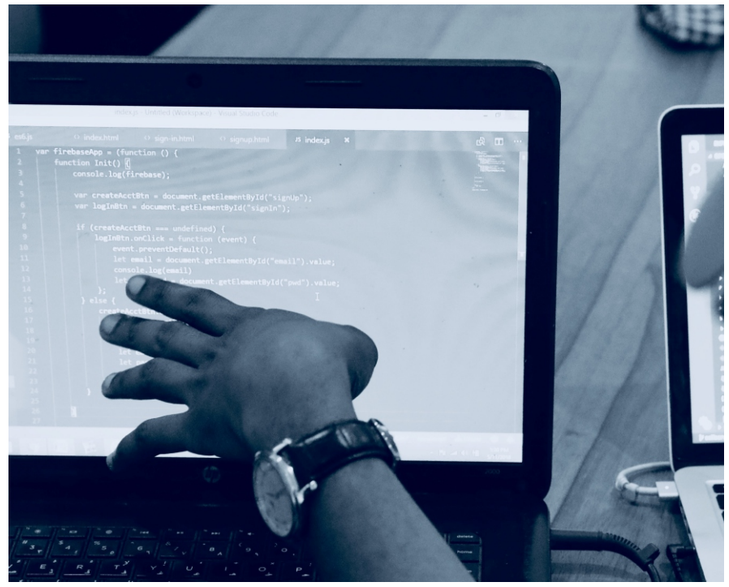
i. If statements.
ii. Switch Statements.
iii. For loop.
iv. While Statements.
v. Do While statements

Further Reading

- Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.
- Eckel, B. (2003). Thinking in JAVA: Prentice Hall Professional.
- Keogh, J., Giannini, M., & Rinaldi, W. (2004). OOP demystified: McGraw-Hill, Inc.
- Lambert, K., & Osborne, M. (2006). Fundamentals of Java: AP* Computer Science Essentials for the A & AB Exams: Course Technology Press.
- Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.

References

- Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.
- Eckel, B. (2003). Thinking in JAVA: Prentice Hall Professional.
- Keogh, J., Giannini, M., & Rinaldi, W. (2004). OOP demystified: McGraw-Hill, Inc.
- Lambert, K., & Osborne, M. (2006). Fundamentals of Java: AP* Computer Science Essentials for the A & AB Exams: Course Technology Press.
- Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.



UNIT 3

Single Dimensional Array

Introduction

In this lesson, I will duly introduce you to the concept of single dimensional array in details.

You will learn to declare single dimensional array, acces and manipulate the elements in the array, copy the contents of one array to another array, pass array to and return array from a method .

Learning Outcomes

At the end of this unit, you should be able to:

1. State single array
2. Assign a value to array
3. Copy one array to another array
4. Pass array to a method
5. Return array from method

Main Content

You should note that in programming, sometimes there is a need to store a sequence of data and process them. So, for instance, in your program you need to read 50 numbers, find the average of the numbers and those number greater than the average. The program will first compute the average and then compare each number with the average to determine those number greater than the average. I want you to know that such a task is achieved using array. An efficient and organized approach is necessary to do this task. I want to bear in mind that Java provides a data structure called array which stores a fixed-size sequential collection of elements of the same type. In the present case, you can store all 100 numbers into an array and access them through a single array variable. In this unit, you will learn about Java arrays.

Basic of Array

2 mins

SAQ 1

I want you to know that An array is a variable that stores a collection of data of the same data type. You should bear in mind that Instead of declaring individual variables, an array variable is declared to hold individual variables. Declaration of array variables does not allocate memory space to the array variable. It creates only a storage location for the array reference. If an array variable does not have a reference, the value of the variable is null. An array cannot be assigned a value until it has been created. After an array variable is declared, then the keyword new operator is used to create an array with the following syntax:

```
arrayVar = newelementType[arraySize];
```

I want you to know that this statement does two things:

(1) it creates an array using new elementType[array-Size];

(2) it assigns the reference of the newly created array to the variable arrayVar.

Declaring an array variable, creating an array, and assigning the reference of

the array to the variable can be combined in one statement, as shown below:

```
element Type array Var = new element Type [array Size];
```

or

```
element Type array Var[] = newelement Type [array Size];
```

For example:

```
double my List[] = newdouble[10];
```

Try to understand that this statement declares an array variable called my List, creates an array of ten elements of double type, and assigns its reference to my List.

To initialize or assign values to the elements of the array, the syntax is:

```
array Ref Var[index] = value;
```

For example, the following code initializes the array.

```
myList[0] = 6.0;
myList[1] = 5.4;
myList[2] = 3.0;
myList[10] = 3.2;
```

The above array declaration and initialization is pictured in Figure 1.3.1 below.

mylist[0]	6.0
mylist[1]	5.4
mylist[2]	3.0
.	.
.	.
.	.
.	.
mylist[9]	3.2

Figure 1.3.1: The array myList with ten elements of **double** data type and **int** indices from 0 to 9.

Its of great importance for you to know that the array elements are accessed using the index. Array indices are 0 based meaning that; they range from 0 to arrayRefVar. length-1. In the example in Figure 1.3.1, myList holds ten double values, and the indices are from 0 to 9.

Try to understand that each element in the array is referenced by the following syntax, known as an indexed variable:

```
arrayRefVar[index];
```

An important example you should note, mylist[0] and myList[9] represent the first and the tenth element in the array myList with ten elements. Though Java uses square brackets, as in myList[9], other languages use parentheses to reference an array element, as in myList(9).

I want you to bear in mind that When space for an array is allocated in the memory, the array size must be indicated to specify the number of elements that an array can store. The size of an array cannot be changed after the array is created. To obtain the size of an array, use arrayRefVar.length.

For example, myList.length is 10.

Once an array is created, its elements are assigned the default value based on the data type. 0 for the numeric primitive data types, '\u0000' for char types, and false for boolean types.

Copying Array

2 mins

SAQ 2 & 3

Sometimes you should note that a need arises in a program to duplicate an array or a part of an array, in such cases an assignment statement, equal sign (=), is used as follows

list2 = list1;

This statement does not actually copy the contents of the array referenced by list1 to list2, but merely copies the reference value from list1 to list2. It will interest you to know that after this statement, list1 and list2 reference the same array. The array previously referenced by list2 is no longer referenced, and it becomes garbage for collection by the Java Virtual Machine automatically.

In Java, I want you to understand that assignment statements are used to copy primitive data type variables, but not arrays variables. May I let you know that Assigning one array variable to another array variable actually copies one reference to another and makes both variables point to the same memory location.

Bear in mind that there are three ways to copy arrays to another array as list1 to list2 above are:

i. Use a loop to copy individual elements one by one.

ii. Use the static array copy method in the System class. I discuss class later in module 3.

iii. Use the clone method to copy arrays. This method is outside the scope of this work.

When Using a loop to copy every element from the original array to the corresponding element in the destination array, I want you to know that we illustrate with the example below to copy source Array to target array using a for loop.

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];
for (int i = 0; i < sourceArray.length; i++)
{
    targetArray[i] = sourceArray[i];
}
sourceArray is the original array to be copied
targetArray is the destination array to copy to.
sourceArray.length is the length of original array
```

Another important approach you to be aware of is to use the arraycopy method in the java.lang.System class to copyarrays instead of using a loop. The syntax for arraycopy is:

Arraycopy (sourceArray, src_pos, targetArray, tar_pos, length);

In the above syntax, you should know that the src_pos is the starting positions in the original array, tar_pos is the starting position in the destination array.

The number of elements copied from the original array ,sourceArray, to destination array, targetArray, is indicated by the length.

For example, you can rewrite the loop using the following statement:

System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);

The arraycopy method in the above example, does not allocate memory space to the destination array. The destination array has been created with its memory space allocated. After the copying takes place, targetArray and sourceArray have the same content but independent memory locations.

Passing Array

2 mins

SAQ 4

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

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

The printArray method can be invoked to display 5,4,3,2,1 and 0 as in the example.

printArray(new int[]{5, 4, 3, 2,1, 0});

you should be aware that Java uses pass-by-value to pass arguments to a method. I want you to know that there are important differences between passing the values of variables of primitive data types and passing arrays.

i. For an argument of a primitive type, the argument's value is passed.
ii. For an argument of an array type, the value of the argument is a reference to an array,it is this reference value that is passed to the method.

The array in the method is the same as the array being passed. So, if you change the array in the method, you will see the change outside the method.

Consider taking the following code:

```
public class Test
{
public static void main(String[] args)
{
        int x = 1; // x represents an int value
        int[] y = new int[10]; // y represents an array of int values
        m(x, y); // Invoke m with arguments x and y
        System.out.println("x is " + x);
        System.out.println("y[0] is " + y[0]);
}
public static void m(int number, int[] numbers)
{
        number = 1001; // Assign a new value to number
        numbers[0] = 5555; // Assign a new value to numbers[0]
}
}
```

You will see that after m is invoked, x remains 1, but y[0] is 5555. This is because y and numbers, although they are independent variables, reference to the same array. When m(x, y) is invoked, the values of x and y are passed to number andnumbers. Since y contains the reference value to the array, numbers now contain the same reference value to the same array.

Returning Array

2 mins

SAQ 5

Just as passing array to a method when it is invoked, you will see that the array can be returned from a method. A method may also return an array. Consider the method shown below. It returns an array that is the reversal of another array:

```
public static int[] reverse(int[] list)
{
        int[] result = new int[list.length];
        for (int i = 0, j = result.length - 1;
        i <list.length; i++, j--)
        {
          result[j] = list[i];
        }
        return result;
}
int[] result = new int[list.length]; //creates a new array called result.

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


For example, I want you to note that the following statement returns a new arraylist2 with elements 6, 5, 4, 3, 2, 1 in reverse order.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

list2 = {6, 5, 4,3, 2, 1};
```

Summary

In this unit, you have learnt:

i. how to declare single array;
ii. that the declaration of an array variable does not allocate any space in memory for the array;
iii. that you cannot assign elements to an array unless it has already been created;
iv. that when an array is created, its elements are assigned the default value of 0 for the numeric primitive data types, '\u0000' for char types, and false for boolean types;
v. how to copy one array to another array;
vi. how to Pass array to a method; and
vii. how to Return array from method.

Self-Assessment Questions

SAQ

1. Declare a single array variable of type double and initialize the array
2. Assign a value to array
3. Copy one array to another array
4. Pass array to a method
5. Return array from method

Tutor Marked Assessment

Using Java,

i. Find the length of an array.
ii. Demonstrate a one-dimensional array.

Further Reading

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures Java: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

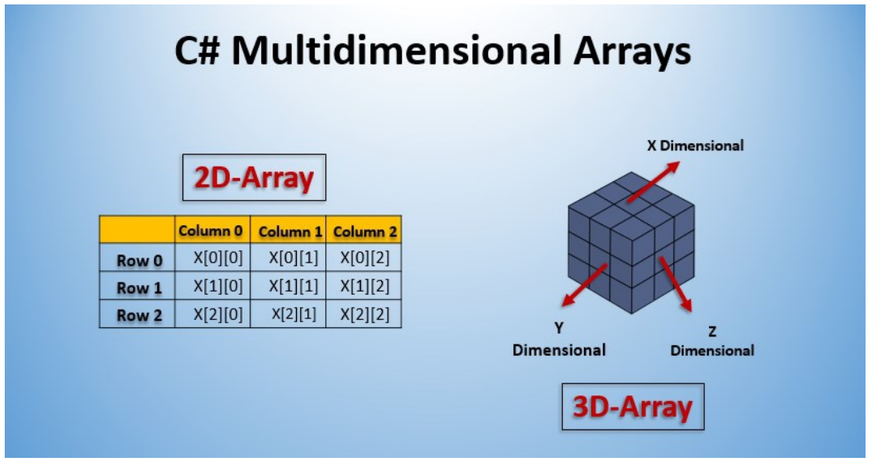
References

Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures i
Java: Cambridge University Press

Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.

Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.

Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.



UNIT 4

Multi-Dimensional Array

Introduction

In the previous unit, you learnt how to use one-dimensional arrays to store linear collections of elements. However, when there is a need to compute two-by-two or higher matrix, you need a two-dimensional or multi-dimensional array for such a matrix or a table. This unit introduces to you two-dimensional array including declaration, assignment of value to the array and passing of the array to other methods for further processing.

Learning Outcomes

At the end of this unit, you should be able to:

1. Represent data using two-dimensional arrays;
2. Declare variables for two-dimensional arrays;
3. Assess elements in a two-dimensional array using row and column indexes;
4. Demonstrate data for two-dimensional arrays; and
5. Pass two-dimensional arrays to methods.

Main Content

Basic of Array

1 min

SAQ 1,2 &3

I want you to know that Like one-dimensional array, two-dimensional array is declared and created before it can be used. To declare and create variables of two-dimensional arrays the syntax is:

elementType[][] ***arrayRefVar***;

or

elementType*arrayRefVar*[][];

it is important for you to note that a two-dimensional array uses [][] for subscript reference.

As an example, here is how a two-dimensional array variable called matrix ofint values is declared:

int[][] ***matrix***;

or

int*matrix*[][];

A two-dimensional array of 5-by-5 int values and assign to matrix using this syntax:

matrix = new int[5][5];

you should be aware that two subscripts are used in a two-dimensional array, one for the row and the other for the column. As in a one-dimensional array, the index for each subscript is of the int type and starts from 0 to variable.lenght – 1.

To assign the value 20 to a specific matrix element at row 3 and column 2, use the following:

matrix[3][2] = 20;

I want you to try to understand that it is a common mistake to try to usematrix[2, 1] to access the element at row 2 and column 1 in java. Rather it should be matrix[2][1]. Bear in mind that as for, Java, each subscript must be enclosed in a pair of square brackets.

Also, its important for you to note that a two-dimensional array can also be declared, created, and initialized using initializer, as shown in the example below.

```
int[][] array = {
                  1   2   3 ,
                  4   5   6 ,
                  7   8   9 ,
                  10  11  12 ,
                 };
```


Processing Two-Dimensional Array

1 min

SAQ 4

I want you to understand that we illustrate this by examples for convenience and simplicity.

Suppose an array matrix is created as follows:

int[][] matrix = new int[10][10];

Here are some examples of two-dimensional processing arrays that I want you to note:

To initialize arrays with input value, its important you know that the following loop initializes the array with user input values as shown below

```
java.util.Scanner input = new Scanner(System.in);
System.out.println("Enter " + matrix.length + " rows and " +
matrix[0].length + " columns: ");
for (int row = 0; row < ; row++)
{
for (int column = 0; column < ; column++)
  {
matrix[row][column] = input.nextInt();
  }


}
```

Passing Two-Dimensional Array

1 min

SAQ 5

You can pass a two-dimensional array to a method in a way that you pass a one-dimensional array. The example with a method that returns the sum of all the elements in a matrix.

01 | passing two dimensional array
source| Programmer sought

```
import java.util.Scanner;
public class PassTwoDimensionalArray {
public static void main(String[] args) {
 // Create a Scanner
Scanner input = new Scanner(System.in);
// Enter array values
int[][] m = new int[3][4];
System.out.println("Enter " + m.length + " rows and
+ m[0].length + " columns: ");
 for (int i = 0; i <m.length; i++)
 for (int j = 0; j < m[i].length; j++)
 m[i][j] = input.nextInt();

// Display result
System.out.println("\nSum of all elements is " +
sum(m);
 }
public static int sum(int[][] m) {
int total = 0;
 for (int row = 0; row <m.length; row++) {
 for (int column = 0; column < m[row].length;
column++) {
 total += m[row][column];
 }
 }
return total;


 }
 }
```

It will intrest you to know that the method **sum** in the above example has a two-dimensional array argument. The number of rows is obtained using m.length, and the number of columns in each row is obtained by m[row].column.

Summary

In this unit, I am very sure you have learnt to:

i. represent data using two-dimensional arrays;
ii. declare variables for two-dimensional arrays;
iii. access array elements in a two-dimensional array using row and column indexes; and
iv. pass two-dimensional arrays to methods.

Self-Assessment Questions

1. explain how to represent data using two-dimensional arrays;
2. declare variables for two-dimensional arrays of size 5,5 to store integer values
3. using row and column indexes, access array elements in a two-dimensional array
4. display data for two-dimensional arrays
   pass two-dimensional arrays to methods

Tutor Marked Assessment

i. How do you use two-dimensional arrays to represent integer value?
ii. declare variables for two-dimensional arrays to store integer values of a 5X5 matrix

Further Reading

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures I Java: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

References

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures IJava: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

Module 2
Methouds

units

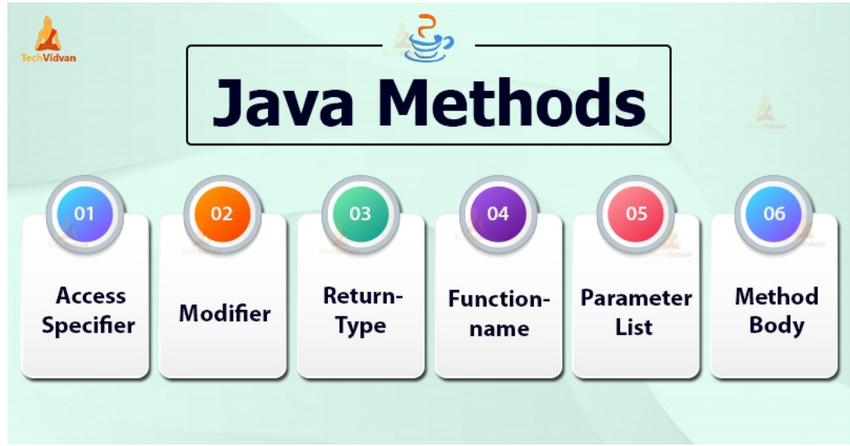
Defining Method

Calling Method

Passing Parameter

Method Overriding

Method Overloading



UNIT 1

Methods

Introduction

I welcome you to this unit where youlearn about methods, how to define a method and its signature. Also, you will learn about Standard and user defined methods. You will be introduced to variable shadowing and hiding.

Learning Outcomes

At the end of this unit, you should be able to:

- define methods;
- invoke methods with a return value;
- invoke methods without a return value, and
- determine the scope of variables.

Main Content

Method Definition

3 mins

SAQ 1

It is very vital for you to note that a method is a collection of statements that perform some specific task and return the result to the caller unit of the program. I want you to alsso know that Method can perform some specific task without necessarily returning any value to the calling program. You should understand Methods save time and help reuse the code without retyping it. In general, it will intrest you to know that method declaration has six components:

i. **Modifier**-: Defines the access type of the method from where it can be accessed in the application. In Java, there are 4 types of access specifiers.
   a. **public**: accessible in all class in the application.
   b. **protected**: accessible within the class in which it is defined and in the subclass(es) of the class in which it is defined.
   c. **private**: accessible only within the class in which it is defined.
   d. **default** (declared/defined without using any modifier): accessible within same class and package within which its class is defined.

ii. The **return type**: The data type of the value returned by the method or void if it does not return a value.

iii. **Method Name**: the rules for field names apply to method names as well, but the convention is a little different.

iv. **Parameter list**: Comma-separated list of the input parameters is defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().

v. **Exception list**: This is the exceptions thrown by the method.

vi. **Method body**: it is enclosed between braces brackets. The code you need to be executed to perform your intended operations in the method.

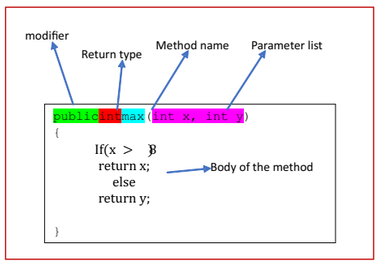


Figure2.1.1: Method Definition

Method signature: I want you to know that method signature consists of the method name and a parameter list (number of parameters, type of the parameters and order of the parameters). The return type and exceptions are not considered as part of it. Method Signature of the above method is

max (int x, int y).

It is very important you know how to name a Method: A method name is typically a single word that can be a verb in lowercase or multi-word, that begins with a verb in lowercase followed by adjective or noun. After the first word, the first letter of each word should be capitalized. For example, findSum,computeMax, setX and getX

You should know that every method must have a name. Otherwise, there is no way it can be called. Let us carefully look into some additional considerations for naming a method:

i. the method name must begin with a letter — any letter will do;
ii. method name cannot begin with a number or special character;
iii. numbers can appear anywhere else in the method name;
iv. the only special character that can be used is the underscore (_). For example, a method name can't include an ampersand (&); and
v. generally, you should know that method has a unique name within the class in which it is defined, but sometimes a method might have the same name as other method names within the same class as in the case of method overloading.

Another vital thing of note I want you to bear in mind is the method header which specifies the modifiers, return value type, method name, and
parameters of the method. A method may return a value. ReturnValueType is the data type of the value the method returns. Some methods perform desired operations without returning a value. In this case, the ReturnValueType is the keyword **void**. If a method returns a value, it is called a Value Returning Method; otherwise, it is a Void Method.
Also another important point I want you to understand is the variables which whendefined in the method header are known as formal parameters or simply parameters. It will intrest you to know that a parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is

referred to as an actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a method. The method name and the parameter list together constitute the method signature. Parameters are optional; that is, a method may contain no parameters. I will want you to note this example, the Math.random() method has no parameters.

The method body contains a collection of statements that define what the method

Types of Methods

3 mins

SAQ 4

I want you to carefully note that the types of methods in Java is dependent on whether a method is defined by the user, or available in a standard library, there are two types of methods:

i. Standard Library Methods
ii. User-defined Methods

i. Standard Library Methods
The standard library methods are built-in methods in Java that are readily available for use, this is very important for you to bear in mind. These standard libraries come along with the Java Class Library (JCL) in a Java archive (*.jar) file with JVM and JRE. For example,print() is a method of java.io.PrintSteam. The print("...") prints the string inside quotation marks. Similarly, sqrt() is a method of Math class. It returns the square root of a number.

ii. User-defined Methods
In Java, I want you to try to understand that there are three different User-defined methods. It will intrest you to know that programmer can develop any type of method depending on the scenario.

1. Static methods: A static method is a method that can be called and executed without creating an object. In general, static methods are used to create instance methods. A static method can be invoked directly via class name, i.e., we don't have to create an object for a class in order to initiate a static method.

2. Instance methods: These methods act upon the instance variables of a class. Instance methods are classified into two types

a. Accessor methods: These are the methods which read the instance variables, i.e., just access the instance variables data. In general, these methods are named by prefixing with "get".

b. Mutator methods: These are the methods, which not only read the instance variables, but also modify the data. Generally, these methods are named by prefixing with "set".

Set Methods Body

1 min

SAQ 5

I want you to understand that set Method is like every other method in which the *method body* is delimited by a pair of *braces* ({}) having codes to perform the method's task(s). let me tell you, mostly, Set method body contains a single statement that assigns the value of the name *parameter* to the class's name *instance variable*, thus storing the value.

I want you to have it at back of your mind that If a method contains a local variable with the *same* name as an instance variable, that method's body will refer to the local variable rather than the instance variable. In this case, the local variable is said to *shadow* the instance variable in the method's body. You should know that the method's body can use the keyword **this** to refer to the shadowed instance variable explicitly. Important of note that I want you to know about setter method:

i. setter method is to set new values or assign new values to an instance variable;
ii. method name should follow naming convention setVariablename();
iii. it should accept some values as an argument. Here method argument should be according to the type of variable;
iv. it should have a statement to assign argument value to the corresponding variable;
v. it does not have any return type. Void should be the method return type; and
vi. in order to set some value to a variable, we need to call the corresponding setter method by passing the required value.

I want you to understand that get method body is not different from the set method described above. However, its important you know that while the set method assigns value to an instance variable, get method retrieves the value of instance variable. Important notes for you to know about the get method are:

i. getter method gets the value of the instance variable;
ii. it follows naming convention getVariablename();
iii. it requires no arguments;
iv. it returns the corresponding variable value;
v. return type must be of the type of variable from the calling method; and
vi. In order to get the variable value, we need to call the corresponding getter method of variable.

```
package DataHiding;

/*
 * This class illustrates basic data hiding
concepts
*/
public class ExampleSetGet
     {
     //  public    -  a violatio n of data hiding
'rules'
            public int height;
      //  private - not visible outside the class
            private float weight;
      //  Here are 'get' and 'set' methods for the
     //    private variable declared above.
            public float GetWeight()
               {
                    return weight;
               }
             public float SetWeight(float weight)
               {
                    if (weight >= 0)
                    this.weight = weight;
                    return this.weight;
               }
     }
```


Constructor

2 mins

I want you to understand that a constructor is a method that bears the same name as its class and is used to produce new instances of the class. It is executed automatically when a class is instantiated with the keyword **new**. It is vital for you to know that It is generally used to initialize object member parameters and allocate the necessary resource to object members.

It will interest you to know that when the object is created, a call to the constructor is made first, and the code in the constructor is then executed. Also, you should know that when you create a new object, automatically a call to the constructor method is made implicitly. Every class has a constructor associated with it and no need to define one in code if the class is fine with the default. Its good you know that the constructor has four different types. It must be noted that:

i. a constructor must have the same name as the class itself;
ii. constructors do not have a return type—not even void;
iii. constructors are invoked using the new operator when an object is created; and
iv. constructorsplay the role of initializing objects.

- **Default Constructor.**
  I want you to bear mind that a constructor without any parameters is called a default constructor; in other words, this type of constructor does not take parameters. I want you to know that the drawback of a default constructor is that every instance of the class created is given an initial value of zero for all numeric fields and null for string and object fields. These initial values cannot be changed.

```
class Example
{
   //Default constructor
Example()
   {
System.out.println("Default constructor");
   }
```

- **Parameterized Constructor.**
  Its important for you to note that a parameterized constructor has at least one parameter when initializing an object. This gives the advantage of initializing each instance of the class with a different value.

```
class Example
{
     /* Parameterized constructor withtwo integer arguments
*/
Example(inti, int j)
     {
System.out.println("constructor with Two parameters");
     }
}
```

- **Private Constructor.**
  I want you to know that Private constructor is created with the private keyword modifier. You should be aware that the purpose of a private constructor is to provide singleton classes. A singleton class limits the number of objects of that class to one. This ensures that no more than one object can be created at a time. Hence, private constructor prevents class instances from being created in any other place order than its class.

```
public class Counter
    {
        private Counter()    //private constrctor declaration
        {
        }
        public static intcurrentview;
        public static intvisitedCount()
        {
            return ++ currentview;
```

Abstract method

1 min

Good of note for you to know is that abstract methods are declared using the **abstract** keyword. You should understand an abstract method consists of a method definition without implementation or no method body.
You should know that an abstract method can only set as either **public** or **protected**. That is, an abstract method cannot be static or final in the method declaration. I want you to note that:

i. abstract methods do not have a body;
ii. they just have method declaration;
iii. a class with an abstract method must be an abstract class;
iv. an abstract class does not need to have an abstract method compulsory;
v. If a regular class extends an abstract class, then the class must have to implement all the abstract methods of abstract parent class else, it has to be declared abstract as well.

For example, the java segment below shows the abstract method declared in an abstract class

```
public abstract class classA
{
   abstract void mehodA();    // abstract method
}
           class classB extends classA
           {
                 // implements the abstract method
                 void mehodA ()
                  {
                  .........
                  }
           }
```

Variable Shadowing

I want to know that Variable shadowing happens when you define a variable in a closure scope with a variable name that is the same as one for a variable you have already defined in an outer scope. In other words, when a local variable has the same name as one of the instance variables, you should know that the local variable shadows the instance variable inside the method block. I want you to know that in the following example, there is an instance variable named x and inside method print Local Variable (), you are shadowing it with the local variable x.

```
class Parent
{
    // Declaring instance variable with name `x`
    String x = "Parent`s Instance Variable";
    public void printInstanceVariable()
     {
          System.out.println(x);
     }
    public void printLocalVariable()
{
// Shadowing instance variable `x` with a local variable with
the same name
     String x = "Local Variable";
     System.out.println(x);
// If we still want to access the instance variable, we do that
by using `this.x`
     System.out.println(this.x);
      }

}
```

variableHiding

I want you to understand that Variable hiding happens when you define a variable in a child's class with the same name as the one you defined in the parent's class. You should also know that a child class can declare a variable with the same name as an inherited variable from its parent class, thus hiding the inherited variable. In other words, when the child and parent classes both have a variable with the same name, the child class' variable hides the parent class' variable. In the following example, I want you to know that we are hiding the variable named x in the child class while its parent class already defines it.

```
class Child extends Parent
{
// Hiding the Parent class's variable `x` by defining a variable
in the child class with the same name.
     String x = "Child`s Instance Variable";
      @Override
     public void printInstanceVariable()
{
     System.out.print(x);
      // If you still want to access the variable from the super
class, you do that by using `super.x`
     System.out.print(", " + super.x + "\n");
          }
     }
```

Summary

In this unit, you have learnt to:

i. identify a method;
ii. define a method;
iii. identify types of methods;
iv. identify components of methods signature; and
v. shadow and hide variables;

Self-Assessment Questions

SAQ

1. define methods with two parameters, add two variables and return their sum;
2. describe how to invoke methods with a return value;
3. explain how you will invoke methods without a return value;
4. state the function of a get method
5. describe the purpose of the set method;

Tutor Marked Assessment

i. explain the components of a method definition;
ii. state what a set method is defined to achieve;

Further Reading

- Arnow, D. M., Weiss, G., & Dexter, S. (2000). *Introduction to programming using java: an object-oriented approach*: Addison-Wesley.
- Eckel, B. (2003). *Thinking in JAVA*: Prentice Hall Professional.
- Keogh, J., Giannini, M., & Rinaldi, W. (2004). *OOP demystified*: McGraw-Hill, Inc.
- Lambert, K., & Osborne, M. (2006). *Fundamentals of Java: AP* Computer Science Essentials for the A & AB Exams*: Course Technology Press.
- Samanta, D. (2006). *Object-oriented Programming with C++ and Java*: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). *An Introduction to object-oriented programming with Java TM*: McGraw-Hill Incorporated.

References

- Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.
- Eckel, B. (2003). Thinking in JAVA: Prentice Hall Professional.
- Keogh, J., Giannini, M., & Rinaldi, W. (2004). OOP demystified: McGraw-Hill, Inc.
- Lambert, K., & Osborne, M. (2006). Fundamentals of Java: AP* Computer Science Essentials for the A & AB Exams: Course Technology Press.
- Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.

UNIT 2

Calling Method

Introduction

In this unit, you will come to understand that methods are part of a class which perform action or task. You will learn to invoke in-built and user-define methods. You will also be introduced to variables and their scopes.

Learning Outcomes

At the end of this unit, you should be able to:

- explain what stack call is;
- use the return value from a method call
- list types of variable
- explain local variables

Main Content

I want you to understand that the type of task to be performed determines the method to invoke. It will interest you to know that users can define their own method. So apart from standard or language in-build methods, every user can build their own method to serve their purpose. This unit describes to you what the user-defined method is and how to call the method to action. Also, you will learn types of variables and their scope.

Types of Method call

2 mins

In creating a method, you define what the method is to do. To use a method, you have to call or invoke the method. There are two ways to call a method, depending on whether the method returns a value or not.
If the method returns a value, a call to the method is usually called a value return method. For example:

```
int larger = max(3, 4);
```

calls max(3, 4) and assigns the result of the method to the variable larger.
Another example of a call that is treated as a value is

```
System.out.println(max(3, 4));
```

Which prints the return value of the method call max(3, 4).
If the method returns void, a call to the method must be a statement. For example,
the method println returns void. The following call is a statement:

System.out.println("Welcome to Java!");

A value-returning method can also be invoked as a statement in Java. In this case, the you simply ignores the return value. I want you to know that this is not often done but is permissible if the caller is not interested in the return value.
I want you to try to note that when a program calls a method, program control is transferred to the called method. A called method returns control to the caller when its return statement is executed or when its method ending closing brace is reached.


```
public class FindMax
{
 /** Main method */
     public static void main(String[] args)
     {
          inti = 7;
          int j = 6;
          int k = int k = max(i, j);
          System.out.println("The maximum between " + i +
           " and " + j + " is " + k);
     }

/** Return the max between two numbers */
public static intmax(int num1, int num2)
     {
          int result;

          if (num1 > num2)
          result = num1;
          else
          result = num2;
}
```

You should also know that this program contains the main method and the max method. The main method is just like any other method except that the Java Virtual Machine JVM invokes it. The main method's header does not change; it remains the same always.
I want to be aware that the main in the above example includes: modifiers public and static, return value type void, method name main, and a parameter of the String[] type. String[] indicates that the parameter is an array of String.
Its worthy of note for you that the statements in the main method may call other methods that are defined in the class that contains the main method or in other classes. In this example, you should know that the main method invokes max(i,j), which is defined in the same class with the main method.

When the max method is invoked, variable i's value 7 is passed to num1, and variable j's value 6 is passed to num2 in the max method. The flow of control transfers to the max method. The max method is executed. When the return statement in the max method is executed, the max method returns the control to its caller, in this case, the main method.

Call Stacks

1 min

SAQ 1

Each time a method is called, I want you to know the system stores parameters and variables in an area of memory known as a stack, which stores elements in last-in, first-out fashion. When a method calls another method, you should be aware that the caller's stack space is kept intact, and new space is created to handle the new method call. When a called method finishes its work and returns to its caller, its associated space is released.

Understanding of stacks helps to comprehend how methods are invoked. Its important for you to note the last example, the variables defined in the main method are i, j, and k. The variables defined in the max method arenum1, num2, and result. The variables num1 and num2 are defined in the method signature and are parameters of the method. Their values are passed through method invocation.

Method Call

2 mins

SAQ 2

It will intrest you to know that a method needs to be called to use its functionality. I want you to carefully look into the following which explains the components of a method.

1 A method is the equivalent of a function in other languages such as C which helps in code reusing. A set of statements make a method, and this method can be invoked through other statements. When invoked (called), all the statements that are a part of the method would be executed. For instance, look at this method:

```
"public static void methodExample() {}".
```

It currently has no code in it, but there are three keywords before the method name. These are public, static, and void.

2 The word public before the method name means that the method itself can be called from anywhere which includes other classes, even from different packages (files) as long as you import the class. Three other words can replace public. You should understand that they are protected, private and default. If a method is protected, then only this class and subclasses can call the method. If a method is private, then

the method can only be called inside the class. The last keyword is really not even a word. This is if you had nothing in the place of public, protected, or private. This is called the default, or package-private. This means that only the classes in the same package can call the method.

3 The second keyword, static means that the method belongs to the class and not an instance of the class. Static methods must be called using the class name:

```
ExampleClass.methodExample();
```

However, I want you to know if the keyword static was not there, then the method can be invoked only through an object. For instance, if the class was called ExampleObject and it had a constructor for making objects, then you could make a new object by typing

```
ExampleObjectobj = new ExampleObject();
```

and call the method with "obj.methodExample();".

4 The last word before the method name is void. The word void means that the method doesn't return anything (it does not return anything when you run the method). If you do want a method to return something, then simply replace the word void with a data type (primitive or reference type) of the object (or primitive type) that you wish to return. Then just add return plus an object of that type somewhere toward the end of the method's code.

5 When calling a method that returns something, you can use what it returns. For example, if a someMethod() returns an integer, then you can set an integer to what it returns with

```
int a = someMethod();
```

6 Some methods require a parameter. A method that requires a parameter of an integer would look like someMethod(int a). When using a method like this, you would write the method name, and then an integer in the parentheses:

```
someMethod(5);
```

or

```
someMethod(n)
```

if n is an integer.

7 Methods can also have multiple parameters, simply separated by commas. If the method someMethod required two parameters, int a and Object obj, it would look like

```
someMethod(int a, Object obj);
```

I want you to try to understand that to use this new method, it would be called by the method name followed by an integer and an Object in parentheses:

```
someMethod(4, thing);
```

where a thing is an Object.

It will interest you to know that there can be three situations when a method is called. I want you to know that a method returns to the code that invoked it when:

a) It completes all the statements in the method
b) It reaches a return statement
c) Throws an exception

```
// Program to illustrate methods in java
import java.io.*;
class Addition
{
     int sum = 0;
     public intaddTwoInt(int a, int b)
     {       sum = a + b; // adding two integer value.
        return sum;      //returning summation of two values.
     }
}
class callingAdd
{
     public static void main (String[] args)
     {
     // creating an instance of Addition class
     Addition add = new Addition();
     // calling addTwoInt() method to add two integer using
     // instance created  in above step.
     int s = add.addTwoInt(1,2);
     System.out.println("Sum of two integer values :"+ s);
     }
}
```


variables and methods

3 mins

SAQ 3 & 4

it is important you know Java allows declaration of variables whenever needed. We can categorize all our variables into three categories, which have different scopes:

I. **Instance variables**: I want you to know that instance variables is defined inside a class and have an object-level scope. It will interest you to know that Instance variables are declared in a class, but outside a method. Space is allocated in the memory, and a slot for each instance variable value is created. Every instance of the class, i.e. each object of the class contains its own copy of these variables. Try to understand the data for one object is separate and unique from the data for another. An instance variable can be declared public or private or default (no modifier). When a variable's value is not to be changed out-side its class, it is declared private. Public variables can be accessed and changed from outside of its class. Instance, variables can have any of the three access modifiers of public, private or protected. However, it's important you note that instance variables cannot be marked as final, abstract or static. The instance variable will get default value means instance variable can be used without initializing it.

```
package com.jbt;
public class VariablesInJava
{
/*
 Below variable is INSTANCE VARIABLE as it is outside any method
and it is not using STATIC modifier with it. It i  s using default
access modifier.
*/
     intinstanceField;
     public void method()
     {
     final String localVariable = "Initial Value";
               System.out.println(localVariable);
     }public static void main(String args[])
     {    VariablesInJavaobj = new VariablesInJava();
/*Instance variable can only be accessed by Object of the class
only as below.  */
     System.out.println(obj.instanceField);
     System.out.println(obj.staticField);
     System.out.println(VariablesInJava.staticField);
System.out.println(new VariablesInJava().instanceField);
     }
}
```

i. Class variables: defined inside a class with the static keyword. Have It at the back of your mind that class variables have class-level scope common to all objects of the same class. Try to understand Class variables, also known as static fields, share characteristics across all Objects within a Class. I want you to also note when a field is declared to be static, only a single instance of the associated variable is created, which is common to all the Objects of that Class. Hence when one Object changes the value of a Class variable, it affects all the Objects of the Class. We can access a Class variable by using the name of the Class and not necessarily using a reference to an individual Object within the Class. Static variables can be accessed even when no Objects of that class exists.

You should also note that the Static keyword can be applied to Method, Variable, Class nested within another Class and Initialization Block. However, it cannot be applied to Class, Constructor, Interfaces, Method Local Inner Class, Inner Class methods, Instance Variables and Local Variables.

```
public class JavaStaticExample {
     static inti = 10;
     static void method()
     {
          System.out.println("Inside Static method");
     }
     public static void main(String[] args)
     {    // Accessing Static method
     JavaStaticExample.method();
     // Accessing Static Variable
     System.out.println(JavaStaticExample.i);
     JavaStaticExample obj1 = new JavaStaticExample();
     JavaStaticExample obj2 = new JavaStaticExample();
     System.out.println(obj1.i);
     // Accessing satic method using reference.
     // Warning by compiler
     obj1.method();
     }
}
```

iii. Local variables: defined inside a method or in any conditional block. It will intrest you to know that they have a block-level scope and are only accessible in the block where they are defined. I want you to know that local variables cannot use any of the access levels since its scope is only inside the method. Final is the Only Non-Access Modifier that can be applied to a local variable. It is important you note that Local variables are not assigned a default value; hence they need to be initialized.

Summary

In this unit, you have learnt:

i. to call a method;
ii. built-in methods;
iii. user define method; and
iv. types of variables and their scope in a method.

Self-Assessment Questions

1. Explain what stack call is;
2. assign the return value from a method call to a variable
3. list types of variable
4. explain local variables

Tutor Marked Assessment

i. explain what makes a variable class variable
ii. explain the return value from a method call
iii. list types of variable
iv. explain instance variables

Further Reading

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures i Java: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

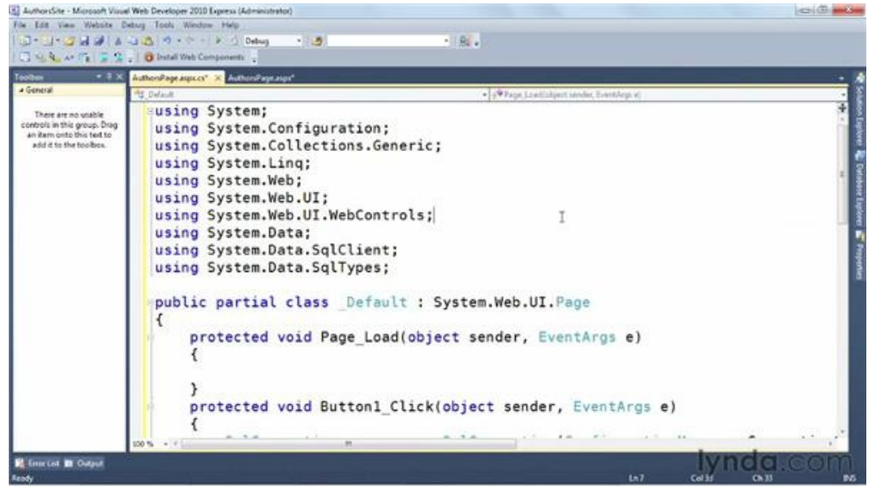
References

Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures i
Java: Cambridge University Press

Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.

Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.

Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.



UNIT 3

Passing Parameter

Introduction

In this unit, you will learn the two most prevalent modes of passing arguments to methods. These two methods will be explained to you in detail.

Learning Outcomes

At the end of this unit, you should be able to:

- pass parameter by value;
- pass parameter by reference;and
- Pass parameter by object.

Main Content

Passing Parameters by Values

2 mins

SAQ 1,2 & 3

I want you to know the power of a method is its ability to work with parameters. The method println can be used to print any string and ,method max to find the maximum between any two int values. I want you to be aware that when calling a method, there is a need to provide arguments, which must be given in the same order as the respective parameters in the method signature. This is known as parameter order association.

Let us take this for example, the following method prints a message n times:

```
public static void nPrintln(String message, int n)
{
for (inti = 0; i < n; i++)
System.out.println(message);
}
```

You can use nPrintln("Hello", 3) to print "Hello" three times. The nPrintln("Hello",
3) statement passes the actual string parameter "Hello" to the parameter message and passes 3 to n;

However, you should know the statement nPrintln(3, "Hello") would be wrong. The data type of 3 does not match the data type for the first parameter, message, nor does the second parameter, "Hello", match the second parameter, n.

I want you to know that the arguments of the calling method must match the signature of the called method as defined in the method signature. Compatible type means that you can pass an argument to a parameter without explicit castings, such as passing an int value argument to a double value parameter.
When you invoke a method with a parameter, the value of the argument is passed to the parameter. This is referred to as pass-by-value. If the argument is a variable rather than a literal value, the value of the variable is passed to the parameter. The variable is not affected, regardless of the changes made to the parameter inside the method.

It will interest you to know that the Java Programming Language features eight primitive data types. Primitive variables are directly

stored in stack memory. Whenever any variable of primitive data type is passed as an argument to a method, the actual parameters are copied to formal arguments and these formal arguments are allocated space in stack memory.

Try to understand that the lifespan of these formal parameters lasts only as long as that method is running, and upon returning, these formal arguments are cleared away from the stack and are discarded.

```
public class PrimitivesUnitTest {

    @Test
public void
whenModifyingPrimitives_thenOriginalValuesNotModified() {
int x = 1;
int y = 2;

        // Before Modification
assertEquals(x, 1);
assertEquals(y, 2);

modify(x, y);

        // After Modification
assertEquals(x, 1);
assertEquals(y, 2);
    }

public static void modify(int x1, int y1) {
        x1 = 5;
        y1 = 10;
    }
}
```

Let's try to understand the declarations in the above program by analyzing how these values are stored in memory:

The variables “x” and “y” in the main method are primitive types, and their values are directly stored in the stack memory
When a call to the method modify() is executed, an exact copy for each of these variables is created and stored at a different location in the stack memory.
You should also know that any modification to these copies affects only the copied values in the memory and leaves the original variables unaltered.

Pass-by-Reference

1 min

SAQ 4

It is important you bear in mind that the fundamental concepts in any programming language are “values” and “references”. Try to understand that in Java, primitive variables store the actual values, whereas non-primitives store the reference variables which point to

the addresses of the objects they're referring to. Both values and references are stored in the stack memory.

Let me say that arguments in Java are always passed-by-value. During method invocation, a copy of each argument, whether it is a value or reference, is created in stack memory which is then passed to the method.

In case of primitives, the value is simply copied inside stack memory which is then passed to the callee method; in case of non-primitives, a reference in stack memory points to the actual data which resides in a heap. When you pass an object, the reference in stack memory is copied and the new reference is passed to the method.

Passing Object References

1 min

I want you to know A Java object, in contrast to primitives, is stored in two stages. The reference variables are stored in stack memory and the object that they are referring to, are stored in a Heap memory.

You should note whenever an object is passed as an argument, an exact copy of the reference variable is created which points to the same location of the object in heap memory as the original reference variable.

As a result of this, whenever you make any change in the same object in the method, that change is reflected in the original object. However, if you allocate a new object to the passed reference variable, then it won't be reflected in the original object.

Summary

In this unit, I am sure you have learnt to pass:

i. parameter by value;
ii. parameters by reference; and
iii. parameter by object.

Self-Assessment Questions

SAQ

1. explain Passing Arrays to Methods
2. explain returning an Array from a method.
3. describe pass parameter by value
4. explain pass parameter by reference

Tutor Marked Assessment

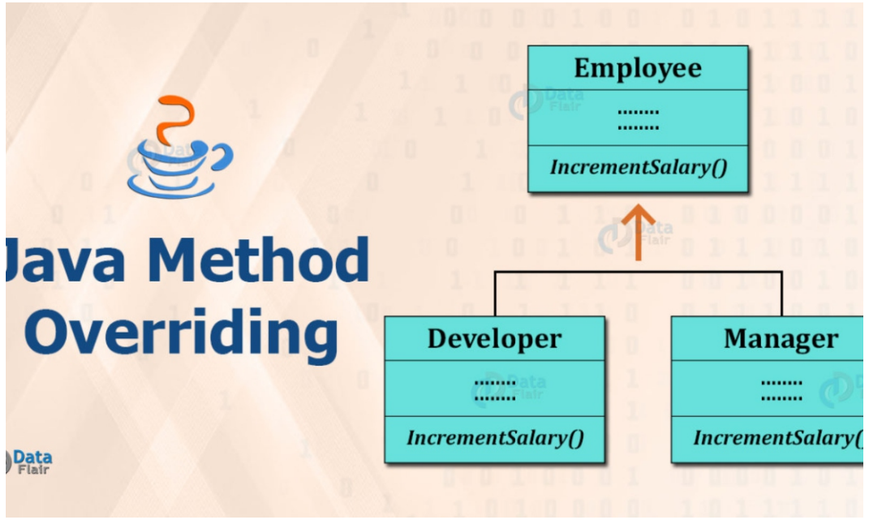
i. What ways can Arrays be passed to Methods
ii. mention how to pass a parameter by value is achieve
iii describe pass parameter by reference

Further Reading

Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures i
Java: Cambridge University Press
Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

References

Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures i
Java: Cambridge University Press
Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.



UNIT 4

Method Overriding

Introduction

In this unit, you will learn about overriding and overridden methods. You will also be introduced to rules guiding overriding of methods and how to resolve issue arising during a call to an overridden method at run time.

Learning Outcomes

At the end of this unit, you should be able to:

- Describe the overriding method;
- Create the overriding method;
- Describe the overridden method; and
- Dist the rules of using the overriding method.

Main Content

I want you to know it is sometimes possible to use the same identifier name for a different method in java. The methods so created have the same signature or parameter. Such a case occurs between subclass and superclass; it is said that the subclass method overrides the superclass method.

Overriding methods

2 mins

SAQ 1 & 2

Overriding is when a method in a subclass need to implement a method in the superclass with the same name, same parameters or signature and same return type as a method in its superclass, then the method in the subclass is said to **override** the method in the superclass.

Try to understand that in inheritance, a subclass inherits all of the public and protected methods of its superclass. However, you should know a subclass needs to change the functionality defined in the superclass; hence overriding technique is employed to redefining an inherited method. You should note that the subclass method overriding the superclass method must have the same number of parameters as the original method and the same return type as the original return type.

Method overriding is used by a subclass to provide a specific implementation of a method already provided in its super class.class A{ //Overridden method public void eat{System.out.println("Father is eating"); }}class B extends A{//Overriding method Let's consider an example to understand this. We have two classes: a subclass B and a superclass A. The B class extends A class. Both the classes have a common method void eat(). B class is giving its implementation to the eat()method or in other words it is overriding the eat() method.

```
class A
{
      //Overridden method
   public void eat()
   {
System.out.println("Father is eating");
   }
}
class B extends A
{
  //Overriding method
```



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

You should also note that the purpose of Method Overriding is clear here. Subclass wants to give its own implementation so that when it calls this method, it prints Boy is eating instead of Father is eating.

Rules for method overriding

2 mins

SAQ 3 & 4

You should carefully note the following rules are to be adhered to when method overriding is to be created.

i. The argument list of the overriding method should be exactly the same as that of the overridden method.
ii. The return type should be the same return type declared in the original overridden method in the superclass.
iii. The access level cannot be more restrictive than the overridden method's access level. For example, if the superclass method is declared public, then the overriding method in the subclass cannot be either private or protected.
iv. Instance methods can be overridden only if the subclass inherits them.
v. A method declared final cannot be overridden.
vi. A method declared static cannot be overridden but can be re-declared.
vii. If a method cannot be inherited, then it cannot be overridden.
viii. A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
ix. A subclass in a different package can only

override the non-final methods declared public or protected.

x. An overriding method can throw any uncheck exceptions, regardless of whether the overridden method throws exceptions or not. However, the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
xi. Constructors cannot be overridden.
xii. When invoking a superclass version of an overridden method, the super keyword is used.

Dynamic Method Dispatch

I want you to know that dynamic method dispatch involves resolving a call to an overridden method at run time, rather than compile time. It worths for you to note that a superclass reference variable can refer to a subclass object. Java uses this fact to resolve calls to overridden methods at run time. When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time of the call. Thus, I want you to know that this determination is made at run time. When different types of objects are referred to, different versions of an overridden method will be called. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed. Therefore, try to have it at the back of your mind that if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed. Here is an example that illustrates dynamic method dispatch:


```
// Dynamic Method Dispatch class A
{
     voidcallme()
     {
     System.out.println("Inside A's callme method");
     }
 }

class B extends A
{
 // override callme()
voidcallme()
     {
     System.out.println("Inside B's callme method");
     }
 }
class C extends A
{ // override callme()
     voidcallme()
     {
     System.out.println("Inside C's callme method");
     }
}
class Dispatch
{
     public static void main(String args[])
     {
          A a = new A(); // object of type A
          B b = new B(); // object of type B
          C c = new C(); // object of type C
```

It is important you note that the output from the program is shown here:
Inside A's call me a method
Inside B's call me a method
Inside C's call me a method
This program creates one superclass called A, and two subclasses of it, called B, and C. Subclasses B and C override call me( ) declared in A. Inside the main( ) method, objects of type A, B, and C are declared. Also, a reference of type A, called r, is declared. The program then, in turn, assigns a reference to each type of object to r and uses that reference to invoke call me( ). As the output shows, the version of calling me( ) executed is determined by the type of object being referred to at the time of the call. Had it been determined by the type of the reference variable, r, you would see three calls to A's call me( ) method.

Summary

I am very sure in this unit, we describe the methods that have the same name and same signature in subclass and superclass. The method that overrides is in the subclass, and it is called overriding method while the method displaced is in the superclass and called the overridden method.
In this unit, you learnt to:

i. describe the overriding method;
ii. create the overriding method
iii. describe the overridden method
iv. list the rules of using the overriding method

Self-Assessment Questions

SAQ

1. explain what you understand by overriding method
2. describe the overridden method
3. state the rules of using the overriding method
4. list rules of overriding method

Tutor Marked Assessment

i. list rules of overriding method
ii. describe the overridden method
iii. explain what you understand by overriding method
iv. state the rules of using the overriding method

ii. describe the overridden method
iii. explain what you understand by overriding method
iv. state the rules of using the overriding method

Further Reading

Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures Java: Cambridge University Press

Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.

Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.

Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

References

Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures i
Java: Cambridge University Press

Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.

Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.

Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

Java Method Overloading

UNIT 5

Method Overloading

Introduction

In this unit, you will learn about overloading a method. You will be introduced to creating and use overloading methods; valid overloading and invalid overloading.

Learning Outcomes

At the end of this unit, you should be able to:

- Identify method signature;
- Create the overloading method;
- Use overloading method;
- Apply the overloading method;
- Identify valid overloading
- Identify invalid overloading

Main Content

In method overloading, I want you to know the methods involve are having the same name in the class. The methods involve are defined in the same class. This makes the codes to be flexible and thus, suitable method invokes appropriate task

Definition

2 mins

SAQ 1 & 2

It is important you note that Method Overloading is a feature that allows a class to have more than one method having the same name but with different argument lists. To overload a method, the argument lists of the methods must differ in either of these three ways.

i. **Several parameters**: The number of parameters of the overloading method differs. For example

add(int, int);

add(int, int, int);

Try to understand that the two methods above differ in the number of parameters. The first method has only one parameter, while the second has two parameters.

```
class DisplayOverloading
{    public void add(int num1, int num2)
    {
     num = num1+num2;
System.out.println('Total '+ " "+num);
    }
    public void add(int num1, int num2, num3)
    {
     intnum;
     num = num1+num2+num3;
System.out.println('Total '+ " "+num);
    }
}
class Sample
{   public static void main(String args[])
   {
DisplayOverloadingobj = new DisplayOverloading();
obj.add(10,5);
obj.add(20,10,5);
   }
}
```


ii. **The data type of parameters**: In this case, the data types of the methods are different.

add(int, int);

add(int, float);

In the above example, I want you to know the first method has type int and the second has type float.

```
class DisplayOverloading2
{public void add(int num1, int num2)
    {
     num = num1+num2;
System.out.println('Total '+ " "+num);
    }
    public void add(int num1, float num2)
    {
     intnum;
     num = num1+num2;
System.out.println('Total '+ " "+num);
    }}class Sample2
{    public static void main(String args[])
    {
        DisplayOverloading2 obj = new DisplayOverloading2();
obj.add(10,20);
obj.add(10,2.5);
    }
}
```

iii. **The sequence of Data type of parameters:** The sequence of types in the parameter list is not the same as in the example below.

add(int, float);

add(float, int);

```
class DisplayOverloading3
{
public void add(float num1, int num2)
    {
     num = num1+num2;
System.out.println('Total '+ " "+num);
    }
public void add(int num1, float num2)
    {
intnum;
num = num1+num2;
System.out.println('Total '+ " "+num);
    }
}

class Sample2
{
    public static void main(String args[])
    {
        DisplayOverloading3 obj = new
       DisplayOverloading3();
obj.add(1.0,20);
obj.add(10,2.5);
    }
}
```

Valid/invalid cases of method

2 mins

SAQ 3 & 4

Sometimes, it is important you note thatoverloading is misused and result in errors which are not detected earlier enough to be debugged.We consider some cases of method overloading

Case 1:

```
intmymethod(int a, int b, float c);

intmymethod(int var1, int var2, float var3);
```

In the case presented above, I want you to know the two methods are the same. Both methods are having the same number, data types and the same sequence of data types. This result in Compile time error.

Case 2:

```
intmymethod(int a, int b);

intmymethod(float var1, float var2);
```

try to understand this above is a valid case of overloading. The two methods have data types of different arguments.

Case 3:

```
intmymethod(int a, int b);

intmymethod(intnum);
```

Result: Perfectly fine. Valid case of overloading. Here a number of arguments are different.

Case 4:

```
floatmymethod(int a, float b);

floatmymethod(float var1, int var2);
```

Result: Perfectly fine. Valid case of overloading. I want you to be aware that the sequence of the data types of parameters are different, the first method is having (int, float) and second is having (float, int).

Case 5:

```
intmymethod(int a, int b);

floatmymethod(int var1, int var2);
```

Result: Compile time error. Argument lists are exactly the same. It will interest you to know even though the return type of methods are different, it is not a valid case. Since return type of method doesn't matter while overloading a method.

01 | Method Overloading and Overriding in java
source| Edukare

Overloading vs Overriding in Java

1 min

I want us to carefully look into this;

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

7. The return type of method does not matter in case of method overloading; it can be same or different. However, in case of method overriding the overriding method can have a more specific return type (refer to this).
8. The argument list should be different while doing method overloading. The argument list should be the same in method Overriding.

Summary

At the end of this unit, you have learnt:

i. methodoverloading;
ii. Valid and invalid overloading method; and
iii. different between overloading and overriding methods.

Self-Assessment Questions

1. Explain the components of method signature;
2. create a method to overload an existing method;
3. give two examples of valid overloading
4. give two examples of invalid overloading

Tutor Marked Assessment

i. explain what method signature signifies;
ii. create an overloading method to add two integer and two double numbers;
iii. What is method overloading?
iv. Is it permissible to define two methods that have the same name but different parameter types?
v. Is it permissible to define two methods in a class that have identical method names and parameter lists but different return value types or different modifiers?

Further Reading

- Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.
- Eckel, B. (2003). Thinking in JAVA: Prentice Hall Professional.
- Keogh, J., Giannini, M., & Rinaldi, W. (2004). OOP demystified: McGraw-Hill, Inc.
- Lambert, K., & Osborne, M. (2006). Fundamentals of Java: AP* Computer Science Essentials for the A & AB Exams: Course Technology Press.
- Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.

References

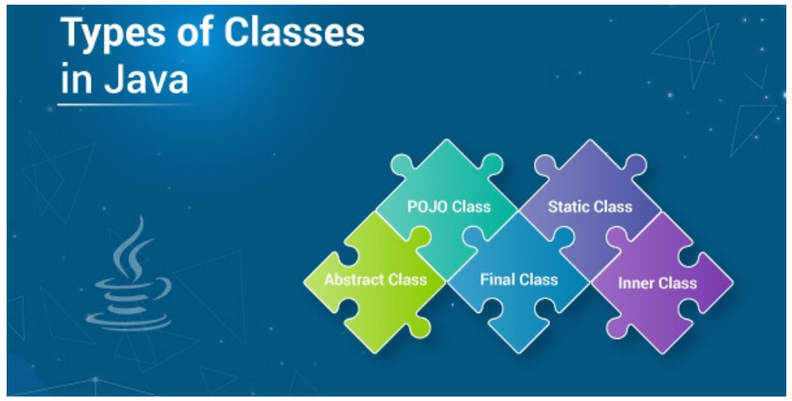
- Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.
- Eckel, B. (2003). Thinking in JAVA: Prentice Hall Professional.
- Keogh, J., Giannini, M., & Rinaldi, W. (2004). OOP demystified: McGraw-Hill, Inc.
- Lambert, K., & Osborne, M. (2006). Fundamentals of Java: AP* Computer Science Essentials for the A & AB Exams: Course Technology Press.
- Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.

Module 3
Class and Object

Units

Class

Object



UNIT 1

Class

Introduction

This unit introduces you to the class and the components of a class. You will learn types of classes and how to create a class.

Learning Outcomes

At the end of this unit, you should be able to:

i. Define class;
ii. List types of class;
iii. Declare a class;
iv. Initialize a class; and
v. Insatiate a class.

Main Content

Defining Class

1 min

SAQ 1

I want you to note that a class is a user blueprint or prototype for creating objects as needed. It will interest you to know it consists of the set of properties and methods that are required by a particular type of objects. Try to understand that in Java, class uses variables to define data fields and methods to define actions.I want you to also know a class provides methods of a special function, known as constructors, which are invoked tocreate a new object. Let me say a constructor can perform any action, however, constructors are designed toperform initializing actions, such as initializing the data fields of objects.

To declare a class, it is important you note the following components is required in the order listed.

i. Modifiers: This is to control accessibility of the class can be public or has default access.
ii. Class name: This is the name to identify the class. It should begin with an initial letter (capitalized by convention).
iii. Superclass (if any): The name of the class's superclass if any, preceded by the keyword extends. A class can only extend one superclass.
iv. Interfaces (if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
v. Body: The class body surrounded by braces, {}.
vi. Constructors are used for initializing new objects. Fields are variables that provides the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.


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

Try to understand that this example includes modifier **public**, name **dog** and body in a brace bracket. It will also interest you to know the class has three methods. These are **barking**(), **hungry**() and **sleeping**(). This class does not include superclass and interface. However, I want you to know the constructor is implied. Which was laterdescribed in Module 2.

Types of Java Class

3 mins

SAQ 2 & 3

Based on the location of its declaration class, I want you to bear in mind that it is divided into outer and inner or nested classes.

Outer Class

Outer class is a class which has another class defined in its body.

Inner class

You should also understand that there are two types of inner or nested classes: static and non-static. A static nested class is one that has the static modifier applied to it during definition.

Non-Static Inner Class or Nested Class

It good you know an inner class is a non-static nested class that can access the variables and methods of its outer class and may refer to them directly in the same way that other non-static members of the outer class can. I want you to also know an inner class, or nested class, is defined within the scope of another class. It may be used just like any regular class. Another important thing I will want you to undersand is the definition of a class which is definedas an inner class if it is used by its outer class. An inner class has the following features:

For example

```
public class Test {
                    ...
                    }

public class A{
```

Example 1: Not Inner Class

```
public class Test {
                    ...
                    public class A{
                    ........
                              }

}
```

Example 2: Inner Class

In the above example, **Example 1** is not an inner class because class A is not enclosed in the Class Test. This is two outer classes. **Example 2** has class A embedded in the class Test hence it is an inner class.

An inner class has its own methods and variables declare in its own body definition. An inner class can be use like any other class. This is illustrated below.

```
// OuterClass.java: inner class demo
public class OuterClass {
private int data;
/** A method in the outer class  namedoutermethod*/
                    public void outermethod () {
                    // codes to execute here by outer method
                    }
          // An inner class
                    class InnerClass {
     /** A method in the inner class */
                    public void innermethod() {
               // codes to execute here by inner method
// defined in its outer class
                              }
                    }
          }
```

I want you to bear in mind that an inner class has the following features:

i. An inner class is compiled into a class named OuterClassName$InnerClass-Name.class.For example, if class A is inner class of class Test, at compilation time it will be compiled as Test$A.class.

ii. An inner class can reference the data and methods of its outer class in which itnests, there is no need to pass the

reference of an object of the outer class to the constructorof the inner class. For this reason, inner classes can make programs simple and concise.

iii. An inner class can be defined with a visibility modifier subject to the same visibility rules applied to a member of the class.

iv. An inner class can be defined static. A static inner class can be accessed using the outer class name. A static inner class cannot access nonstatic members of the outer class.

v. Objects of an inner class are often created in the outer class. It is possible to create an object of an inner class from another class. If the inner class is non static, first create an instance of the outer class, then use the following syntax to create an object for the inner class use the syntax:

```
OuterClassouter Object= new innerClass();

OuterClass.InnerClassinnerObject = outerObject.newinnerClass();
```

For example, if class A is an inner class of class Test, then to create an object of class A, you use

```
Test.AinnerObj = testObj new A();
```

vi. If the inner class is static, use the following syntax to create an object for it:

```
OuterClass.InnerClassinnerObject = new OuterClass.InnerClass();
```

Static Nested classes

Let me tell you that if a nested class is declared with a modifier **static**, then it's a static inner class. Otherwise, it is simply a non-static inner class. You should also have in mind that even though syntactically the difference is just a single keyword static, semantically there is a huge difference between these kinds of nested classes. Try to understand that inner class instances are bound to the enclosing class, and therefore they have access to its members. You should be aware of this issue when selecting whether to make a nested class be an inner or not.

```
Outer outer = new Outer();
```

```
Outer.Inner inner = outer.newinner();
```

Here are a few vital points I will want you to remember about static nested classes:

i. They can have all types of access modifiers in their declaration
ii. They only have access to static members in the enclosing class
iii. As with static members, these belong to the enclosing class, and not to an instance of the class

Local classes

Try to understand local classes are a special type of inner classes – in which the class is defined inside a method or scope block. The local class has the following features:

i. They cannot have access modifiers in their declaration
ii. They have access to both static and non-static members in the enclosing context
iii. They can only define instance members

```
public class NewEnclosing {

   void outerrun() {
      class Local {

         void localrun() {
            // method implementation
         }
      }
      Local local = new Local();
local.run();
   }

      public void test() {
NewEnclosingnewEnclosing = new NewEnclosing();
newEnclosing.run();
   }
}
```

The local class is created inside the run() method of class NewEnclosing

class instance and instantiation

2 mins

SAQ 4

To understand class instance and method instantiation, you need to first understand declaration and definition of variables and methods.

Declaration

When the method is declared without body content or a variable declared with no value assigned to it is called method or variable declaration respectively.


```
public abstract class Student2 {

   // member variables or fields
intstudentRollNo;
   String studentName;
intstudentAge;

   // abstract method
   public abstract void getStudentDetails();
}
```

In the above example, I want you to know variables studentRollNo, student Name and student Age were declared. Similarly, the method gets Student Details() was also declared without brace brackets.

Initialization

I want you to understand that Initialization or definition is when a declared member variable is assigned some values; then the variable is said to be defined or initialized. Also, it important you know a method is defined with body having codes inside open and close braces brackets; then the method is said to be defined or method definition as against the abstract method, which has no definition.

```
public class Student {

    // member variables or fields
intstudentRollNo = 1001;
    String studentName = "Bob";
intstudentAge = 35;

    // instance method of a class
    public void getStudentResult() {
        // method definition goes here
    }
}
```

Carefully look into the above example, I want you to know variables studentRollNo, student Name and student Age were defined with values 1001, Bob and 35, respectively. Similarly, method getStudentResult() was also declared with brace brackets holding the necessary codes.

Instantiation

When a blueprint or full-fledged class is defined with its member variables and method definition, then you should know we need to create or instantiate an object; to access all possible variables and methods of the class. Instantiate is the correct word to describe the process instead of create, which is used for laymen understanding of the terms.

```
public class Student {

    // member variables or fields
intstudentRollNo = 1001;
    String studentName = "Bob";
intstudentAge = 35;
    // instance method of a class
    public void getStudentResult() {
        // method definition goes here
System.out.println("Sampublic class Student {
    // member variables or fields
intstudentRollNo = 1001;
    String studentName = "Pup";
intstudentAge = 35;

    // instance method of a class
    public void getStudentResult() {

        // method definition goes here

System.out.println("Sample method "
                + "invocation for Student with details: \n");
System.out.println("Roll No.:- " + studentRollNo);
System.out.println("Name:- " + studentName);
System.out.println("Age:- " + studentAge);
    }
    // main() method
    public static void main(String args[]) {
        // instantiate or create an Object of type Student
        Student student = new Student();
        // invoking method using newly instantiated Student Object
student.getStudentResult();
    }
}        System.out.println("Roll No.:- " + studentRollNo);
System.out.println("Name:- " + studentName);
System.out.println("Age:- " + studentAge);
                        }
```

The instance of a class

Whenever we instantiate an object of a class, then that particular reference variable is referred to as an instance of that class. When an object is instantiated, there is always a new operator associated with it for the creation of a new object. Using the instance of an operator, one can check or verify whether particular reference variable is an instance of a class or NOT


```
public class Student {

    // member variables or fields
intstudentRollNo = 1001;
    String studentName = "Bob";
intstudentAge = 35;

    // instance method of a class
    public void getStudentResult() {

        // method definition goes here
    }

    // main() method
    public static void main(String args[]) {

        // instantiate or create an Object of type Student
        Student student = new Student();

        // verifying using instanceof operator
        if(student instanceof Student) {
System.out.println("It is an instance "



   // main() method
   public static void main(String args[]) {
       // instantiate or create an Object of type Student
       Student student = new Student();

       // invoking method using newly instantiated Student Object
student.getStudentResult();
   }
}
```

Summary

In this unit, you have learnt:

i. Class and types of class;
ii. How to declare and initialize variables and methods; and
iii. How to instantiate a class.

Self-Assessment Questions

1. State the components of a class.
2. List types of class.
3. Describe the features of local class.
4. State the steps to insatiate a class.

Tutor Marked Assessment

i. define the components of a class
ii. enumerate types of class
iii. Describe each type of class enumerated.
iv. Describe the features of local class
v. State the steps to insatiate a class

Further Reading

- Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.
- Eckel, B. (2003). Thinking in JAVA: Prentice Hall Professional.
- Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.

References

Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.

Eckel, B. (2003). Thinking in JAVA: Prentice Hall Professional.

Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.

Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.

UNIT 2

Class and Object

Introduction

In this unit, you will be learning about an object and how to create an object. You will also learn the different types of object initialization. Lastly, you will be introduced to creating multiple objects.

Learning Outcomes

At the end of this unit, you should be able to:

i. define an Object;
ii. Initialize an object;
iii. describe Anonymous Objects; and
iv. Initialize multiple objects.

Main Content

Defining Object

1 min

SAQ 1

I want you to know a java object is a combination of data and procedures working on the available data. An object has a state and behaviour. It will interest you to know the state of an object is stored in fields (variables), while methods (functions) display the object's behaviour. Objects are created from templates known as classes.

It is important for you to note that there are three steps to creating a Java object:

i. Declaration of the object;
ii. Instantiation of the object; and
iii. Initialization of the object.

I want you to understand that when a Java object is declared, a name is associated with that object. The object is instantiated so that memory space can be allocated. Initialization is the process of assigning a proper initial value to this allocated space. The properties of Java objects include:

I want you to be aware that one can only interact with the object through its methods. Hence, internal details are hidden. When you are coding, an existing object may be reused. When a particular object hinders a program's operation, that object can be easily removed and replaced.

Another important thing of note is A new object t from the class "tree" which is created using the following syntax:

```
Tree t = new Tree();
```

Object Initialization

2 mins

SAQ 2

I want you to know initializing is done by a java constructor; memory is allocated, and a reference is given to that memory. I want you to also bear in mind that there are three ways by which an object can be initialized in Java. These are:

i. By reference variable;
ii. By method; and
iii. By constructor.

Initialization through reference

I want you to know that initializing an object means storing data into the object. I want us to see a simple example where we are going to initialize the object through a reference variable.


```
class Student{
int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
System.out.println(s1.id+" "+s1.name);//printing members with a
white space
 }
}
```

Initialization through method

You should be aware that this uses a method to initialize object of a class. As an example, the illustrates object initialization through method.

```
class Student{
introllno;
 String name;
 void insertRecord(int r, String n){
rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
 Student s1=new Student();
 Student s2=new Student();
 s1.insertRecord(111,"Karan");
 s2.insertRecord(222,"Aryan");
 s1.displayInformation();
 s2.displayInformation();
}
```

In this example, we are creating two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

Initialization through a constructor

It is important you know classes provide constructor methods to initialize a new object of that type. You should be aware that In a class declaration, constructors can be distinguished from other methods because they have the same name as the class and have no return type.

For example,

Constructors are used in constructing objects. To construct an object from a class, you have to invoke a constructor of the class using the new operator, as follows:

newClassName();

When you see constructor such as the one shown above that takes no argument, they are known as the default constructor. The class has at least one constructor, the default constructor. However, classes can have multiple constructors, all with the same name but with a different number or type of arguments.

Creating multiple objects

1 min

SAQ 3 & 4

You can also create multiple objects and store information in it through the reference variable as given below:

```
class Student{
int id;
 String name;
}
class TestStudent3{
     public static void main(String args[]){
          //Creating objects
          Student s1=new Student();
Student s2=new Student();
          //Initializing objects
          s1.id=101;
s1.name="Sonoo";
          s2.id=102;
          s2.name="Amit";
           //Printing data
          System.out.println(s1.id+" "+s1.name);
          System.out.println(s2.id+" "+s2.name);
 }
}
```

Anonymous Objects

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only. If you have to use an object only once, an anonymous object is a good approach. For example:
Let's see the full example of an anonymous object in Java.

```
class Calculation{
 void fact(int  n){
int fact=1;
for(inti=1;i<=n;i++){
   fact=fact*i;
  }
System.out.println("factorial is "+fact);
}
public static void main(String args[]){
 new Calcu  lation().fact(5);//calling method with anonymous
object
}
}
```


Summary

In this unit, you have learnt about:

i. object;
ii. initializing objects;
iii. creating multiple object; and
iv. anonymous objects.

Self-Assessment Questions

SAQ

1. Define an Object
2. Initialize the object you just defined
3. describe anonymous Object
4. Initialize multiple objects

Tutor Marked Assessment

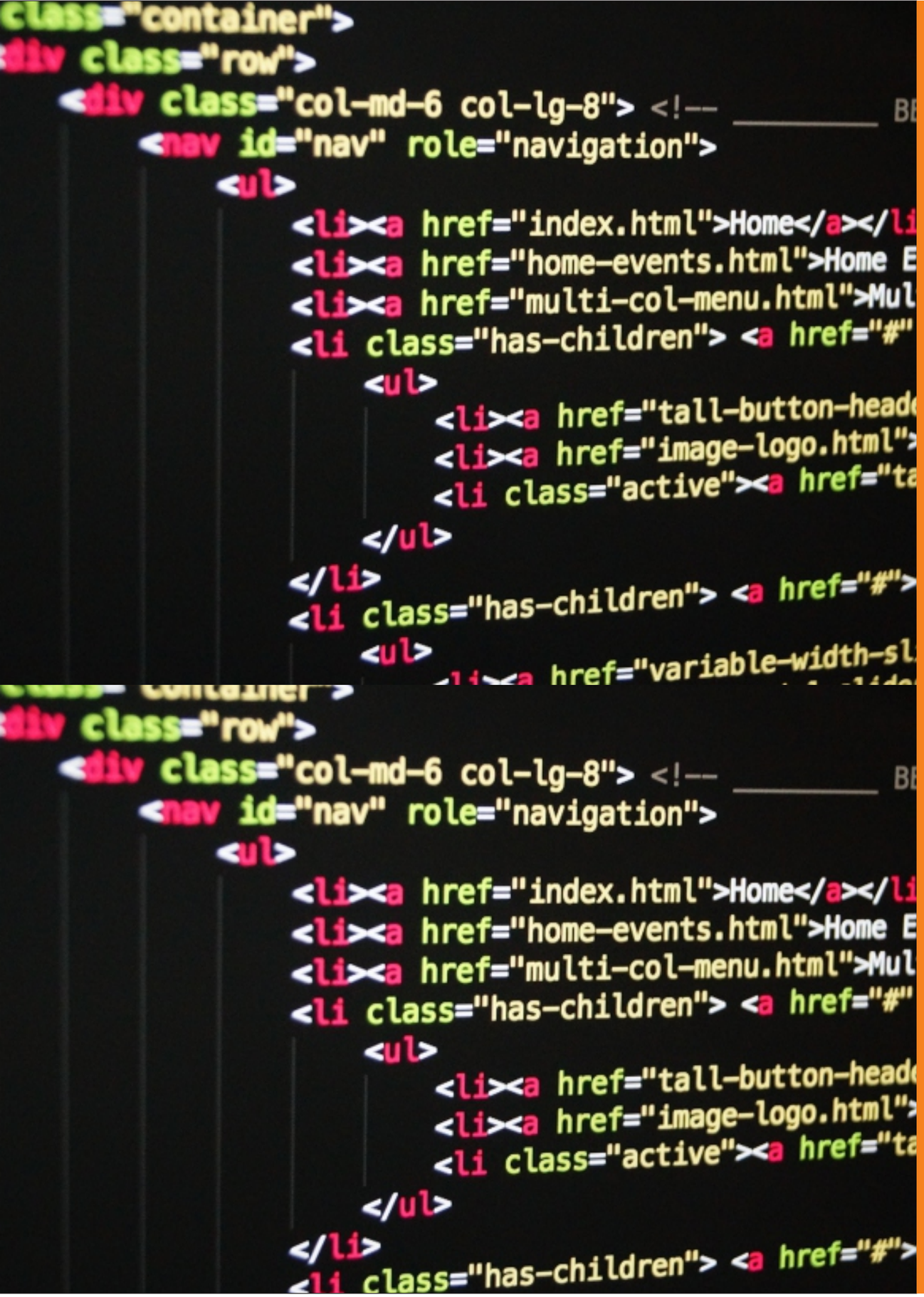
i. Describe an Object
ii. Initialize the object you just described
iii. Initialize multiple objects

Further Reading

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structuresJava: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

References

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures i
  Java: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.



Module 4

Basic concepts of Object-Oriented Programming

Units

Data Abstraction

Data Hiding

Encapsulation

Inheritance

Polymorphism

Object Modeling

UNIT 1

Data Abstraction

Introduction

Welcome to this unit where you will learn about data abstraction and how it is achieved using java. You will also be introduced to an abstract class, access modifier and interface. Finally, you will be able to identify differences between abstract class and interface.

Learning Outcomes

At the end of this unit, you should be able to:

i. Describe data abstraction
ii. Explain abstract class
iii. Explain Abstract using Access Modifiers
iv. Create an interface
v. Distinguish an Abstract Class from an Interface

Main Content

Definition

1 min

The term “Abstraction” hererefers to the act of representing essential features without including the background details. It is important you note that the internal details need not be known to the user, and what is happening inside is hidden completely to the user. Thus, abstraction knows the essential things to operate without knowing the background working details of the operation.

Data abstraction specifies what you do to data but not how you do it. It focuses on what operations on the data are to be provided rather than implementation on the data.

Consider the method

```
public static double random()
{

}
```

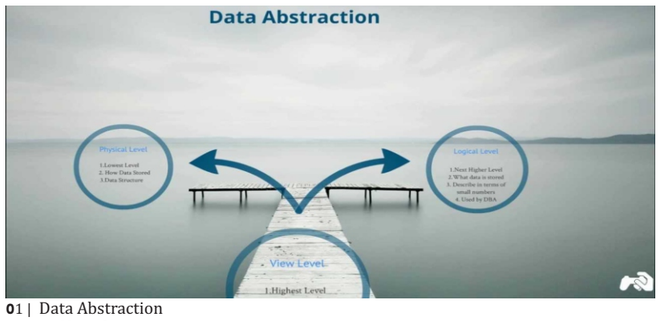
The user of this method does not need to know how the method generates random number, which is hidden to the user, but a call to the method returns a random number between 0 and 1 for its caller. I want you to try to understand that this is just what the user needs.

You should be aware that there are two ways Java implements abstraction:

i) abstract classes

ii) interfaces.

With abstract classes, we achieve partial abstraction, while interfaces provide total abstraction. Let us discuss more on Abstraction Data Type ADTS.



01 | Data Abstraction
source| Youtube.com

Abstract Class

1 min

SAQ 1

It is important you note an abstract class contains the keyword 'abstract' preceding the class itself. By this, what I meant is that the class cannot instantiate an object. It can contain abstract methods as well as other methods. However, it will interest you to know that if a class contains an abstract method, then itself must be declared abstract. A subclass of an abstract class must provide implementations for abstract methods in its parent class; otherwise, the subclass itself must also be declared abstract.

Another thing I will want you to know is that Java uses the technique of abstraction to define classes with the list of abstract attributes and methods to operate on these attributes. The attributes are called *data members* and they are used to store data and the functions called *function members* which are used to manipulate those data members.

```
abstract class GObject
{
    // declare fields
    // declare nonabstract methods
    abstract void move();
}
```

The java code segment above illustrates the abstract class declaration

It is important for you to note that:

i. An abstract class is declared using the “**abstract**” keyword;
ii. A class with an abstract method is an abstract class;
iii. A normal class cannot have an abstract method;
iv. An abstract class can have a non-abstract method as well; and
v. An abstract class may not contain an abstract method.

Abstract using Access Modifiers

2 mins

SAQ 2

Let me tell you that Access Modifiers which is also called Visibility Modifiers are keywords that restrict access to classes, variables and methods in object-oriented programming. It will interest you to know that these Modifiers determine whether a variable or a method in a class can be used or invoked by another method in another class or sub-class. Access Modifiers are used to regulate access. Java uses access modifiers to implement abstraction.

In Java, it is very important for you to note methods and data members of a class can have one of the following four access modifiers. The access modifiers are listed for you to critically look into according to their restrictiveness order.

- **private**

Private Modifiers have the most restrictive level of accessibility. Methods and variables declared as private are accessed within the same class to which the methods and variables belong. Private methods and variables are not visible to subclasses and cannot be inherited by subclasses. Using Private Modifier, encapsulation can be achieved and data hidden from the outside world.

- **default** (when no access modifier is specified)

When no access modifier is set, it follows the default accessibility level. There is no default modifier keyword. Classes, variables, and methods can be default accessed. Using default modifier, classes, methods, or variables can be accessed within the same package but not from outside this package.

- **protected**

Methods and fields declared as protected can only be accessed by the subclasses in other packages or any class within the package of the protected members' class. The protected access modifier cannot be applied to class and interfaces.

- **public**

Public Modifiers provide the highest level of accessibility. Classes, methods, and variables declared as public can be accessed from any class whether these classes are in the same package or another package.

However, I want you to know the classes and interfaces can have only two access modifiers when declared outside any other class.

- public
- default (when no access modifier is specified)

Try to understand members declared as **private** in a class, can be accessed only from within the class. They are not allowed to be accessed from any part of the code outside the class.

Members declared as **public** in a class can be accessed from anywhere in the program.

You should know that we can easily implement abstraction

using the above two features provided by access modifier. Suppose, the members that define the internal implementation can be marked as private in a class. And the important information needed to be given to the outside world can be marked as public. And these public members can access the private members as they are inside the class.

Also, you should note there are non-access modifiers used in java. These are abstract, final and static. Abstract applies to class and methods only.

Also to note for you is that final non-access modifier applies to class, method and variable while static is used with methods and variables. A final class is declared with the keyword **final,** and other classes cannot extend such class. A final class cannot be inherited. Similarly, the final method is defined with the final keyword and cannot be overridden by subclasses. A final variable is just like a constant whose value cannot be changed.

It will interest you to know static variables are also called class variable while the static method is the class method. The reason being that they can be accessed by class name and doesn't need any object.

Other non-access modifiers are outside the scope of this course. These are:

- synchronized;
- native;
- transient;
- volatile; and
- strictfp.

Interface

1 min

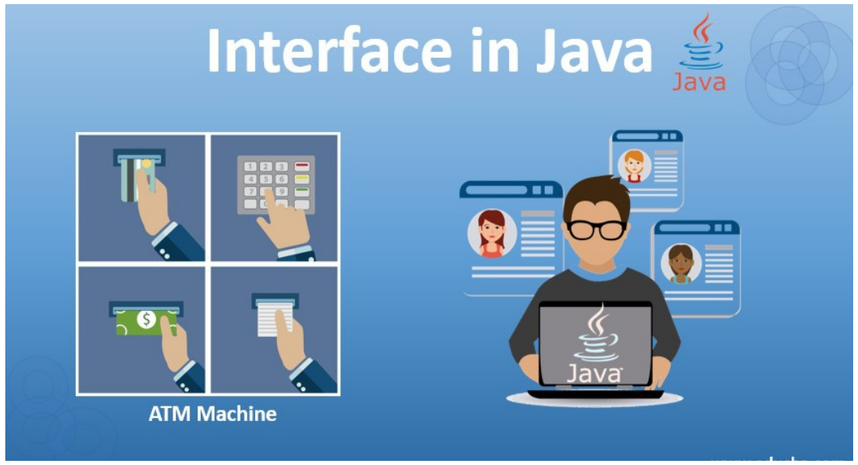
SAQ 3

It is important you know an interface is a blueprint or a standard of a class which allows the creation of one or more methods in a class with no corresponding implementation. You should know this means the methods declared in an interface are abstract methods. The implementation of the methods is presented by the subclass class that implements the interface. The interface is invoked with the keyword **extends**.

Have it at the back of your mind that Data variables in an interface are declared **static** and **final**. These are non-accessible modifiers indicating that only one copy of the data is available and not modifiable. Data variables in an interface are declared as constants and are made available to methods for use. You should be aware that all data variables are implicitly declared as static and final in an interface definition; hence there is no need for you to precede the data variables with these keywords.

I want you to know methods in an interface are abstract methods only, and every class that implements the interface must provide an implementation for the methods. You should also know an interface does not have to declare its methods abstract using abstract keyword explicitly. Similarly, interface methods are always public, and the access modifier's general keyword is not required since it is implied in the interface declaration.

Let us consider how an interface is declared and implemented in Java.



01 | Interfaces in java
source| EduCBA


```
interface Animal {
  public void animalSound(); // interface method (does not
have a body, abstract )
  public void sleep(); // another interface method (does
not have a body)
}

A class Pig implements the Aminal interface as follows

// Pig "implements" the Animal interface
class Pig implements Animal
{
          public void animalSound()
     {
     // The body of animalSound() method  is provided here
in the Pig class
     System.out.println("The pig sounds: wee wee");

 }
 public void sleep()
{
      // The body of sleep() is provided here
     System.out.println("The pig sleeps");
     }
}

class MyMainClass {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Abstract Class Vs Interface

1 min

SAQ 4

As we have earlier discussed, **Abstraction** hides the internal implementation of the feature and only shows the functionality to the users. It's worthy of note for you that both abstract class and interface are used for abstraction. The following are some differences that exist between them.

	Abstract class	Interface
Type of methods	An abstract cla ss can have abstract and non-abstract methods	An interface can have only abstract methods
Final Variables	An abstract class may contain non - final variables	Variables declared in a Java interface are by default final
Type of variables	An abstract class can have final, non - final, static and non-static variables	The interface has only static and final variables.
Implementation	An abstract class can provide the implementation of an interface	— — -
Inheritance vs Abstraction	An abstract class can be extended using the keyword “extends	A Java interface can be implemented using the keyword “implements.”
Multiple implementations	An abstract class can extend another Java class and implement multiple Java interfaces	An interface can extend another Java interface only
Accessibility of Data Members	A Java abstract class can have class members like private, protected, etc	Members of a Java interface are public by default

Summary

In this unit, I am sure you have learnt:

i. data abstraction;
ii. abstract class;
iii. interface;
iv. access modifiers;
v. the differences between abstract class and interface.

Self-Assessment Questions

SAQ

I. Describe data abstraction.
2. Explain abstract class.
3. What is an Interface?
4. List the differences between Abstract Class and Interface.

Tutor Marked Assessment

i. Describe data abstraction
ii. Explain abstract class
iii. How can Access Modifiers be used to achieve Abstract?
iv. List the differences between Abstract Class and Interface

Further Reading

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structuresJava: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

References

Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structuresJava: Cambridge University Press

Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub. Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.

Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

UNIT 2

Data Hiding

Introduction

This unit introduces you to data hiding and how it is achieved using java. you will also learn the differences between data hiding and abstraction. You will also learn how to implement data hiding using java.

Learning Outcomes

At the end of this unit, you should be able to:

- Explain data hiding.
- Distinguish between data hiding and abstraction
- Implement data hiding

Main Content

Definition

2mins

SAQ 1 & 2

Data hiding involves protection of the data from the components of the program that does not require the data. Also, you should know Data hiding or information hiding is the isolation of the data from direct access by the program. To implement data hiding, you should understand encapsulation is used where data and function of a class are protected from unauthorized access. In contrast, when the data and function are encased into a single unit, it is known as encapsulation. Hence, bear in mind that the data hiding assists in achieving encapsulation. The functional details of an object can be handled through access specifiers. Using the data hiding concepts, the data and function in a class are made private so that it will not be accessed falsely by the functions outside the class and protected from accidental alteration.

I want you to carefully note the Differences between Data Hiding and Abstraction

i. Abstraction shows the relevant information and rejects the non-essential details. On the other hand, data hiding is used to hide the data from the parts of the program.
ii. The prior purpose of abstraction is to hide the complex implementation detail of the program or software. On the contrary, data hiding is implemented to attain encapsulation.
iii. Abstraction is used in class to create a new user-defined data type. As against, in classes, data hiding is used to make the data private.
iv. The abstraction concentrates on the observable behaviour of the data, whereas data hiding restricts or allows the use of data within a capsule.

Implementing Data Hiding

1 min

SAQ 3

Try to understand Data Hiding is one of the important concepts of Object-Oriented Programming (OOP) that allows developers to protect their private data and hide implementation details from intrusion by other users. However, certain data need to be made available to others; hence, such data require to be accessed in some way using public 'get' and 'set' methods for those private variables in the class. In this unit, basic data hiding techniques in Java is demonstrated by implementing a simple program.


```
package DataHidingInJava;

public class Person {
//  public - keeping a variable public is a violation of data
hiding 'rules'
    public float height;
 //  private - This is not visible outside the class, hence
hides the data
    private float weight;
//  Here are 'get' and 'set' methods for the
//    private variable declared above.
// 'get' and 'set' methods are also called 'getters' and
'setters' respectively

 public float GetWeight() {
    return weight;
}

    public float SetWeight(float weight){
        if (weight >= 0)
this.weight = weight;
        return this.weight;
    }

}// class end
```

Also, its vital you note Data hiding is an extreme version of encapsulation where you not only want the user to access the data members, but you also do not want them to be able to see what they are. It is important for you to note that:

i. an outside person can't access our internal data directly, or our internal data should not go out directly; this OOP feature is nothing but data hiding;
ii. by declaring Data Member (Variable) as Private, we can achieve data hiding;
iii. Data hiding is also known as information hiding;
iv. the main advantage of data-hiding is security. Therefore, it is highly recommended to declare data member (variable) as private; and
v. it just provides a way to protect your data from the outside world. What it means is, suppose you make an instance variable public, then anyone can change its state. But if you make instance variable private/protected then you are actually restricting outside entities from making changes to it.

Summary

In this unit, you have learnt:

i. data hiding;
ii. differences between data hiding and abstraction; and
iii. data hiding implementation.

Self-Assessment Questions

1. Explain data hiding
2. State the differences between data hiding and abstraction
3. Using java, implement data hiding

Tutor Marked Assessment

i. Describe data hiding
ii. State the differences between data hiding and abstraction
iii. Implement data hiding with java.

Further Reading

- Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.
- Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.

References

- Arnow, D. M., Weiss, G., & Dexter, S. (2000). Introduction to programming using java: an object-oriented approach: Addison-Wesley.
- Samanta, D. (2006). Object-oriented Programming with C++ and Java: PHI Learning Pvt. Ltd.
- Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM: McGraw-Hill Incorporated.

UNIT 3

Encapsulation

Introduction

In this unit, you will learn the concept of encapsulation. You will be introduced to the implementation of encapsulation using java.

Learning Outcomes

At the end of this unit, you should be able to:

- Explain the concept of encapsulation
- Demonstrate encapsulation using java

Main Content

Introduction

1 min

SAQ 1 & 2

Its good you note Encapsulation is the technique of making the fields and the codes in a class private and providing access to the fields through public methods. You should also understand this is to keep both fields and codes from interference and misuse of the users. A field declared as private cannot be accessed by anyone outside the class, hence hiding the fields within the class. In another way, encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. An interface tightly controls access to the data and code. The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code.

```
public class Employee
        {
                private String name;
                public String getName()
        {
                return name;
                }
                public void setName(String name)
        {
                this.name = name;
                }
                public static void main(String[] args)
        {
                }
        }
```

Let us look at the code below to get a better understanding of encapsulation:

I want us to consider the above code. A class Employee was created with a private variable **name.** Two methods of getName and setName were created such that through the variable, name of the employee class can be accessed. Through these methods, any class wishing to access the name variable has to do it using these methods.

Benefits of Encapsulation

1 min

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

I want you to know Encapsulated code is more flexible and easier to change with new requirements. By providing only getter and setter methods access, you can make the class read-only.

Try to understand Encapsulation in Java makes unit testing easy. A class can have total control over what is stored in its fields. Suppose you want to set the value of marks field, i.e. marks should be a positive value, then you can write the logic of positive value in the setter method.

Have it at the back of your mind that Encapsulation also helps to write immutable class in Java which are a good choice in multi-threading environments. Encapsulation allows you to change one part of code without affecting other parts of code.

Summary

In this unit, we have discussed the following:

i. Encapsulation
ii. Benefit of encapsulation

Self-Assessment Questions

1. Explain the concept of encapsulation
2. Achieve encapsulation using java

Tutor Marked Assessment

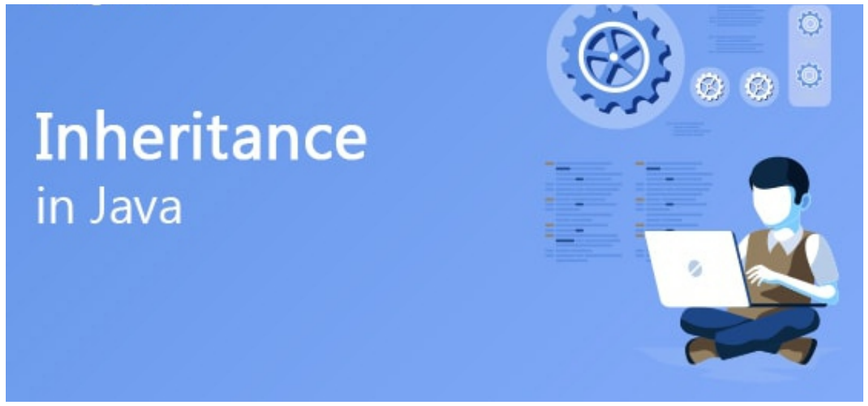
i. Describe the concept of encapsulation.
ii. Use java to achieve encapsulation.

Further Reading

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures Java: Cambridge University Press
- Litvin, M., & Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

References

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures i
  Java: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.



UNIT 4

Inheritance

Introduction

In this section, you will learn the concept of inheritance and its types.

Learning Outcomes

At the end of this unit, you should be able to:

- explain the concept of inheritance;
- list types of inheritance; and
- demonstrate inheritance using java.

Definition

3 mins

SAQ 1 & 2

I want you to know that Inheritance enables new classes to receive (or inherit) the properties and methods of existing classes. **Inheritance** in **object-oriented programming** enables new objects to take on the properties of existing objects. A class that is used as the basis for **inheritance** is called a superclass or base class. A class that **inherits** from a superclass is called a subclass or derived class. In Java, classes may **inherit** or acquire the properties and methods of other classes.

Types of inheritance

I want you to bear in mind that Object-Oriented Programming supports six different types of inheritance, as listed below.

i. Single inheritance
ii. Multi-level inheritance
iii. Multiple inheritance
iv. Multipath inheritance
v. Hierarchical Inheritance
vi. Hybrid Inheritance

i. Single inheritance

I want you to understand that in this inheritance, a subclass is created from a superclass. The subclass directly inherits the superclass. Figure 4.4.1 below illustrates single inheritance.

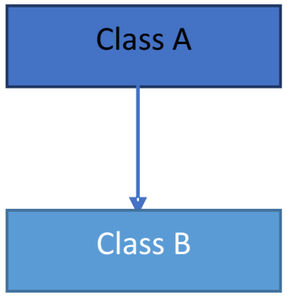


Figure 4.4.1: Single Inheritance


```
Single inheritance syntax
//Base Class
class A
{
public void fooA()
{
//TO DO:
}
}

//Derived Class
class B extend A
{
public void fooB()
{
//TO DO:
}
}
```

In the given example, I want you to have it at the back of your mind that Class A is the superclass, and Class B is the subclass class since Class B inherits the features and behaviour of the parent class A.

ii. Multi-level inheritance

In this inheritance, I want you to be aware thata subclass is created from another subclass.

In figure 4.4.2, it will interest you to note class C inherits the properties and behaviour of class B and class B inherits the properties and behaviour of class A. So, here A is the parent class of B and class B is the parent class of C. Therefore, class C implicitly inherits the properties and behaviour of class A along with properties and behaviour of Class B, hence a multilevel of

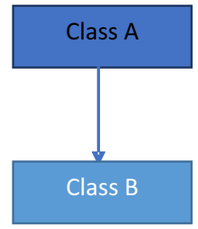
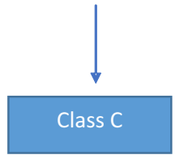


Figure 4.4.2: Multi Level Inheritance



The syntax for Multi-level Inheritance

```
//Base Class
class A
{
 public void fooA()
 {
 //TO DO:
 }
}

//Derived Class
class B extend A
{
 public void fooB()
 {
 //TO DO:
 }
}

//Derived Class
class C extend B
{
 public void fooC()
 {
 //TO DO:
 }
}
```

iii. Multiple inheritances

This is another inheritance, I want you to have it at the back of your mind that in this inheritance, a subclass is created from more than one superclass. Try to understand that this inheritance is not supported by

Java Language and other languages like dotNET Languages etc.

In the given example, it is important for you to note that class C inherits the properties and behaviour of class B and class A at the same level. So, Class A and Class B are the parent classes for Class C.

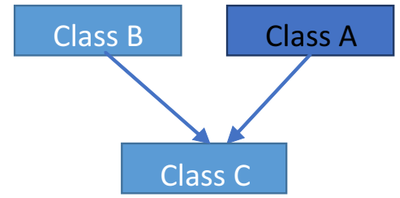


Figure 4.4.3: Multiple Inheritance

Another important thing I want you to know is that Java doesn't allow multiple inheritances as a result of ambiguity associated with it. Example of such a problem is the diamond problem that occurs in multiple inheritances.Figure 4.4.3. shows multiple inheritances as Class C extends both classes A & B. Now suppose a method in class A overrides a class B method in their own way since C is extending both A and B, if C wants to use the same method which method would it called? (the overridden method of B or the overridden method of A). it will be good if you can understand that this ambiguity is the main reason why Java doesn't support multiple inheritances. Java implements more than one interfaces to cater for multiple inheritances.

iv. Multipath inheritance

In this inheritance, I want you to know a derived class is created from another derived class and the same base class of another derived class, as illustrated in figure 4.4.4. In the given example, you should understand class D inherits the properties and behaviour of class C and class B as well as Class A. Both class C and class B inherits the Class A. So, Class A is the parent for Class B and Class C as well as Class D. Java does not support this type of inheritance.

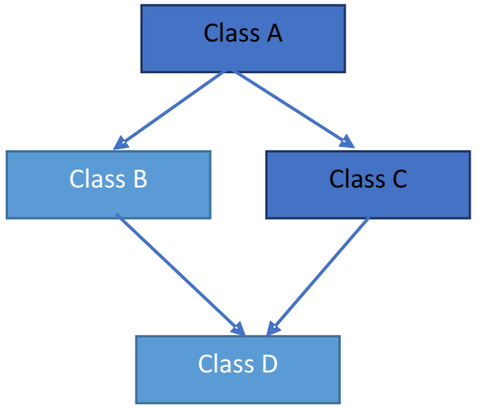


Figure 4.4.4: Multipath Inheritance

Hierarchical Inheritance

In this inheritance, try to note that more than one subclasses are created from a single superclass, and further subclasses act as a superclass for more than one subclasses. In the given example of figure 5, I want you to bear in mind that class A has two subclasses B and C. Furthermore, classes B and C have two subclasses - class D and E and class F and G, respectively.

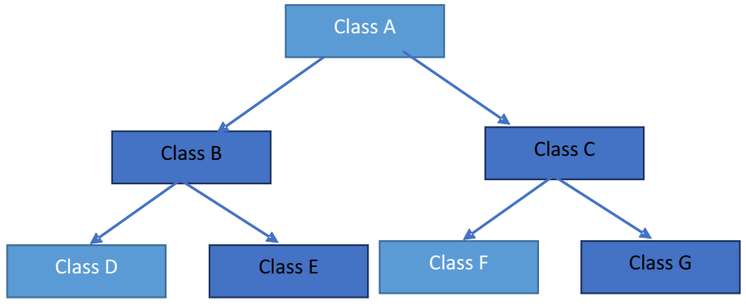


Figure 4.4 .5: Hierarchical Inheritance

In Java, the syntax is given below//Base Class


```
//Base Class
class A
{
 public void fooA()
 {
 //TO DO:
 }
}

//Derived Class
class B extends A
{
 public void fooB()
 {
 //TO DO:
 }
}

//Derived Class
class C extends A
{
 public void fooC()
 {
 //TO DO:
 }
}
public void fooD()
 {
 //TO DO:
 }

 //Derived Class
 class E extends B
 {
  public void fooE()
  {
  //TO DO:
  }
 }
```

```
//Derived Class
class F extends C
{
 public void fooF()
 {
 //TO DO:
 }
}

//Derived Class
class G extends C
{
 public void fooG()
 {
 //TO DO:
 }
}
```

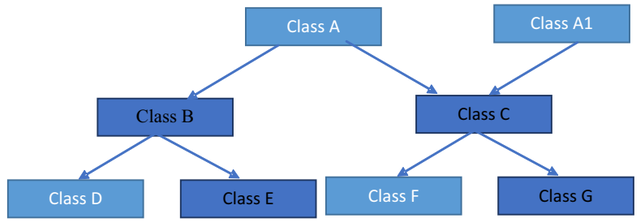


Figure 4.4.6: Hybrid Inheritance

Inheritance and controlling access

1 min

SAQ 3

I want you to carefully look into the following as the rules which must be enforced are applied to inherited methods.

i. methods declared public in a superclass also must be public in all subclasses.
ii. Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.
iii. methods declared private can notbe inherited.

Also, important for you to know is the access modifieras relates to inheritance which is given in the table below.

Access	Discription
Public	available to any caller both superclass and subclass
Private	available only to the class that defines it.
Protected	available only to the class that defines it and to any subclasses of that class.
Internal(Default)	available to any caller within the same package

Summary

In this unit, I am very sure we have fully explained the following:

i. the concept of inheritance
ii. types of inheritance
iii. achieving inheritance using control access

Self-Assessment Questions

1. explain the concept of inheritance
2. list types of inheritance
3. explain inheritance with access control attribute

Tutor Marked Assessment

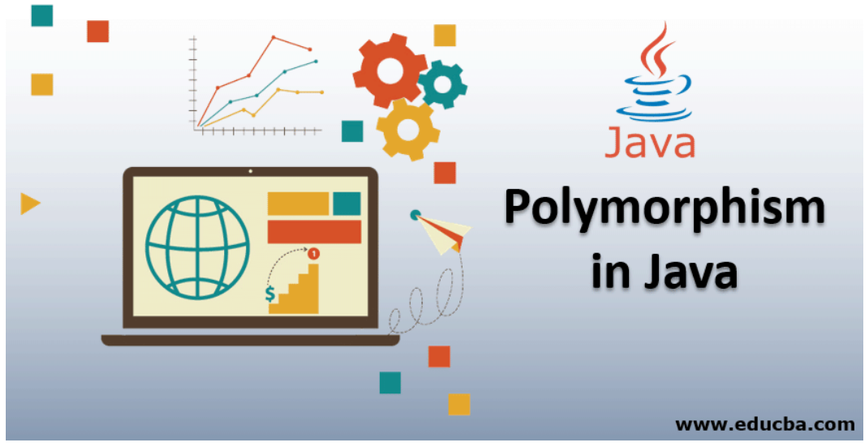
i. describe the concept of inheritance
ii. state types of inheritance
iii. achieve inheritance using java

Further Reading

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures i
Java: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

References

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures i
  Java: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.



UNIT 5

Polymorphism

Introduction

This unit explains to you the concept of polymorphism and different forms of polymorphism.

Learning Outcomes

At the end of this unit, you should be able to:

i. Explain the concept of polymorphism
ii. State the types of polymorphism

Main Content

1 min

SAQ 1

I want you to know Polymorphism is one of the OOPs features that allows the performance of a single action in different ways. Bear in mind that Polymorphism means one thing in many forms. Basically, polymorphism is the capability of one object to behave in multiple ways.

For example, I want us to consider a superclass Animal that has a method sound() since this is a generic class, so it gives a generic message. You should be aware that to have a specific message, there will be need for a specific implementation of sound().

```
public class Animal
{

     public void sound()
{
     System.out.println("Animal is making a sound");
      }
}
```

Now two subclasses of superclass Animal are Horse and Cat that extends Animal class and provide the implementation to the same method sound() thus:

```
public class Horse extends Animal
{       //Override
    public void sound()
     {System.out.println("Neigh");
      }
}
And public class Cat extends Animal
{
    //Override
    public void sound()
     {System.out.println("Meow");
      }
}
```


You should try to understand that the above shows common action for all subclasses sound(), but there are different ways to do the same action. This is a perfect example of polymorphism feature that allows performing a single action in different ways. Bear in mind that it would not make any sense just to call the generic sound() method as each Animal has a different sound.

Types of Polymorphism

1 min

SAQ 2

It is important for you to note that there are two types of polymorphism in java:

i. Static Polymorphism.
ii. Dynamic Polymorphism

i. Static Polymorphism, I want you to know it is also known as compile-time polymorphism, is achieved with overloading and operator overloading.

You should also be aware that A polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading is an example of compile-time polymorphism.

Method overloading is one of the ways java supports static polymorphism. Here you have two definitions of the same method add(). Which add method to 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 TestCalculator
{
intadd(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[])
   {
     TestCalculatorobj = new TestCalculator();
System.out.println(obj.add(10, 20));
System.out.println(obj.add(10, 20, 30));
   }
}
```

i. Dynamic Polymorphism, also known as runtime polymorphism, achieve with overriding means.It is vital you know 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 is why it is called runtime polymorphism.

I want you to carefully Consider

```
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();
   }
}
```

In this example, you have two classes ABC and XYZ. ABC is the superclass and XYZ is a subclass. The subclass is overriding the method myMethod() of the superclass. The subclass object is assigned to the superclass 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

• Summary

In this unit, I want to believe the following were discussed:

i. Concept of polymorphism
ii. Types of polymorphism

Self-Assessment Questions

1. Explain the Concept of polymorphism
2. Describe Types of polymorphisms

Tutor Marked Assessment

i. Explain the Concept of polymorphism
ii. Describe Types of polymorphisms

Further Reading

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures in Java: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

References

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures in Java: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

UNIT 6

Object Modeling

Introduction

In this unit, you will learn the concept of object modelling. You will also be introduced to the stages of modelling, including analysis, design and implementation.

Learning Outcomes

At the end of this unit, you should be able to:

- describe object model
- discuss the stages of modelling object
- discuss analysis
- describe the design of object
- implement code

1 min

SAQ 1

I want you to understand that objects oriented modelling contain stores values of instance variables. The development of objects from instance variables of an object is referred to as object-oriented modelling. You should also know it transforms the application and database development into a single data model and language. It will interest you to know Object-oriented modelling supports data abstraction, inheritance and encapsulation for object identification and communication. Object-Oriented Modeling is popular among other development techniques employed for system and software life cycle development for object-oriented programming. OOM is easy to draw, understand and use. Have it at the back of your mind that OOM is used in system life cycle development. OOM uses the entire phases of software development life cycle, which includes requirement specification, analysis, design, implementation, testing, deployment and maintenance.

3 mins

SAQ 2,3 4

I want you to know that this section applies the OOM to system development life cycle. Let me say the stages involved in the system development life cycle are analysis, design, implementation, testing, deployment and maintenance.

Analysis–Analysis is the first phase of OOM methodology I will want you to note. The purpose of the analysis phase is to study and understand the situation on ground. The characteristics and scope of the requirement of the system. It is concerned with getting things right from the beginning. It is important you are aware that the phase describes the detailed problem statement so as not to leave out important requirements and the objectives of the proposed system. Also, another important thing I will want you to note is the well-established problem statement which is categorized into three models; an object model, a dynamic model and a functional model.

Try to understand that the object model is to identify the static data structure need of the system and categorize the application into its identified objects. You should also know that the identified objects are then grouped into classes and the relationships between the objects. It also identifies the main attributes and operations requirements of each

class.

It is very important for you to note that the step by step process of object modelling is visualized in the following steps:

i. identify objects;
ii. identify classes;
iii. identify the relationships among classes;
iv. create a user object model diagram;
v. define user object attributes;
vi. define the operations that should be performed on the classes; and
vii. Review glossary.

The dynamic model, on the other hand, represents the relationship between identified objects events, states and transitions. You should be aware that the purpose of dynamic modelling is to analyze the objects in line with the requirements of the proposed system and response to internal and external events.

It will interest you to know that the process of dynamic modelling follows the steps highlighted:

i. identify states of each object;
ii. identify events and analyze the applicability of actions;
iii. construct dynamic model diagram, comprising of state transition diagrams;
iv. express each state in terms of object attributes; and
v. validate the state–transition diagrams drawn.

I want you to bear in mind that with the analysis phase, object model diagrams, state diagrams, event flow diagrams and data flow diagrams are generated.

Try to note that Functional Modelling is the last stage of object-oriented analysis. You should also know that the functional model describes the processes that object will perform and how data moves and changes from one method to another. It specifies the meaning of the operations of object modelling and the actions of dynamic modelling. The functional model corresponds to the data flow diagram of traditional structured analysis.

It is important you know that the process of functional modelling can be visualized in the following steps:

i. identify all the inputs and outputs;
ii. construct data flow diagrams showing functional dependencies;
iii. state the purpose of each function;
iv. identify constraints; and

v. specify optimization criteria.

System design - After the analysis phase follows the system design phase. I want you to know that the overall system architecture and the subsystems that form it are determined at the system design phase. I want you to also bear in mind that during the system design phase, the high-level structure of the proposed system is designed. At the design phase, critical decisions are made, which includes the organization of the desired system into subsystems, processes, and tasks to be performed by the subsystems putting into consideration relationships among them. Also, storage capacity is determined as well as management of global information.

Object design–You should be aware that the object design phase comes after the system design phase is over. Object design is concerned with fully classifying the existing and remaining classes, associations among the classes, attributes of each class and operations required to achieve the solution to the problem. I want you to know in object design, the implementation plan is developed, operations and data structures are fully defined along with any internal objects needed for implementation. Issues of inheritance, aggregation, association and default values are checked.

Implementation–I want you to knowthat implementation phase transforms the design into a programming language of choice. It is important to follow standard software engineering practice in order to have a smooth transformation from the design into the implementation phase. In selecting programming language, you should bear in mind that the following noteworthy points should be kept in mind:

i. to increase efficiency;
ii. to increase flexibility;
iii. to make amendments easily; and
iv. for the design traceability.

The databases are created, and the specific hardware requirements are ascertained. Once the code is in shape, it is tested using specialized techniques to identify and remove the errors in the code.

Generalization and Specialization

3 mins

I want you to note that generalization and specialization represent a hierarchy of relationships between classes and subclasses in an inheritance relation.

Generalization

In the generalization process, it will interest you to know that the common characteristics of classes are combined to form a class at a higher level of the hierarchy. In this process, try to understand that subclasses are combined to form a generalized super-class. It represents an "**is – a – kind – of**" relationship. For example, "car is a kind of land vehicle", or "ship is a kind of water vehicle". In this example, the car is a subclass of superclass land vehicle.

Specialization

Specialization is the reverse process of generalization. Here, I want you to know that the distinguishing features of groups of objects are used to form specialized classes from existing classes. Let me say it could be said that the subclasses are the specialized versions of the super-class.

Links and Association

Link

It is important you know that a link represents a connection among the objects through which the objects collaborates and exchange information with other objects. Another thing to note is Rumbaugh definition of linkwhich was defined as "a physical or conceptual connection between objects". Through a link, one object may invoke the methods or navigate through another object. A link shows the relationship between two or more objects.

Association

You should bear in mind that association is a group of links having common structure and common behaviour. Association depicts the relationship between objects of one or more classes. A link can be defined as an instance of an association.

Degree of an Association

Try to understand that degree of an association represents the number of classes involved in a connection. You should have it at the back of your mind that the degree may be unary, binary, or ternary.

A **unary** relationship connects objects of the same class.
A **binary** relationship connects objects of two classes.
A **ternary** relationship connects objects of three or more

classes.

Cardinality Ratios of Associations

Let me tell you that the cardinality of a binary association denotes the number of instances participating in an association. It is very important for for you to bear in mind that there are three types of cardinality ratios; these are:

One–to–One: A single object of class A is associated with a single object of class B.

One–to–Many: A single object of class A is associated with many objects of class B.

Many–to–Many: I want you to know that An object of class A may be associated with many objects of class B, and conversely an object of class B may be associated with many objects of class A.

Aggregation or Composition

Aggregation or composition is a relationship among classes where a class is made up of any combination of objects of other classes. Its important you are aware that It allows objects to be placed directly within the body of other classes. Aggregation is means "**part–of**" or "**has–a**" relationship, with the ability to move from the whole to its parts. An aggregate object is an object that is composed of one or more other objects.

It will interest you to know Aggregation denotes physical containment. Example, a computer is composed of monitor, CPU, mouse, keyboard, and so on.

Benefits of Object Model

The benefits of using the object model are:

i. it helps in faster development of software;
ii. it is easy to maintain. Suppose a module develops an error, then a programmer can fix that particular module, while the other parts of the software are still up and running;
iii. it supports relatively hassle-free upgrades;
iv. it enables reuse of objects, designs, and functions; and
v. it reduces development risks, particularly in the integration of complex systems.

Unified Modeling Language (UML)

3mins

The Unified Modeling Language (UML) is the graphical representation of Object-Oriented Analysis and Design (OOAD) with a standard way of developing systems software blueprint. It is important you know it gives visuals and documents the objects of an object-oriented system. It is used to represent the structures and the relationships in a proposed system.

Systems and Models in UML

The system is a set of elements organized to achieve certain objectives. It is good you know that Systems are organized into subsystems, and a set of models describes each subsystem.

I want you to also note that model is a simplified, complete and consistent abstraction of a system that is created for better understanding of the whole system.

Conceptual Model of UML

The Conceptual Model of UML has three major elements which I want you to take note of. These are:

i. Basic building blocks;
ii. Rules; and
iii. Common mechanisms.

i. Basic building blocks

The three building blocks of UML are –

a. Things

I want you to also note that there are four kinds of things in UML, namely:

Structural Things – These are the nouns of the UML models representing the static elements that may be either physical or conceptual. The structural things are class, interface, collaboration, use case, active class, components, and nodes.

Behavioural Things – These are the verbs of the UML models representing the dynamic behaviour over time and space. The two types of behavioural things are interaction and state machine.

Grouping Things – They comprise the organizational parts of the UML models. There is only one kind of grouping thing, i.e., package.

Annotational Things – These are the explanations in the UML models representing the comments applied to describe elements.

b. Relationships

Relationships are the connection between things. You should carefully look into the four types of relationships that can be represented in UML which are mentioned below

Dependency – This is a semantic relationship between two things such that a change in one thing brings a change in the other. The former is the independent thing, while the latter is the dependent thing.

Association – This is a structural relationship that represents a group of links having common structure and common behaviour.

Generalization
This represents a generalization/specialization relationship in which subclasses inherit structure and behaviour from super-classes.

Realization – This is a semantic relationship between two or more classifiers such that one classifier lays down a contract that the other classifiers ensure to abide by.

c. Diagrams

I want you to know that a diagram is a graphical representation of a system. It comprises of a group of elements generally in the form of a graph. UML includes nine diagrams in all, namely –

Class Diagram
Object Diagram
Use Case Diagram
Sequence Diagram
Collaboration Diagram
State Chart Diagram
Activity Diagram
Component Diagram
Deployment Diagram

ii. Rules

Rules
UML has some rules so that the models are semantically self-consistent and related to other models in the system harmoniously. UML has semantic rules for the following:
Names
Scope
Visibility
Integrity
Execution

iii. Common mechanisms

UML has four common mechanisms –

i. Specifications
ii. Adornments
iii. Common Divisions
iv. Extensibility Mechanisms

i. Specifications

In UML, behind each graphical notation, there is a textual statement denoting the syntax and semantics. These are the specifications. The specifications provide a semantic backplane that contains all the parts of a system and the relationship among the different paths.

ii. Adornments

Each element in UML has a unique graphical notation. Besides, there are notations to represent the important aspects of an element, like name, scope, visibility, etc.

iii. Common Divisions

Object-oriented systems can be divided in many ways. The two common ways of division are –
Division of classes and objects – A class is an abstraction of a group of similar objects. An object is a concrete instance that has actual existence in the system.
Division of Interface and Implementation – An interface defines the rules for interaction. Implementation is the concrete realization of the rules defined in the interface.

iv. Extensibility Mechanisms

UML is an open-ended language. It is possible to extend the capabilities of UML in a controlled manner to suit the requirements of a system. The extensibility mechanisms are:

Stereotypes – It extends the vocabulary of the UML, through which new building blocks can be created out of existing ones.

Tagged Values – It extends the properties of UML building blocks.

Constraints – It extends the semantics of UML building blocks.

Summary

i. object model
ii. stages of modelling object
iii. the analysis of object model
iv. design of object
v. implementation of code

Self-Assessment Questions

1. Explain Object model
2. State the stages of modelling object
3. Describe the analysis stage of object modelling
4. Explain the activities of Design of an object
5. Explain code Implementation.

Tutor Marked Assessment

i. Explain Object model
ii. State the stages of modelling object
iii. Describe the analysis stage of object modelling
iv. Explain the activities of Design of an object
i. Explain code Implementation.

Further Reading

- Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures in Java: Cambridge University Press
- Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.
- Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.
- Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.
- v .
- https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_uml_basic_notation.htm

References

Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures in Java: Cambridge University Press

Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.

Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.

Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

Java Operators

Module 5
Java Operators and Expressions

Units
Operators
Expression

UNIT 1

Operators

Introduction

In this unit, you be introduced to java operators and preference.Also, you will learn the various types of java operators to manipulate different types of variables.

Learning Outcomes

At the end of this unit, you should be able to:

- Discuss operators
- State their types
- Make operator preferences

Main Content

Basic Operators

6 mins

SAQ 1 & 2

I want you to know that these are set of symbols used to perform operations and manipulate both variables and values in java. The operators act on operands to yield a result which can be stored in a variable. It will interest you to know that Java provides different types of operators, as discussed in the next subheading.

Types of operators

It is very important for you to know that Java provides various types of *operators* according to the purpose it is required for. They are classified based on functionality. These are:

i. Arithmetic Operators
ii. Relational Operators
iii. Bitwise Operators
iv. Logical Operators
v. Assignment Operators
vi. Misc Operators

i. Arithmetic Operators

You should be aware that arithmetic operators are used in the same way mathematical expressions are used in algebraic expressions. I want you to carefully note the table below which provides the arithmetic operators with their description and example. If integer variable X holds 10 and variable Y holds 20, then:

Operator	Description	Example
+ Addition	Adds values on either side of the operator	X + Y will give 30
- Subtraction	Subtracts right -hand operand from the left-hand operand	X - Y will give -10
* Multiplication	Multiplies values on either side of the operator	X * Y will give 200
/ Division	Divides left -hand operand by right -hand operand	Y / X will give 2
% Modulus	Divides left -hand operand by right -hand operand and returns the remainder	Y % X will give 0
++ Increment	Increases the value of operand by 1	Y++ gives 21
-- Decrement	Decreases the value of operand by 1	Y-- gives 19

i. Relational Operators

Relational operators are used to determining the relationship between the two operands. It will interest you to know that it checks to determine if an operand is greater than, less than, equal to, not equal to and so on. The operator returns true or false based on the relationship. The following table describes relational operators supported by Java language. Supposing variable X holds 10 and variable Y holds 20, then:

Operator	Description	Example
== Equal To	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(X == Y) is **false**.
!=Not Equal To	Checks i f the values of two operands are equal or not, if values are not equal, then condition becomes true.	(X != Y) is **true**.
>Greater Than	Checks if the value of the left operand is greater than the value of right operand, if yes then condition becomes true.	(X>Y) is **false**.
<Less Than	Checks if the value of the left operand is less than the value of right operand, if yes then condition becomes true.	(X<Y) is **true**.
>= Greater Than or Equal To	Checks if the value of the left operand is greater than or equa l to the value of right operand, if yes then condition becomes true.	(X>= Y) is **false**.
<= Less Than or Equal To	Checks if the value of the left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(X<= Y) is **true**.

i. Bitwise Operators

It is very vital for you to note that Java bitwise operators are used to perform a bit-by-bit operation on operands. Try to understand that the operands involve integer types, long, int, short, char, and byte. The following table gives the java bitwise operators:

Suppose integer variable X holds 60 and variable Y holds 13, then:

X = 0011 1100
Y = 0000 1101

Operator	Description	Example
& Binary AND	Binary AND Operator copies a bit to the result if it exists in both operands.	(X&Y) will give 12 which is 0000 1100
| Binary OR	Binary OR Operator copies a bit if it exists in either operand.	(X | Y) will give 61 which is 0011 1101
^ Binary XOR	Binary XOR Operator copies the bit if it is set in one operand but not both.	(X ^ Y) will give 49 which is 0011 0001
~ Binary One's Complement	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~X) will give -60 which is 1100 0011
<< Binary Left Shift	Binary Left Shift Operator. The left operands valu e is moved left by the number of bits specified by the right operand.	X<< 2 will give 240 which is 1111 0000
>> Binary Right Shift	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	X>> 2 will give 15 which is 1111
>>> Shift right zero fill	Shift right, zero -fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	X>>>2 will give 15 which is 0000 1111

i. Logical Operators

I want you to know that Logical operators operate on boolean expressions. I want you to critically look into this table which lists the logical operators: Assume Boolean variables X holds true, and variable Y holds false, then:

Operator	Description	Example
&& Logical AND	If both the operands are non -zero, then the condition becomes true.	(X&&Y) is false.
|| Logical OR	If any of the two operands are non-zero, then the condition becomes true.	(X || Y) is true.
! Logical NOT	Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(X&&Y) is true. ! X is false!Y is true.

i. Assignment Operators

It is good for you to understand that assignment operators are used in Java to assign values to variables. The assignment operator assigns the value on its right to the variable on its left. Also, its important you bear in mind that Java supports the following sets of assignment operator which is shown in the table below:

Operator	Description	Example
= Simple assignment	Assigns values from right side operands to left side operand	Z = X + Y will assign value of X + Y into Z
+= Add AND assignment	It adds right operand to the left operand and assigns the result t o left operand	Z += X is equivalent to Z = Z + X
-= Subtract AND assignment	It subtracts right operand from the left operand and assigns the result to left operand	Z -= X is equivalent to
*= Multiply AND assignment	It multiplies right operand with the left operand and assigns the result to left operand	Z *= X is equivalent to Z = Z * X
/= Divide AND assignment	It divides left operand with the right operand and assigns the result to the left operand.	Z /= X is equivalent to Z = Z / X
%= Modulus AND assignment	It takes modulus using two operands and assigns the result to left operand	Z %= X is equivalent to Z = Z % X

i. Misc Operators

Java's miscellaneous operators are special operators used for specific purposes. These includes:

a. ternary operator
b. member access
c. comma
d. array index
e. new
f. instanceof
g. typecast

a. Ternary operator

The ternary operator is also known as the conditional operator. It is good you are aware that It consists of three operands and is used to evaluate boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator syntax is:

variable x =(expression)? value if true: value if false

I want you to look into this example
If a =10;
The expression b = (a ==1)? 20:30 will results in the value of **b** to be **30.**
The expression b = (a ==10)? 20:30 will results in the value of **b** to be **20.**

b. member access operator

it is vital you understand that the Java member operator is also called access operator or dot operator represented as

a dot (.) symbol that is used to access data members and member methods of a class by its objects.
The syntax is:

objectRefVar.dataFieldreferences a data field in the object.
objectRefVar.method(arguments) invokes a method on the object.
For example:
object a = new object();
a.ToString();

a. **Comma operator**
Java comma operator is a ',' sign that is used to separate function arguments and to declare more than one variable of the same type in one statement.

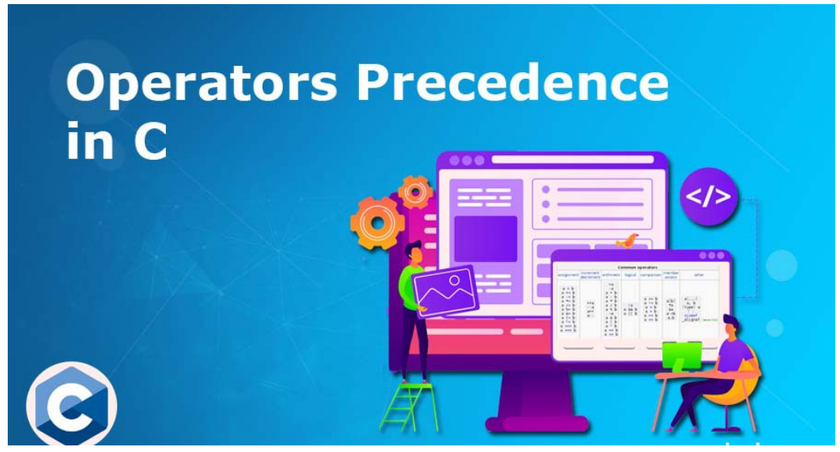
b. **array index operator**
Java array index operator, a set of square brackets ([]), is used to declare and access array elements.

c. **new operator**
I want you to know the Java new operator is used to create a new object. Operator new is a Java keyword. You need to be aware that it is followed by a call to a constructor, which initializes the new object. You should also Note that declaring an object and creating an object are two different things. Simply declaring a reference variable does not create an object. For that, we need to use the new operator. The new operator creates an object by allocating memory to it and returns a reference to that memory location. The Java new operator needs a single, postfix argument: a call to a constructor. The name of the constructor provides the name of the class to instantiate. Following piece of code demonstrates the use of a new operator.

a. **instance of operator**
it is important you know this operator is used only for object reference variables. I want you to bear in mind that the operator checks whether the object is of a particular type class type or interface type). instance of operator is written as: (Object reference variable)instance of (class/interface type)

b. **Typecast**
Java instance of operator which is also called type comparison operator. I want you to know it compares an object to a specific type. It follows the syntax objRefinstanceof type. Here objRef is the object name, and the type is the name of object type whom objRef will be compared to. The equals() method of Java is a nice

example that uses the instanceof operator to check if two objects are equal. The following example (InstanceOfDemo.java) shows the use of Java instanceof operator.



01 | Operators in java
source| educba.com

Operator Precedence

1 min

SAQ 3

You should be made to know that operator precedence refers to how terms in an expression are grouped, which determines how an expression is evaluated. Certain operators have higher precedence over the others; for example, the multiplication operator has higher precedence over the addition operator: For example, z = 6 + 4 * 2; here, x is assigned 14, not 20 because operator * has higher precedence over +, so it first gets multiplied with 4*2 and then adds 7. Its good you know that Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	*, /, %	Left to right
Additive	+, -	Left to right
Shift	>>>, >>, <<	Left to right
Relational	>>= <<=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Summary

In this unit, I want to believe the following were discussed

i. operators
ii. types of operators
iii. operator preferences

Self-Assessment Questions

1. What are operators?
2. state types of operators
3. describe operator preferences

Tutor Marked Assessment

i. What are operators?
ii. state types of operators
iii. describe operator preferences

Further Reading

- https://docs.oracle.com/javase/specs/jls/se7/html/jls-15.html
- https://notendur.hi.is/snorri/SDK-docs/lang/lang120.htm
- http://www.netfoo.net/oreilly/java/langref/ch04_js.htm

References

Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures in Java: Cambridge University Press

Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.

Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.

Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.

UNIT 2

Expression

Introduction

In this unit, I will critically explain the java expression to you. The various types of java expression and assignment of expression to different types of variables are examined. Errors generated as a result of a wrong expression will also be explained. I will also discuss oder of evaluation of an expression.

Learning Outcomes

At the end of this unit, you should be able to:

- Define expression
- Highlight the types of expression
- Discuss the evaluation of expression
- State the order of expression

Main Content

Expression

1 min

SAQ 1

I want you to that an expression represents a computation that involves variables, values and operators that evaluate to a single value. Sometimes an expression can as well gives a value to a variable. Have it at the back of your mind that an expression uses values, variables, operators and method calls to evaluate. An expression statement is terminated by a semicolon (;).

I want you to also note that expressions can result into a value using a wide range of Java arithmetic operators, conditional operators or comparison operators. For instance, arithmetic operators include + (addition), * (multiplication), / (division), ++ (increment) and % (modulus); conditional operators like &&(and), ? (ternary: compare two values), | | (or); and the comparison operators such as== (equal to), < (less than), <= (less than or equal to) and > (greater than).

Types of expression

3 mins

SAQ 2 & 3

You need to be aware that expression in Java is either used to assign, compute or save values. The data type of the expression must be the same as the variable to store it. I want you to try to understand that the value of the expression will therefore, be stored in a variable with the same data type; otherwise, an error will occur. Its worth note that a primitive type expression evaluates to a same primitive type. For example, an integer expression will be stored in an integer variable.

Class Instance Creation Expressions

A class instance creation expression is used when a new object of the instance of a class is to be created. It is very important for you to know it is said that a class is instantiated when an instance of the class is created using a class instance creation expression. You should also know that class instantiation comprises of class to be instantiated, enclosing instances (if any) of the newly created instance, a constructor is invoked to create the new instance, and arguments to be passed to that constructor.

Array Creation Expressions

I want to tell you that an array creation expression is used to create new

arrays. An array creation expression creates new array object having elements of the type specified by the primitive datatype or other objects. I want you to have it at the back of your mind that the type of each dimension expression must be an integer type; otherwise, a compile-time error occurs. This means that the specified dimension expression must be an integer. In an array creation expression, it is possible for you to have more than one dimension expressions, each having its own brackets. Each dimension expression to the left is fully evaluated before any dimension expression to its right.

Method Invocation Expressions

It is very important you note tahat a method invocation expression is required to invoke a class or instance method. At compile-time processing of a method invocation, several steps are involved in determining the method that will be invoked by a method invocation expression. I want you carefully note the three following steps as the most essential.

Step 1: Determine Class or Interface to Search for method invocation. First, determine the name of the method to be invoked and in which class or interface is the method definition.

Step 2: Determine Method Signature

The second step searches the class or interface determined in the first step for method declarations. This step uses the name of the method and the types of argument expressions to locate method declarations that are both applicable and accessible. A declaration that can be correctly invoked on the given arguments list. There may be more than one such method declaration, in which case the most specific one is chosen. The descriptor (signature plus return type) of the most specific method declaration is one used at run time to do the method dispatch.

Step 3: Is the Chosen Method Appropriate?

If there is a most specific method declaration for a method invocation, it is called the compile-time declaration for the method invocation and is the chosen method. Otherwise, further checks must be performed on the compile-time declaration to determine the appropriate method to be invoked.

Array Access Expressions

It is important you know that an array access expression refers to a variable component of an array. An array access expression contains two subexpressions, the array reference expression is before the left square bracket, and the index expression is within square the brackets.

I want you to have it at the back of you mind that in evaluating an array access expression, the array reference expression is first evaluated. Also important for you to note is that if an error occurs, then the array access expression completes abruptly, and for the same reason, the index

expression is not evaluated. Otherwise, the index expression is evaluated. If index expression evaluation completes abruptly, then the array access completes abruptly for the same reason.
Another thing to have in mind is that if the value of the array reference expression is null, then a **NullPointerException** is thrown. Note that the value of the array reference expression indeed refers to an array. It will interest to know that if the value of the index expression is less than zero, or greater than or equal to the array's length, then an **IndexOutOfBoundsException** is thrown.
However, I want you to bear in mind that if No error occurs, the result of the array reference is the variable of type T, within the array, selected by the value of the index expression.

Cast Expressions
You should try to understand that a cast expression converts, a value of one numeric type to a similar value of another numeric type; or verifies that the type of an expression is boolean; or checks that a reference value refers to an object whose class is compatible with a specified reference type at compile time. I want you to also note that cast operator refers to the parentheses and the type they contain in cast expression.

Another important thing I want you to know is the data type that appears in parentheses of a cast expression which refers to the data type of the cast expression. The result of a cast expression is not a variable, but a value, even if the result of the operand expression is a variable. At run time, the operand value is converted by casting conversion to the type specified by the cast operator. A **ClassCastException** is thrown if a cast is found at run time to be impermissible.

Evaluation of expression

2 mins

SAQ 4

I want you to be aware that every expression has a procedure of evaluation by which computational steps are executed. The process of evaluating an expression is either complete normally or complete abruptly. Its very important for you to note that an expression is said to complete normally or successfully if all the steps of processing are carried out without throw exception. On the other hand I will want you to bear in mind, an expression is said to complete abruptly of failed if an exception is thrown during the evaluation process. Every exception thrown has associated reason, which I will want you to keep note of as it is stated below:

a. **OutOfMemoryError**occurs if there is insufficient memory available in a class instance creation expression, array creation expression, array initializer expression, or string concatenation operator expression. Other expressions that can throw **OutOfMemoryError** are an assignment to an array component of reference type, a method invocation expression, or a prefix or postfix increment or decrement operates as a result of boxing conversion.

b. **NegativeArraySizeException**occurs if the value of any dimension expression in an array creation expression is less than zero.

c. **NullPointerException**is caused when the value of the object reference expression is null during a field access expression or array reference expression.

d. **NullPointerException** is occasioned by the target reference been set to null in a method invocation expression that invokes an instance method.

e. **ArrayIndexOutOfBoundsException** occurs if the value of the array index expression is negative or greater than or equal to the length of the array during an array access expression.

f. **ClassCastException**is thrown if a cast is found to be impermissible at run time in a cast expression.

g. **ArithmeticException** is thrown if the value of the right-hand operand expression is zero in an integer division or integer remainder operator.

h. **ArrayStoreException** is thrown when the value to be assigned is not compatible with the component type of the array in an assignment to an array component of reference.

I. An exception can be thrown as well if an error occurs that causes execution of the method body to complete abruptly.

j. A class instance creation expression exception can be thrown if an error occurs that causes execution of the constructor to complete abruptly.

k. Various linkage and virtual machine errors may also occur during the evaluation of an expression. By their nature, such errors are difficult to predict and difficult to handle.

l. If an error occurs during expression evaluation, then the evaluation

of the expressions can be terminated before all steps are completed; such expressions are said to **complete abruptly**.

An expression that requires evaluation of a sub expression may complete abruptly when the sub expression is abruptly completed.

Evaluation Order of expression

3 mins

I want you to know that in Java programming language, an expression is evaluated from left to right. It will interest you to know that the left-hand operand of a binary operator is first evaluated fully before any other part of the right-hand operand is evaluated. If for any reason an error occurs when evaluating the left-hand operand of a binary operator, no part of the right-hand operand of the operator will be evaluated. Thus, the evaluation completes abruptly. I want you to bear in mind that Java programming language follows the order of evaluation of expression as indicated by parentheses and operator precedence.

You should try to understand that in Java method or constructor invocation and class instance creation expressions, argument expressions are listed in parentheses, separated by commas. Each argument expression is fully evaluated before any part of the argument expression to the right. If an error occurs while evaluating the argument expressions, the evaluation completes abruptly, and no part of the argument expression to the right would be evaluated.

Summary

In this unit, I believe you have learnt the followingd:

i. expression
ii. types of expression
iii. evaluation of expression
iv. order of expression

Self-Assessment Questions

1. What is an expression?
2. Explain types of expression
3. Describe various order of expression
4. Explain what is meant by complete abruptly and complete normally.

Tutor Marked Assessment

i. What is an expression?
ii. Explain types of expression
iii. Describe various order of expression
iv. Explain what is meant by complete abruptly and complete normally.

Further Reading

- https://docs.oracle.com/javase/specs/jls/se7/html/jls-15.html
- https://notendur.hi.is/snorri/SDK-docs/lang/lang120.htm
- http://www.netfoo.net/oreilly/java/langref/ch04_js.htm

References

Wiener, R., & Pinson, L. J. (2000). Fundamentals of OOP and data structures in Java: Cambridge University Press

Litvin, M., &Litvin, G. (2001). Java Methods: An Introduction to Object-oriented Programming: Skylight Pub.

Carrano, F. M., & Prichard, J. J. (2004). Data abstraction and problem solving with Java: walls and mirrors: Pearson/Addison Wesley.

Etheridge, D. (2009). Java: the fundamentals of objects and classes: Bookboon.