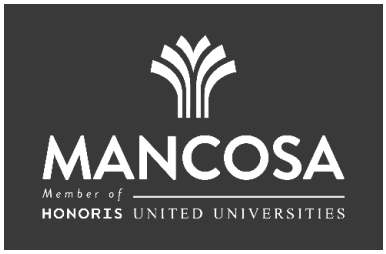
Bachelor of Commerce in Information and Technology Management (Year 1)

INFORMATICS 1B

Module Guide


MANCOSA
Member of
HONORIS UNITED UNIVERSITIES

Bachelor of Commerce
in Information and Technology Management (Year 1)
INFORMATICS 1B

Table of Contents

Preface

A. Welcome

Dear Student

It is a great pleasure to welcome you to **Informatics 1B (INF1B5)**. To make sure that you share our passion about this area of study, we encourage you to read this overview thoroughly. Refer to it as often as you need to, since it will certainly make studying this module a lot easier. The intention of this module is to develop both your confidence and proficiency in this module.

The field of **Informatics 1B (INF1B5)** is extremely dynamic and challenging. The learning content, activities and self- study questions contained in this guide will therefore provide you with opportunities to explore the latest developments in this field and help you to discover the field of **Informatics 1B (INF1B5)** as it is practiced today.

This is a distance-learning module. Since you do not have a tutor standing next to you while you study, you need to apply self-discipline. You will have the opportunity to collaborate with each other via social media tools. Your study skills will include self-direction and responsibility. However, you will gain a lot from the experience! These study skills will contribute to your life skills, which will help you to succeed in all areas of life.

Welcome to the MANCOSA Informatics 1B module, and congratulations on having passed Informatics 1A. We certainly hope that you enjoyed Informatics 1A, and that you found it enlightening, and that you will find this module just as enjoyable as challenging as the first module.

This module is essentially a continuation of the Informatics 1A. In Informatics 1A you took your first steps toward software development by writing your first computer programs. A complete software package, however, consists of more than just programming code; in reality, it consists of two major components: the program code and the *data* on which the code works.

Data is an extremely important aspect of a software application, because it is the data which is transformed by the computer program into useful information; and it is this information which is the desired outcome, because with it a company or an organisation can run efficiently. Hence it will be safe to say that in most cases the only reason for the existence of the computer system and the software application is the data; data is, essentially, what drives the information age.

Over the years, computer scientists have developed different ways to store different types of data in a computer's memory system. It is essential that each data type is stored correctly in its proper place; otherwise, the information which will be produced will be flawed, and can cause serious damage to a business. This module focuses primarily on the data aspect of software development.

This module is an integral part of the MANCOSA B. Com Information and Technology Management degree, and combines our world-renowned business courses with quality IT courses. We hope this degree will empower you with the knowledge and skills to enter a rewarding and lucrative career in the field of IT.

We hope you enjoy the module.

B. Module Overview

The module is a **15 credit** module at **NQF** level **5**.

About this Study Guide

This study guide comprises of 8 units:

Unit 1: Data types

In this unit you will be introduced to data types which are mechanisms for storing data.

Unit 2: Arrays and Linked Lists

Arrays and Linked Lists are two data structures for speedily storing, sorting and retrieving large amounts of data.

Unit 3: Object oriented programming - Introduction

In this unit you will be introduced to object oriented programming

Unit 4: Object oriented programming - advanced

In this unit you will be introduced to advanced object oriented concepts like inheritance.

Unit 5: Stacks and queues

Like arrays and Linked Lists, stacks and queues are data storage structures for quickly storing and retrieving large amounts of data. However, stacks and queues are highly specialised, and very different in the way they perform their functions.

Unit 6: Files

How would you store data permanently on disk, in order to retrieve it later? Databases and files are the de facto storage mechanisms for data. In this unit we will study the concept of file data storage.

Unit 7: Ethical and professional issues in Information Technology

Information technology has raised many social and professional issues for IT professionals. In this unit you will encounter some introductory concepts.

Unit 8: Conclusion

This unit provides a quick look into what is in store in Informatics 2A.

C. Exit Level Outcomes and Associated Assessment Criteria of the Programme

Exit Level Outcomes (ELOs)	Associated Assessment Criteria (AACs)
• The structure of data, information and knowledge in an organisational setting	• Structure of data, information and knowledge is reviewed within an organisational setting to emphasize data processes
• The application of information and knowledge management principles and theories in a variety of organisational settings	• Information and knowledge management principles and theories are applied in a variety of organisational settings to eliminate disarray
• The application of information and knowledge management principles in the different functional units of an enterprise	• Information and knowledge management principles is applied in different functional units of an enterprise to maintain consistency
• The architecture, platforms and configuration of systems to generate information and knowledge for decision making	• Architecture, platforms and configuration of systems is examined to generate information and knowledge for decision making
• Utilisation of information and knowledge management as a strategic tool for competitive advantage	• Information and knowledge management is applied as a strategic tool for competitive advantage to promote organisational information and knowledge management skills
• Analysis, evaluation and representation of financial, quantitative and functional information and knowledge for meaningful interpretation	• Financial, quantitative and functional information and knowledge is analysed, evaluated and presented for meaningful interpretation to establish organisational information and knowledge management needs

• Proposal of business solutions through information and knowledge management techniques	• Business solutions are proposed utilising information and knowledge management techniques to enhance organisational performance

D. Learning Outcomes and Associated Assessment Criteria of the Module

LEARNING OUTCOMES OF THE MODULE	ASSOCIATED ASSESSMENT CRITERIA OF THE MODULE
• Explain the meaning of abstract data types	• Abstract data types are reviewed in understanding the types of data and information to defined a set of value and a set of operations
• Explain the structure of internal data types such as stacks, linked lists and queues.	• Stacks, linked lists and queues are explored in grasping the concept of internal data type structures to facilitate the way of organizing and storing the data in a computer
• Explain the structure of external data types such as sequential, direct and indexed files	• Sequential, direct and indexed files are examined to understand external data type structures
• Explain and compare the object orientated programming paradigm and the component approach to program development	• Object orientated programming paradigm is explored and compared with the component approach in program development to understand JavaScript
• Develop object-oriented and component based programs in a computer language such as Java	• Object-oriented and component based programs is developed using Java to demonstrate general-purpose programming language that is class-based, object-oriented
• Explain introductory concepts concerning social and professional issues in Information Technology	• Social and professional issues in Information Technology is introduced and explored to determine the computer users' issues that they have experienced in their social and professional lives

E. Unit Learning Outcomes and the Associated Assessment Standards

You will find the Unit Learning Outcomes and the Associated Assessment Standards on the introductory pages of each Unit in the Module Guide. The Unit Learning Outcomes and Associated Assessment Standards lists an overview of the areas you must demonstrate knowledge in and the practical skills you must be able to achieve at the end of each Unit lesson in the Module Guide.

F. Notional Learning Hours

Notional Learning Hour Table for the Programme	
Types of learning activities	**Learning time**
	%
Lectures/Workshops (face to face, limited or technologically mediated)	**15**
Tutorials: individual groups of 30 or less	**0**
Syndicate groups	**0**
Practical workplace experience (experiential learning/work-based learning etc.)	**0**
Independent self-study of standard texts and references (study guides, books, journal articles)	**27**
Independent self-study of specially prepared materials (case studies, multi-media, etc.)	**40**
Other: Online	**18**
TOTAL	**100**

G. How to Use this Module

This Module Guide was compiled to help you work through your units and textbook for this module, by breaking your studies into manageable parts. The Module Guide gives you extra theory and explanations where necessary, and so enables you to get the most from your module.

The purpose of the Module Guide is to allow you the opportunity to integrate the theoretical concepts from the prescribed textbook and recommended readings. We suggest that you briefly skim read through the entire guide to get an overview of its contents. At the beginning of each Unit, you will find a list of Learning Outcomes and Associated Assessment Criteria. This outlines the main points that you should understand when you have completed the Unit/s. Do not attempt to read and study everything at once. Each study session should be 90 minutes without a break

This module should be studied using the prescribed and recommended textbooks/readings and the relevant sections of this Module Guide. You must read about the topic that you intend to study in the appropriate section before you start reading the textbook in detail. Ensure that you make your own notes as you work through both the textbook and this module. In the event that you do not have the prescribed and recommended textbooks/readings, you must make use of any other source that deals with the sections in this module. If you want to do further reading, and want to obtain publications that were used as source documents when we wrote this guide, you should look

at the reference list and the bibliography at the end of the Module Guide. In addition, at the end of each Unit there may be link to the PowerPoint presentation and other useful reading.

Programming, like many things in life such as driving a car, can only be learnt by doing. As such, in order to master the concepts in this module you will need to practice a lot. We recommend the following learning plan for studying this module:

1. Commence by reading through a section carefully, in order to get a fair understanding of the concepts being discussed
2. Where references are made to sections in the prescribed text, read those sections as well
3. Read both texts a second, possibly a third time, getting a firm grasp on the concepts being discussed
4. Try out the sample Java code in the study guide
5. Read "**The code explained**" in the study guide to get a good understanding of the code. Make sure that you understand every line of code, and what it is doing. Do not hesitate to experiment and try different things with the code.
6. Do the Activity. Remember that these activities are not optional!
7. At the end of the unit, do the **Review Exercises**

Your time allocation for this module should be no less than 8 hours per week.

H. Study Material

The study material for this module includes tutorial letters, programme handbook, this Module Guide, a list of prescribed and recommended textbooks/readings which may be supplemented by additional readings.

I. Prescribed Textbook

The textbook presents a tremendous amount of material in a simple, easy-to-learn format. You should read ahead during your course. Make a point of it to re-read the learning content in your module textbook. This will increase your retention of important concepts and skills. You may wish to read more widely than just the Module Guide and the prescribed textbook, the Bibliography and Reference list provides you with additional reading.

The prescribed textbook for this module is:

- Currie, Edward. Fundamentals of Programming using Java Cengage Learning. ISBN 978-1-84480-451-1

J. Special Features

In the Module Guide, you will find the following icons together with a description. These are designed to help you study. It is imperative that you work through them as they also provide guidelines for examination purposes.

Special Feature	Icon	Explanation
LEARNING OUTCOMES		The Learning Outcomes indicate aspects of the particular Unit you have to master.
ASSOCIATED ASSESSMENT CRITERIA		The Associated Assessment Criteria is the evaluation of the students' understanding which are aligned to the outcomes. The Associated Assessment Criteria sets the standard for the successful demonstration of the understanding of a concept or skill.
THINK POINT		A Think Point asks you to stop and think about an issue. Sometimes you are asked to apply a concept to your own experience or to think of an example.
ACTIVITY		You may come across Activities that ask you to carry out specific tasks. In most cases, there are no right or wrong answers to these activities. The purpose of the activities is to give you an opportunity to apply what you have learned.
READINGS		At this point, you should read the references supplied. If you are unable to acquire the suggested readings, then you are welcome to consult any current source that deals with the subject.
PRACTICAL APPLICATION OR EXAMPLES		Practical Application or Examples will be discussed to enhance understanding of this module.
KNOWLEDGE CHECK QUESTIONS		You may come across Knowledge Check Questions at the end of each Unit in the form of Knowledge Check Questions (KCQ's) that will test your knowledge. You should refer to the Module Guide or your textbook(s) for the answers.
REVISION QUESTIONS		You may come across Revision Questions that test your understanding of what you have learned so far. These may be attempted with the aid of your textbooks, journal articles and Module Guide.

CASE STUDY		Case Studies are included in different sections in this Module Guide. This activity provides students with the opportunity to apply theory to practice.
VIDEO ACTIVITY		You may come across links to Videos Activities as well as instructions on activities to attend to after watching the video.

Unit 1:

An Introduction to Data

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
1.1 Introduction 1.2 Data versus Information	• Understand what data is • Differentiate between data and information • Describe the various ways in which data is captured, or input, into a computer
1.3 Temporary versus Permanent Data	• Describe the hardware mechanisms within a computer which are used for temporary and permanent storage • Differentiate between temporary and permanent data storage within a computer, and recognise when each is useful
1.4 External and Internal Data Structures	• Differentiate between internal and external data
1.5 The Importance of Data Structures	• Understanding the importance of data structures
1.6 Abstract Data Types	• Define Abstract Data Types
1.7 Unit Summary	• Summarise topic areas covered in unit

Prescribed Textbook:

- Currie, Edward. Fundamentals of Programming using Java Cengage Learning. ISBN 978-1-84480-451-1

1.1 Introduction

Computers are used to input, store, process, find and share vast amounts of data and information, and they can do so in fractions of a second. Have you ever wondered how a computer is able to do this? This question will be answered during the course of this module, but before we begin, it is important to become familiar with some introductory concepts about data, such as exactly what data is, how it is captured, and how it is stored within a computer.

1.2 Data versus Information

Data can be described as facts and statistics collected together for reference or analysis. This data can be of many types: numbers, amounts, words, measurements, observations, descriptions of things, images, sounds or even videos. These facts and statistics are not in any particular order, nor do they necessarily mean anything.

Information, on the other hand, is data which has been processed so that it has a particular order, and placed into a context so that it has meaning. Consider the following example:

78 83 75 Tom 93 Sawyer

The above example means nothing to the observer: it is just a collection of numbers and names. This is typically how data appears in its raw form. Now if this data were to be placed into context, and ordered, it will acquire a meaning, as follows:

Student marks for Mathematics, Science, Biology and English:

Sawyer, Tom: 75, 83, 78, 93

This is what we refer to as information.

In the context of a business, the raw data will typically be in the form of figures such as monthly sales amounts coming in from sales representatives. This data will be captured into the business's information system by a data capture clerk, and then processed by the system into meaningful business data such as total sales amounts for a given period, profit and loss, etc.

1.3 Temporary versus Permanent Data

Temporary data is data which is stored in the memory of the computer. The reason that it is called "Temporary memory" is because it is volatile; which means that if the application which was responsible for the data being stored in the memory closes, or if the computer is shut down, all temporary data is lost.

Consider a person who is typing a document into a word processor app. While he types the document, it is stored in the temporary memory of the computer; if for some reason the word processor program crashes, or the computer goes off, we know exactly what will happen: he will lose his work. The reason his work was lost was that it was stored in temporary memory up until that point.

The main advantage of temporary memory is that it is extremely fast. Data can be added, removed and changed within the computer's memory at incredible speeds, so much so that thousands of operations can take place within the computer's memory, yet to the user they seem instantaneous.

Temporary memory is stored in a computer's Random Access Memory (RAM) chip.
The disadvantage of temporary memory, as you may have gathered, is the fact that it is short-lived, and there is no way to make your data persist after the application has been closed or the computer has been shut down. This effectively means that a person may be able to type in a document using a word processor and print it, but there will be no way to store the document away for future use.

This is where the concept of permanent data comes in. Permanent data, as the name suggests, is data which remains in existence even after the application which created it is closed, or the computer is switched off. Permanent data can be stored on a many different types of devices, such as hard disks, flask memory sticks, CD's, DVD's and tape drives.

The main disadvantage of permanent storage compared to temporary storage, is the speed; although permanent storage devices may be quite fast, they do not come close to temporary storage in speed. If a computer did not have a RAM chip, and had to use its hard disk to store and retrieve temporary data, this would cause the computer to become extremely slow, and tasks which usually take minutes could take hours to do.

As a software developer you will be making extensive use of both temporary as well as permanent data types, so it will be essential for you to understand the intricacies of both types of data. In fact, you have already made use of temporary storage in Informatics 1A, when you stored values in variables. Variables allow you to store information temporarily in the computer's memory, but as you may have noticed, they are destroyed when your app terminates.

Besides variables, there are numerous other ways to store information in the temporary storage mechanism of a computer, such as:

- Arrays
- Linked lists
- Stacks
- Queues
- Objects

All of the above will be discussed in detail in this module. Data can be stored permanently by your application using:

- Databases
- Sequential files

- Direct files
- Indexed files

Databases will be covered in later modules, while the others will be covered in this module.

1.4 External and Internal Data Structures

Variables, arrays, linked lists, stacks, queues and objects are known as **Internal Data Structures** because they are created and used from within your program.

Data which is stored in databases, sequential files, direct files and indexed files is known as External Data because they reside in sources external to your program, and are accessed by your program for processing.

1.5 The Importance of Data Structures

Remember that in most computer programs, the most important component is the data. As a result, the data which is input, stored and processed by the program needs to be pristine, in order for the correct operation of your program. Using correct data structures in your program is critical for the following reasons:

1. **Correct and efficient operation of the program**: Remember that the computer's resources, such as memory and processor power, are limited, and have to be used in the most efficient manner possible in order to avoid programs which run slowly or, worse still, crash.
2. **Simplification of the programming task**: If your data structures are correctly selected and used, the logic required to manipulate the data is relatively simple.
3. **Algorithm efficiency**: as you learnt in the previous module, there may be many different algorithms to solve a single problem, but not all will be efficient; some may indeed solve the problem at hand, but in a slow and inefficient way which hogs the computer's resources. It has been proven that the best and most efficient algorithms are those which make use of the most suitable data structures.
4. **Correctness of information derived from the data**: this follows the age old principle that in order to produce a quality end product, the raw materials have to be of a good quality. The same is true for information – if the data is captured and stored in the most suitable data structures, then the information which is derived from that data will be useful.

1.6 Abstract Data Types

To understand the concept of an abstract data type, it will first be necessary to understand the meaning of the word "abstract" as it relates to programming. We will explain this via the example of a car: in order to drive a car, we need to switch it on, and press the accelerator to propel it forward. Most of are not aware of, or even concerned with, the complexities of what goes on under the bonnet of the car; all we are concerned with is how to drive the car. In fact, it really isn't necessary to know anything about the engine in order to be a good driver.

This example can be applied to most things in life - be it a computer, a DVD player, a kitchen appliance or a car - for the most part, we are concerned with how to use them, not with how they work. The actual inner workings of these items have been conveniently hidden away from us, and we are content with merely using them without ever worrying about how they work.

This concept of separating the inner workings of an object from its use is called **abstraction**. In a car, the engine and its various parts, as well as all the other related mechanisms which enable the car to work, have been concealed under the bonnet. All that the driver ever sees, or ever needs to see, are the accelerator, brakes, gear lever, steering, etc. We say that the inner workings of the car have been "abstracted away" from the driver.

This concept of abstraction has been adopted by computer scientists and applied to data. **Data abstraction** is defined as a process of separating the logical properties of the data from its implementation. In other words, data abstraction is where we instruct our app to perform certain operations on our data, like storing it, editing it, deleting it, sorting it, etc. without concerning ourselves with *how* the operations are actually performed. Hence the following definition:

Abstract data type: A data type that separates the logical properties from the implementation details.
In this module we will encounter numerous abstract data types, some of which are provided by Java, while others will be user-defined.

1.7 Unit Summary

- Data can be described as facts and statistics collected together for reference or analysis
- Information is data which has been processed and placed into a context so that it has a particular order and meaning
- Temporary data is data which is stored in the memory of the computer temporarily, and is lost when the app terminates or the computer is switched off
- Permanent data is data that is stored on storage media such as hard disks, flask memory sticks, CD's, DVD's and tape drives, and persists even after the application is closed or the computer is switched off.
- Variables, arrays, linked lists, stacks, queues and objects are known as **Internal Data Structures** because they are created and used from within your program
- Databases, sequential files, direct files and indexed files are known as **External Data Structures** because they reside in sources external to your program, and are accessed by your program for processing
- Using the correct data structures is critical for correct and efficient operation of your app
- An **abstract data type** is a data type that separates the logical properties from the implementation details.

1.8 Review Questions

1. Define the term abstract data type
2. Differentiate between data and information
3. Differentiate between internal and external data types
4. Explain some of the consequences which may arise if proper data structures are not used in a computer program.

1.9 Answers to review questions

1. An abstract data type is a data type that separates the logical properties from the implementation details.

2. Data can be described as facts and statistics collected together for reference or analysis. These facts and statistics are not in any particular order, nor do they necessarily mean anything. Information is data which has been processed so that it has a particular order, and placed into a context so that it has meaning.

3. Internal Data Structures are data types which are created and used within a computer program. External data types are data types that are stored outside a computer program and are accessed by your program for processing. Such data is stored in databases, sequential files, direct files and indexed files.

4. If the correct data types are not used, the following consequences will arise:

- **Your program may not work correctly and efficiently**: The computer's resources, such as memory and processor power, are limited, and have to be used in the most efficient manner possible in order to avoid programs which run slowly or, worse still, crash.

- **Programing tasks may become very complex**: If your data structures are incorrectly selected and used, the logic required to manipulate the data will be needlessly complex.

- **Algorithm inefficiency**: Using incorrect data types will lead to algorithms which may solve the problem at hand, but will do so in a slow and inefficient way which hogs the computer's resources. It has been proven that the best and most efficient algorithms are those which make use of the most suitable data structures.

- **Incorrectness of information derived from the data**: this follows the age old principle that in order to produce a quality end product, the raw materials have to be of a good quality. The same is true for information – if the data is captured and stored in the most suitable data structures, then the information which is derived from that data will be useful; if not, then the data will be useless.

Unit 2:

Arrays and Linked Lists

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
2.1 Introduction 2.2 The problem with Variables	• Understand what an array is • Understand how to implement an array in programming languages
2.3 Introduction to Arrays	• Describe how data is stored within an array
2.4 Accessing the elements of an array	• Understand the index system of arrays
2.5 Iterating through an array	• Store and retrieve data from an array
2.6 Declaring and using a String Array	• Identify the advantages and disadvantages of arrays
2.7 Linked Lists	• Explain what a linked list is • Understand how linked lists are structured and how they work
2.8 Linked lists in Java	• Implement a linked list in a programming language
2.9 Linked lists versus Arrays	• Store and retrieve data from a linked list • Identify the advantages and disadvantages of linked lists
2.10 Unit Summary	• Summarise topic areas covered in unit

Prescribed Textbook:

- Currie, Edward. Fundamentals of Programming using Java Cengage Learning. ISBN 978-1-84480-451-1

2.1 Introduction

In Informatics 1A you looked at variables, which provide a powerful and convenient way to store information temporarily in a computer's memory. Variables are suitable for small amounts of data, but as you will discover in this unit, they become quite unwieldy and difficult to work with when it comes to large amounts of data, which you will encounter almost always.

The answer to this limitation of variables is to store data in a collection like an array or a linked list. In this unit you will gain an in-depth understanding of arrays and linked lists, their structure, their uses, their advantages and disadvantages, and how they are implemented by programming languages.

2.2 The problem with Variables

Assume you need to capture a student's examination mark into your computer application for processing and storage. Building on your knowledge from Informatics 1A, you would display an input dialog box and request the user to type in the mark, as follows:

```
String mark;
mark = JOptionPane.showInputDialog("Please type in the student's mark:");
int student1Mark = Integer.parseInt(mark);
```

```
String mark;
mark = JOptionPane.showInputDialog("Please type in the student's mark:");
int student1Mark = Integer.parseInt(mark);

mark = JOptionPane.showInputDialog("Please type in the student's mark:");
int student2Mark = Integer.parseInt(mark);

mark = JOptionPane.showInputDialog("Please type in the student's mark:");
int student3Mark = Integer.parseInt(mark);
```

This will be perfectly fine for one single student's mark. But if you need to capture a second and a third mark, you would have to repeat the code above, and store the marks in new variables called something like student2Mark and student3Mark, as shown below:

Can you imagine how complex your code will become if you had 20 or 30 marks? Or a hundred or a thousand? It will become impossible to manage! Also, by storing the marks in separate variables as done above, there is no simple way to iterate through the marks and performing calculations on them, such as adding them up, sorting them, comparing them, etc.

This is where the concept of an array becomes useful.

2.3 Introduction to Arrays

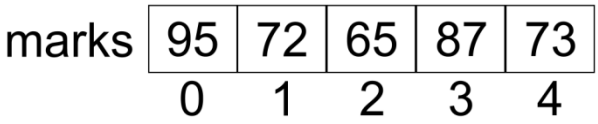
Very simply put, an array is a single variable with multiple compartments, like a filing cabinet with many files in it. An array can have as many of these 'compartments' as you need, and you can store your data in those compartments. Each item in the array is then referenced by the name of the array, followed by a unique index number, which serves as an address for each element in the array.

Assuming you had a list of 5 students' marks which you needed to capture. Using an array, you would do it as shown below:

```
1 int[] marks = new int[5];
2
3 marks[0] = 95;
4 marks[1] = 72;
5 marks[2] = 65;
6 marks[3] = 87;
7 marks[4] = 73;
```

The code explained

The code above will create a scenario in the memory of the computer, which can be illustrated as follows:



The above is a fairly accurate illustration of an array. The name of the array is marks, and it has 5 elements, each having a unique index number from 0 to 4. Array indexes always commence with 0. This means that an array with n elements will always have 0 as its first index, and n-1 as its last element. For example, the last index of an array with 10 elements will be 9.

- **Line 1**: This line is declaring the array. Like any variable, an array must be declared before it can be used. Each part of line 1 is explained below:

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

  int[]: This declares the type of the array we are creating, which is int. The square braces after the keyword int are necessary, and indicate to Java that this is an array.

  marks: This is the name of the array

  new: The keyword new is used to instantiate, or create, a new object. We will deal with objects later in this module

  int[5]: This is where we are actually creating the array. The statement simply instructs Java to create an integer array with 5 elements

- **Line 3**: This instruction takes the number 95 and stores it in position 0 of the array marks. Lines 4 to 7 do the same thing, storing the numbers on the right hand side of the assignment operator (=) into positions 1 through 4 of the array.

2.4 Accessing the elements of an array

The elements in an array are referenced through their respective indexes. Assuming you needed to output the first student's mark; it will be done as follows:

```
System.out.println(marks[0]);
```

This will output the number 95 to the command line.

The elements in an array do not need to be accessed in the order in which they appear in the array. You may access them in any order:

```
System.out.println(marks[4]);
System.out.println(marks[1]);
System.out.println(marks[3]);
System.out.println(marks[0]);
```

The above code will produce the following output:

```
73
72
87
95
```

Activity 2.1

Write a program to do the following:

1. Input 10 students' marks via a showInputDialog box
2. Output the first 5 marks to the screen via a dialog box

2.5 Iterating through an array

A common operation on an array is to step through its elements, one by one. This is known as iterating through the array, and is extremely useful when you need to sort the elements in the array, search through the array, add the elements in the array, etc.

An important point to bear in mind is that the index of an array element is always an integer, as in the case of the number 3 in the square braces:

```
marks[3] = 87;
```

Because the index is an integer, it can be replaced with an integer variable, as shown below:

```
int indexNumber = 3;
marks[indexNumber] = 87;
```

The above code is equivalent to the previous code; that is, in both cases the 3rd element of the array is being populated with the number 87.

The fact that you can use a variable as the index of an array makes it possible to iterate through the array using a loop, using counter variable of the loop as the index of the array. This is demonstrated in the code below:

```
 1 int[] marks = new int[5];
 2
 3 marks[0] = 95;
 4 marks[1] = 72;
 5 marks[2] = 65;
 6 marks[3] = 87;
 7 marks[4] = 73;
 8
 9 for(int counter = 0; counter <= 4; counter++){
10    System.out.println(marks[counter]);
11 }
```

The code explained

- **Line 9**: Here we commence a loop which will count from 0 to 4, which corresponds to the indexes of the elements in the marks array. If the loop counts to more than 4, it will result in a runtime error, because there is no element in the array with an index number greater than 4
- **Line 10**: This line outputs the element in the array which corresponds to the current value of counter during each iteration of the loop. For example, the first time line 10 is executed, the value of counter will be 0. This will be equivalent to the following:
- System.out.println(marks [0]);

  The output of this will be 95 When the loop is executed the second time, the value of counter will be 1, which will be equivalent to:

  System.out.println(marks[1]);

  This will output 72.

  This will continue until the loop terminates at 4.

Activity 2.2

1. Modify the program you wrote in the previous exercise to output all the elements of the array using a for loop

Adding up the elements of an array

The following code will add up the values of all the elements in the marks array:

```
1 int total = 0;
2 int[] marks = new int[5];
3
4 marks[0] = 95;
5 marks[1] = 72;
6 marks[2] = 65;
7 marks[3] = 87;
8 marks[4] = 73;
9
10 for(int counter = 0; counter <= 4; counter++)
11 {
12    total = total + marks[counter];
13 }
14
15 System.out.println(total);
```

The code explained

- **Line 1**: Here we are declaring a variable called total which will hold the total value of all the elements of the array
- **Line 12:** With each iteration of the loop the value of marks [counter] is added to total.

2.6 Declaring and using a String Array

Just as you can store integers in an array, you can store values of other types such as Strings. An array of type String will be declared as follows:

```
String[] studentNames = new String[5];
```

The above code will declare an array of type String, with 5 elements. To store String elements in the arrays, you will do the following:

```
studentNames[0] = "Tom";
```

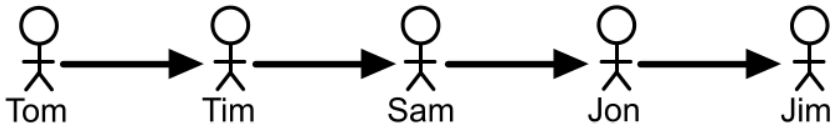
Activity 2.3

Modify the program you wrote in the previous exercise to do the following:

1. Create an array of type String which will hold 10 elements
2. Populate the array with 10 students' names
3. Output the names of the students in the array using a for loop.

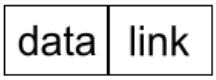
2.7 Linked Lists

A linked list is a collection of data items where each item is linked to its successor. You can think of a linked list as a group of people where each person is only aware of the person who comes after them in the group – they are not aware of who is in the rest of the group. Look at the diagram below:



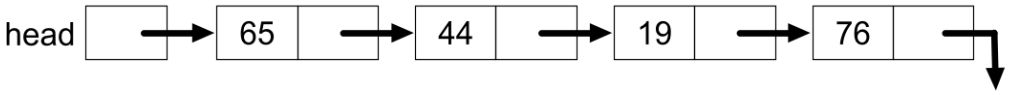
In the diagram, Tom is the first person in the group, and he is aware of his successor, Tim. Tim, on the other hand, is unaware of Tom, but is aware of Sam, and so forth.

Each element stored in a linked list is known as a "node". Each node is made up of two parts:



1. The data: this is the actual data stored in the node
2. The link: this is a reference to the successor, the node which comes after the current one.

The following diagram illustrates a linked list:



2.8 Linked lists in Java

Java provides built-in functionality to create and use linked lists. In order to create a linked list, you need to create an instance of the linked list class:

LinkedList myList = new LinkedList();

The class LinkedList has numerous methods for handling data in the list, for example:

- add(5): inserts the number 5 item to the end of the list
- add(2, 5): insert 5 at position 2 of the list (linked list elements start at index 0, just like arrays)
- addFirst(5) : inserts the number 5 at the beginning of the list
- addLast(5): inserts the number 5 at the end of the list
- remove(3): removes the item at position index 3 of the list; in other words, remove the 4th element from the list.
- removeFirst(): removes the first element from the list
- removeLast(): removes the last element from the list
- set(2,34): sets the value of the third element in the list to 34

The following Java program demonstrates how to create a linked list and to use the above methods:

```
1 import java.util.*;
2 class LinkedListDemo {
3    public static void main(String args[]) {
4       // create a linked list
5       LinkedList myList = new LinkedList();
6       // add elements to the linked list
7       myList.add(65);
8       myList.add(44);
9       myList.addLast(57);
10      myList.addFirst(92);
11      myList.add(3, 52);
12      System.out.println("Original contents ofmyList: " + myList);
13      // remove elements from the linked list
14      myList.remove(2);
15      System.out.println("Contents ofmyList after deletion: "  + myList);
16      // remove first and last elements
17      myList.removeFirst();
18      myList.removeLast();
19      System.out.println("List after deleting first and last: " + myList);
20      // get and set a value
21      Object val = myList.get(1);
22      myList.set(1, 34);
23      System.out.println("List after change: " + myList);
24   }
25 }
```

Activity 2.4

Determine what the output of the above program will be.

2.9 Linked lists versus Arrays

The primary difference between an array and a linked list is that an array's size is fixed once it is created – it cannot accommodate more elements. A linked list, on the other hand, does not require you to fix its size when you create it – hence you can continue adding elements indefinitely.

2.10 Unit Summary

- An array is a variable which can store multiple values
- Each element which is stored in an array has a unique index number
- Each element in an array can be referenced by its index number
- You can iterate through an array quickly and easily using a loop
- Arrays can store elements of type integer, String or any other type

2.11 Review Questions

1. Answer the following questions:

1.1. All the elements in an array have one thing in common. What is it?

1.2. What do we call the number used to call each element in an array?

1.3. What will happen if you call an index of an array which does not exist?

2. Declare an array of type String with 4 elements and populate it with 4 names of your choice

3. There is an error in the following code. What is it?

```
String[] marks = new String[5];

marks[0] = 95;
marks[1] = 72;
marks[2] = 65;
marks[3] = 87;
marks[4] = 73;
```

4. Write a program called Student Marks that inputs 10 student names into an array called “students”, and to input 10 marks into an array called “marks”. Next, output the name of the students with the corresponding mark alongside it. For example, if the first name in the array students is “Tom”, and the first mark in the array marks is “80”, then your program should output the following:

```
Tom 80
```

5. And so forth, until all 10 names and marks have been outputted in the same format.

2.12 Answers to Revision Questions and Activities

1.1 They share the same name

1.2 The index

1.3 It will result in an “Array index out of bounds” runtime error

2. The code follows:

```
String[] names = new String[4];
names[0] = "Jon";
names[1] = "Tim";
names[2] = "Sam";
names[3] = "Ann";
```

3. The array is declared as type String, but is storing integers instead of String types.

4. The code follows:

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

      String[] names = new String[10];
      names[0] = "Jon";
      names[1] = "Tim";
      names[2] = "Sam";
      names[3] = "Ann";
      names[4] = "Pat";
      names[5] = "Harry";
      names[6] = "Sally";
      names[7] = "Mike";
      names[8] = "Mary";
      names[9] = "Jane";

      int[] marks = new int[10];
      marks[0] = 56;
      marks[1] = 91;
      marks[2] = 83;
      marks[3] = 45;
      marks[4] = 67;
      marks[5] = 94;
      marks[6] = 34;
      marks[7] = 77;
      marks[8] = 43;
      marks[9] = 76;

      for(int count = 0; count <= 9; count ++){
         System.out.println(names[count] + " " + marks[count]);
      }
   }
}
```

Unit 3:

Object Oriented Programming

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
3.1 Introduction 3.2 Classes and Objects	• Understand what a class is • Understand what an object is
3.3 Object Oriented Programming	• Explain and compare the object oriented programming paradigm and the component • Approach to program development
3.4 Object oriented programming in Java	• Develop object-oriented and component based programs in Java
3.5 Constructors	• Explain and describe constructors
3.6 Implicit constructors	• Understanding constructors
3.7 Overloaded constructors	• Analyse overloaded constructors
3.8 Unit Summary	• Summarise topic areas covered in unit

Prescribed Textbook:

- Currie, Edward. Fundamentals of Programming using Java Cengage Learning. ISBN 978-1-84480-451-1

3.1 Introduction

In this unit you will be introduced to the concepts of Classes and Objects, and you will take your first steps into the world of Object-oriented programming (OOP). Object-oriented programming greatly simplifies the process of software development by breaking code down into small, manageable parts, and by facilitating the use of reusable code components. It is virtually impossible to create an app of decent size without using OOP, since almost all modern programming languages, including Java, are object-oriented.

3.2 Classes and Objects

Objects in real life

In order to understand the concept of object-oriented programming, it is essential to understand what an object is. Hence we will begin with an explanation of the concept of an object.

If you look around you, you will see numerous real-world objects, such as your pen, your books, your fan, your cell phone etc. Each of these objects has two characteristics: state and behaviour. By state we mean what the object *is*, and by behaviour we mean what the object can *do*. For example, your fan may have a few states: off, low speed, medium speed and high speed. It will have multiple behaviours as well: switch to low speed, switch to medium speed, switch to high speed and switch off.

Observing real life objects in terms of their states and behaviours will provide you with a very good idea of what an object is in programming.

Activity 3.1

Make a list of 10 real life objects around your home or workplace, listing next to each one about 5 of its states and behaviours.

Objects in Programming

Objects in programming are similar to objects in real life, in the sense that they also comprise of states and behaviours. The states of an object will be represented by its variables, while the behaviours are represented by its methods.

As an example, a C*at* object may have the following variables and methods:

Cat
Variables:
name
colour
age
height
weight
Methods:
eat()
sleep()
meow()
walk()

Similarly, a *Bank Account* object may have the following variables and methods:

Bank Account
Variables:
account type
account number
balance
Methods:
depositMoney()
withdrawMoney()
showBalance()

The names of the methods above describe what they do.

An object can hence be defined as follows: “An object is a software component which is a combination of data and methods which act on that data, which can be treated as a single unit. An object can be a part of an application or an entire application.”

Simply put, objects can be thought of as "smart" data types which mimic real life objects. Like other data types they are able to store data, but they also include methods which can do things with its own data, thereby saving the programmer both time and effort in performing operations on that data.

3.3 Object Oriented Programming

Now that you understand what an object is, it will be easier to understand the concept of Object-oriented programming (OOP).

Traditionally, computer programs comprised of one single file with a long list of code, which could run into tens of thousands of lines. As programs became bigger and bigger, it became more and more difficult to maintain such code. Programmers then decided to "chunk" the code into reusable parts called "subroutines", which we know as "methods". Methods made life much easier, but soon a time came when computer programs became so big that even methods were no longer sufficient to make the code easy to maintain. Added to that, there was the problem of reusability – functionality in one program or one file could not be easily reused in other programs or in other parts of the same program.

Computer scientists then saw a need to break computer programs down into smaller, more manageable chunks. The question was, how would they break them up? To answer this question, they looked to the real world, and observed that the real world is comprised of *objects*. Everything around us, and in fact we ourselves, are objects. Those objects in turn are made up of even smaller objects. As an example, think of a car. A car is an object, but it is also made up of smaller objects, like the engine, the doors, the steering wheel, etc. Going further, the parts of the car are made up of even smaller objects, as in the case of the engine, which is made up of pistons, spark plugs, and a myriad of other parts. Each of the parts of the engine is an object too, and each performs a small, well—defined function within the engine.

Reusability

Looking at real life objects like car engines, computer scientists made another observation: that the individual objects which comprise the engine are reusable. In other words, a part in one engine can be removed and used in another engine of a similar type, where it will work perfectly. As a result, factories needed only to manufacture generic parts which worked across many engines, rather than making a new set of parts for each engine.

This presented many possibilities for the field of software development. Could it be possible to create miniature software components, or "parts", which could perform small, well-defined tasks, and then put them together to perform more complex tasks? Could it be possible to create those components in such a way that they could be reused in different computer programs, without the need to rewrite components for each computer program? It was discovered that this was possible, via the concept of object-oriented programming.

Object-oriented programming revolutionised the world of computer programming. Programmers now began using individual objects which they "wired up" together, very much like Lego bricks, to form complex software packages. One of the best things about using objects was that an object written for one software package could be reused in a totally different software package. The objects were not even required to be written in-house; programmers could

use objects written by other programmers in their programs. As you can well imagine, this saved them a lot of time and effort.

Reusable objects

A good example of a reusable object in a computer program is an on-screen button, like the examples shown below:

Buttons, as simple as they seem, are remarkably complex objects, and typically require dozens of lines of code to create. Thanks to object-oriented programming, however, you as a programmer will probably never have to create a button from scratch. This task of creating buttons has been done by others, and all you have to do is to reuse the code as many times as you need within your own program. This is the benefit of Object-oriented programming.

3.4 Object oriented programming in Java

Below is the implementation of the Cat class in Java:

```
public class Cat{
   public String name;
   public String colour;
   public int age;
   public int height;
   public int weight;

   public void eat(){
      System.out.println("Cat eating");
   }

   public void sleep(){
      System.out.println("Cat sleeping");
   }

   public void meow(){
      System.out.println("Meow!");
   }

   public void walk(){
      System.out.println("Cat walking");
   }
}
```

The code explained

Class Cat has 5 variables, namely name, colour, age, height and weight; it has 4 methods which simply output text to the console.

You will notice that unlike the previous classes you have created, class Cat does not have a main method. As a result of this, class Cat can be compiled, but it cannot be run. We will ow create another class with a main method which will use class Cat:

```
1 public class UseCat{
2    public static void main(String[] args){
3    Cat c1 = new Cat();
4
5    c1.name = "Tixe";
6    c1.colour = "ginger";
7    c1.age = 2;
8    c1.height = 25;
9    c1.weight = 2;
10
11   System.out.println(c1.name);
12   System.out.println(c1.colour);
13   System.out.println(c1.age);
14   System.out.println(c1.height);
15   System.out.println(c1.weight);
16
17   c1.meow();
18   }
19 }
```

The code explained

- **Line 1:** As the name suggests, class UseCat has only one purpose: to make use of class Cat
- **Line 3:** In this line we are doing a few things:
  - Cat c1: We declare a variable called c1 whose type is Cat. This means that it will store an instance of class Cat
  - new Cat(): The keyword new is used to create a new instance of a class. This new instance of class Cat is then assigned to the variable c1.
- **Line 5 to Line 9:** Here we are setting values for the variables in the Cat object c1. Notice how we reference the object using the dot operator. This is the standard way to reference any variable or method of an object.
- **Line 11 to Line 15:** Here we are outputting the variables we have just set.
- **Line 17:** we are calling method meow() of object c1. This method call will cause the word “meow!” to be output to the console.

3.5 Constructors

Let's take another look at the code to instantiate a class:

Cat c1 = new Cat();

Did you notice that the Cat()part of the line looks remarkably like a method call? The reason for this is that it *is* in fact a method call. It is a call to a special method within the class called the constructor.

Constructors are class methods that are automatically executed when an object of a given type is created. As their name suggests, constructors play a role in "constructing" objects; they initialise the data members of the new object.

Constructors have the same name as the class, and are always public, as in the following example:

```
1 pubic class MyClass{
2
3    public MyClass(){
4       //inside the body of the constructor
5    }
6
7 }
```

The code explained

Line 3: The constructor is public, and has the same name as the class; if the name is different, or if it is declared as private, it will be considered by Java to be an ordinary method and not a constructor.

3.6 Implicit constructors

If you take a look at class Cat you will notice it does not have a constructor, yet we call its constructor when we instantiate it. How is this possible?

This is possible because Java creates a constructor for us automatically if our class does not possess one. This constructor is known as an "implicit" constructor. If Java detects that the class has a constructor, it will not create any constructor.

3.7 Overloaded constructors

A class can have more than one constructor, each one with different arguments. Look at the following class:

```
1 public class Person{
2    public String name;
3    public String surname;
4    public int age;
5
6    //this is the default or no-args constructor
7    public Person(){
8
9    }
10
11   public Person(String n){
12      this.name = n;
13   }
14
15   public Person(String n, String s, int a){
16      this.name = n;
17      this.surname = s;
18      this.age = a;
19   }
20 }
```

The code explained

The Person class has 3 constructors:

1. Line 7: A no-args constructor, which is also known as the default constructor.
2. Line 11: A constructor which takes in one String parameter.
3. Line 15: A constructor which takes in two Strings and an int.

- **Line 12:** The this keyword refers to the current object. In other words, this.name refers to the variable name which is declared in line 2. What is happening in this line is that the constructor receives a String parameter, and then sets the value of the variable name to the value of the parameter.
- **Lines 16 to 18:** The object variables are being set to the parameter values received by the constructor, as above.

The following class creates three instances of class Person, each time calling a different constructor:

```
1 public class TestPerson{
2    public static void main(String[] args){
3
4    Person p1 = new Person();
5
6    Person p2 = new Person("Tom");
7
8    Person p3 = new Person("Tom","Sawyer",14);
9
10   }
11 }
```

The code explained

- **Line 4:** Here we are creating an instance of class Person by calling the default constructor. This new instance will be assigned to p1. When p1 is created, none of its variable are given any values, so its variables will be as follows:

  name: null
  surname: null
  age: 0

- **Line 6:** Here we are creating an instance of class Person by calling the constructor which takes in one String parameter, and pass it the name "Tom". As a result, the values of the object p2 will be:

  name: Tom
  surname: null
  age: 0

- **Line 8:** This line calls the constructor with three arguments, and passes it the Strings "Tom" and "Sawyer" and the integer 14. The newly-created instance p3 will have the following values:

  name: Tom
  surname: Sawyer
  age: 14

3.8 Unit Summary

- An object is a software component which is a combination of data and methods which act on that data, which can be treated as a single unit. An object can be a part of an application or an entire application.
- Object oriented programming is a programming paradigm that uses objects to create computer programs.
- Objects enable reuse of software components because an object written for one application can be reused for other applications. This saves a lot of time and money.
- Java is a strictly object oriented programming language.
- Constructors are class methods that are automatically executed when an object of a given type is created. As their name suggests, constructors play a role in "constructing" objects; they initialised the data members of the new object.

3.9 Review questions

1. Define a class
2. Define an object
3. What is a constructor?

4. Write a Java class called “Shape” which has 3 variables: String colour, int size and String type.
   The class should have a default constructor, and an overloaded constructor which accepts 3 parameters: colour, size and type.

5. Create a class called TestShape with a main () method. In the main method, do the following:

a) Create an instance of class Shape using the default constructor, and populate the 3 variables with a colour, a size and a shape.

b) Create an instance of the Shape class using the overloaded constructor, passing it 3 parameters for colour, size and type.

3.9.1 Answers to Revision Questions and Activities

1. A class is a template for an object. A class instantiated to create objects.
2. An object is a software component which is a combination of data and methods which act on that data, which can be treated as a single unit. An object can be a part of an application or an entire application.
3. Constructors are class methods that are automatically executed when an object of a given type is created. As their name suggests, constructors play a role in “constructing” objects; they initialize the data members of the new object.
4. The code follows:

```
public class Shape{
   String color;
   int size;
   String type;

   public Shape(){
   }

   public Shape(String color, int size, String type){
      this.color = color;
      this.size = size;
      this. type = type;
   }

}
```

5. The code follows:

```
public class TestShape{
   public static void main(String[] args){
   Shape s1 = new Shape();
   s1.color = "green";
   s1.size = 5;
   s1.type = "square";

   Shape s2 = new Shape("blue",6,"circle");
   }
}
```

Unit 4:

Object Oriented Programming – Advanced

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
4.1 Encapsulation	• Display knowledge on the following concepts: ✓ Inheritance ✓ Composition ✓ Polymorphism ✓ Encapsulation
4.2 Encapsulation in Java	• Illustrate the concept of encapsulation
4.3 Inheritance	• Demonstrate an understanding of inheritance
4.4 Inheritance in Java	• Explain Inheritance in Java
4.5 Composition	• Establish what composition is
4.6 Composition in Java	• Define composition in Java
4.7 Unit Summary	• Summarise topic areas covered in unit

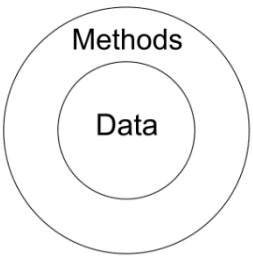
Prescribed Textbook:

- Currie, Edward. Fundamentals of Programming using Java Cengage Learning. ISBN 978-1-84480-451-1

4.1 Encapsulation

When you drive a car, you use the accelerator to control the engine. The engine itself is hidden away from you, and you never need to see it or tinker with it in order to drive the car. We can say that the engine is "encapsulated" under the bonnet of the car.

In a similar way, the data within an object is hidden away, and it is impossible to make any changes to it directly. The only way to modify the data is via methods which are made available by the developer of the class. This is known as "encapsulation".



Bank account objects – an illustration of encapsulation

The benefits of encapsulating data within an object can be explained using the example of a bank account. A typical bank account object will have a variable called *balance*, which will store the amount of money the account holder has in his account. Now imagine if it were possible to directly access and change the value which is in this variable – people would increase it to astronomical amounts and withdraw money that they do not own, and the bank would go bankrupt!

However, the balance *variable* is locked away within the bank account object, and may only be accessed via specific methods which have built-in intelligence to check that no foul play is taking place. So when an account holder goes to an ATM machine to withdraw cash, the ATM machine will typically call a *withdraw()* method which will check to see if the account holder has sufficient funds in his account, as demonstrated by the following pseudo code:

Method withdraw ()

1. Amount = withdrawal amount requested by user
2. Balance = current user's balance
3. If Balance > Amount, dispense the cash and adjust the balance
4. Else If the Balance < Amount, do not dispense cash
5. End

Encapsulation can also be used to keep the data "clean" by only accepting data which fits a specified set of criteria. For example, if there is a variable in an object called *age* which stores the age of a person, it will be legal to store a negative value in it because the variable *age* will typically be declared as an integer, and a variable of type *integer*

can store negative integers as well. But we know that a person's age cannot possibly have a negative value, so we will need to put measures into place to forbid anyone from inserting a negative value.

The solution to this problem, as you may have guessed, will be to encapsulate the variable *age*, and to only allow access to it via a method. This method will then have the task of checking that no negative values are coming through before storing a value into the variable.

4.2 Encapsulation in Java

To illustrate the concept of encapsulation, we've made some modifications to class Cat:

```
1    private String name;
2
3    public String getName(){
4       return name;
5    }
6
7    public void setName(String n){
8       this.name = n;
9    }
```

The code explained

(Please note that the line numbers above do not reflect the actual line numbers from class Cat – they are here to facilitate explanation of the code)

- **Line 1:** Notice that we've replaced the keyword public with private. This essentially "hides" the variable "name" from other classes. The private modifier makes a variable or method visible only from within the same class. In other words, the variable "name" is encapsulated within the class Cat.

- **Line 3:** The getCat()method has only one purpose: to return the value of the variable "name". Because we've declared the variable "name" to be private, it will be inaccessible from any other class such as class UseCat. So the question arises that how will we be able to retrieve the name of the Cat object? The answer is that it will be retrieved by calling the getName() method. Because method getName() is public, it can be accessed from other classes too.

- **Line 7:** As mentioned earlier, variable "name" is hidden within the class Cat, so there is no way to retrieve its value. Similarly, there is no way to directly set its value. Method setName(), which is public, provides a way to set a value for the variable "name". All we need to do is to call method setName() and pass it a String parameter for the name.

These changes to class Cat will necessitate changes to class UseCat:

```
1 public class UseCat{
2    public static void main(String[] args){
3    Cat c1 = new Cat();
4
5    c1.setName("Tixe");
6    c1.colour = "ginger";
7    c1.age = 2;
8    c1.height = 25;
9    c1.weight = 2;
10
11   System.out.println(c1.getName());
12   System.out.println(c1.colour);
13   System.out.println(c1.age);
14   System.out.println(c1.height);
15   System.out.println(c1.weight);
16
17   c1.meow();
18   }
19 }
```

The code explained

Line 5: Notice that we do not call the variable "name" directly here – doing so will result in a compile time error. Rather, we set the name attribute for object c1 by calling its setName() method and passing it a String parameter.

Line 11: In this line we call method getName() rather than reference variable "name" directly. As with the above, it will result in an error if done.

4.3 Inheritance

The concept of inheritance demonstrates the power of Object-oriented programming. As the name suggests, **Inheritance** means that it is possible for one class to *inherit* from another. This is best explained by means of a simple example:

As you know, class Cat will store data about a cat, as shown below:

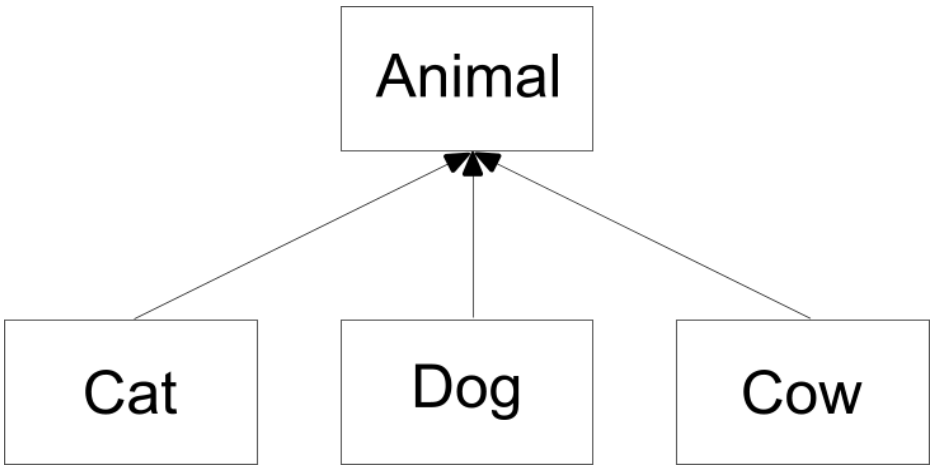
Cat
Variables:
name
colour
age
height
weight
Methods:
eat()
sleep()
meow()
walk()

Similarly, a class Dog will store data about a dog:

Dog
Variables:
name
colour
age
height
weight
Methods:
eat()
sleep()
bark()
walk()

Can you see the similarities? In fact, looking closely at the above classes, you will find that there are actually more similarities than differences between the Cat class and the Dog class. In fact, if you have a Cow class, it will be almost identical as well.

The reason for the similarities is that a cat, a dog and a cow are all animals, and all animals, even though they are different in many ways, have many similarities:

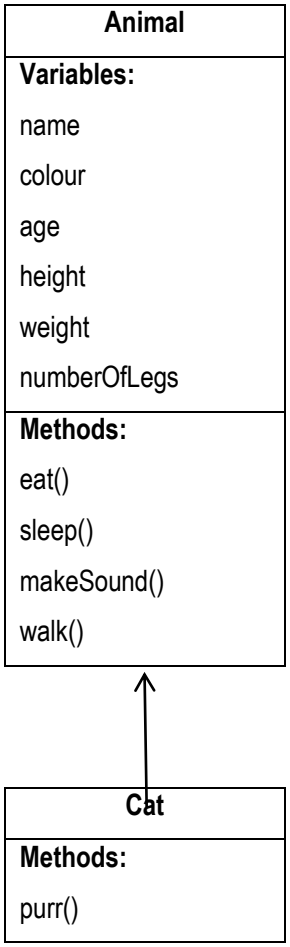


Object-oriented programming allows you to take advantage of these similarities by creating a generic class which includes data which is common to all animals, and then specialising it for each specific type of animal. For example, you can create an Animal class which will be something like the following:

Animal
Variables:
name
colour
age
height
weight
numberOfLegs
Methods:
eat()
sleep()
makeSound()
walk()

The above class will apply to all animals. But what if we wanted to create a class *Cat*, which has a method called *purr()*, which does not exist in class *Animal*?

Bearing in mind that a Cat *is* an animal, we can create a class *Cat* which inherits all the characteristics of class *Animal*, but adds a few of its own characteristics, as shown below:



Can you see the benefits of inheritance? No matter what animal class you create, you will never need to re-implement the variables and methods which are common between them. All you will need to do is to make each animal inherit from class *Animal*, and then implement only the specific characteristics of that animal.

4.4 Inheritance in Java

The following code listing shows the implementation of class Animal in Java. The code is quite straight-forward:

```
public class Animal{
   public String name;
   public String colour;
   public int age;
   public int height;
   public int weight;

   public void eat(){
      System.out.println("Animal eating");
   }

   public void sleep(){
      System.out.println("Animal sleeping");
   }

   public void makeSound(){
      System.out.println("Animal making sound");
   }

   public void walk(){
      System.out.println("Animal walking");
   }
}
```

Now let's take a look at the new version of class Cat:

```
1 public class Cat extends Animal{
2    public void purr(){
3       System.out.println("Cat purring");
4    }
5 }
```

The code explained

Line 1: The extends keyword declares that class Cat is "extending" class Animal; in other words, class Cat is a sub-class or a child class of class Animal.

Line 2 to Line 5: Notice that all the variables and methods of class Cat have been removed! The reason for this is that they have been moved to class Animal; so when class Cat extends class Animal, it actually inherits all those variables and methods, as though they are still in class Cat. This will become clear in the following code:

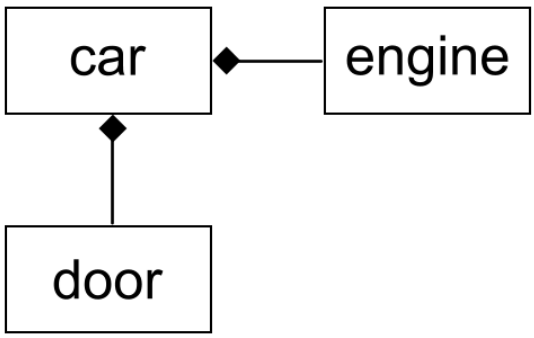
```
1 public class UseCat{
2    public static void main(String[] args){
3    Cat c1 = new Cat();
4
5    c1.name = "Tixe";
6    c1.colour = "ginger";
7    c1.age = 2;
8    c1.height = 25;
9    c1.weight = 2;
10
11   System.out.println(c1.name);
12   System.out.println(c1.colour);
13   System.out.println(c1.age);
14   System.out.println(c1.height);
15   System.out.println(c1.weight);
16
17   c1.walk();
18   c1.purr();
19   }
20 }
```

As you can see, we are calling variables and methods from class Animal against object c1 as though they belong to class Cat. All the variables and methods in lines through 17 belong to class Animal; only method purr() in line 18 belongs to class Cat.

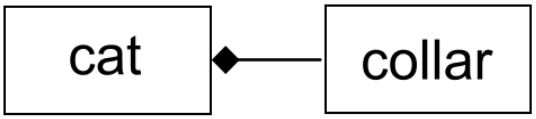
We are allowed to do this because class Cat extends class Animal, and thus inherits all its members.

4.5 Composition

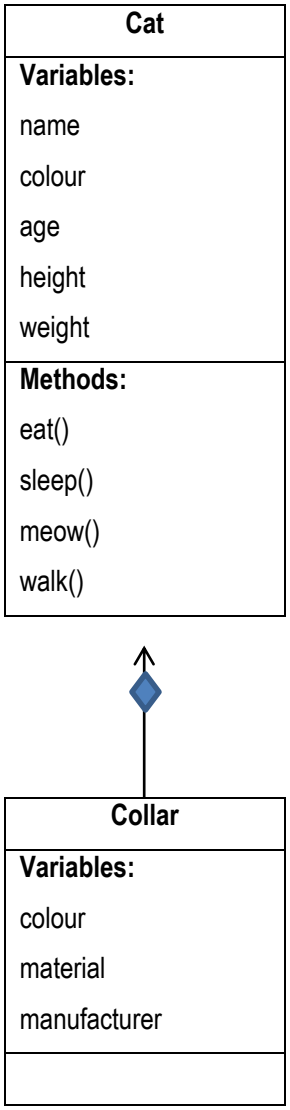
If inheritance represents an *is-a* relationship between two classes, composition represents a *has-a* relationship. Composition represents the case where an object is made up of other objects. To understand this, consider the example of a car which we saw earlier in this chapter. A car is an object, but it is made up of other objects; it has an engine, four wheels, doors, etc. which are objects themselves. This is illustrated in the following diagram:



Similarly, a cat can have a collar, and the collar can have its own characteristics:



The UML diagram for the Cat class is as follows:



4.6 Composition in Java

The above relationship can be expressed in Java as follows:

First we define the class Collar, and save it to a file called Collar.java

```
1 public class Collar{
2    public String colour;
3    public String material;
4    public String manufacturer;
5 }
```

As you can see, class Collar is a very simple class comprising of three String variables. Next, we modify class Cat:

```
1 public class Cat extends Animal{
2    public Collar col = new Collar();
3
4    public void purr(){
5       System.out.println("Cat purring");
6    }
7 }
```

The code explained:

Line 2: We create a variable of type Collar and populate it with an instance of class Collar. This line demonstrates the concept of composition as implemented in Java. Class Cat has an instance of class Collar, so we say that Cat *has-a* Collar.

We will now modify class UseCat to make use of our newly-modified Cat class:

```
1 public class UseCat{
2    public static void main(String[] args){
3    Cat c1 = new Cat();
4
5    c1.name = "Tixe";
6    c1.colour = "ginger";
7    c1.age = 2;
8    c1.height = 25;
9    c1.weight = 2;
10
11   c1.col.colour = "brown";
12   c1.col.material = "leather";
13   c1.col.manufacturer = "Pet King";
14
15   System.out.println(c1.name);
16   System.out.println(c1.colour);
17   System.out.println(c1.age);
18   System.out.println(c1.height);
19   System.out.println(c1.weight);
20
21   System.out.println(c1.col.colour);
22   System.out.println(c1.col.material);
23   System.out.println(c1.col.manufacturer);
24   }
25 }
```

The code explained

- **Lines 11 to 13:** These lines demonstrate how to access the variables of the col object. Notice how we use the dot operator twice, for example: c1.col.colour. This means that we are referencing the variable which is called colour which belongs to object col, which in turn belongs to object c1, as shown below:

  colour belongs to col belongs to c1

Lines 21 to 23: Here we are outputting the values we set in lines 11 to 13 using the same two-dot notation.

4.7 Unit Summary

Encapsulation is a situation where the data within an object is hidden away, and it is impossible to make any changes to it directly. The only way to modify the data is via methods which are made available by the developer of the class.

Inheritance enables programmers to define an "is-a" relationship between a class and a more specialized version of that class. For example, if a Student class extends a Person class, then it should be the case that Student *is-a* Person. The subclass Student builds on the functionality of the superclass Person.

Composition is a relationship between two classes that is based on the aggregation relationship. Object composition is a way to combine simple objects or data types into more complex ones.

4.8 Review questions

1. Create a class called Person that has 5 encapsulated variables: id, firstname, surname, cellphone, email. Create the appropriate getter and setter methods for the variables.

2. Create a subclass of Person called Student that has 2 variables of its own: studentNumber and yearOfStudy. These variables should be declared private with the appropriate getter and setter methods to access them.

3. Create a subclass of Person called Lecturer that has 2 variables of its own: employeeNumber and salary. As in the above scenario, these variables should be declared private with the appropriate getter and setter methods to access them.

4.9 Answers to Revision Questions and Activities

1. The code follows:

```
public class Person {
  private String id;
  private String firstname;
  private String surname;
  private String cellphone;
  private String email;

    public String getId() {
    return id;
  }
  public void setId(String id) {
    this.id = id;
  }

  public String getFirstname() {
    return firstname;
  }

  public void setFirstname(String firstname) {
    this.firstname = firstname;
  }
```

```
  public String getSurname() {
    return surname;
  }

  public void setSurname(String surname) {
    this.surname = surname;
  }

  public String getCellphone() {
    return cellphone;
  }

  public void setCellphone(String cellphone) {
    this.cellphone = cellphone;
  }

  public String getEmail() {
    return email;
  }


  public void setEmail(String email) {
    this.email = email;
  }
}
```

2. The code follows:

```
public class Student extends Person{
   private String studentNumber;
   private int yearOfStudy;

   public String getStudentNumber() {
      return studentNumber;
   }

   public void setStudentNumber(String studentNumber) {
      this.studentNumber = studentNumber;
   }

   public int getYearOfStudy() {
      return yearOfStudy;
   }

   public void setYearOfStudy(int yearOfStudy) {
      this.yearOfStudy = yearOfStudy;
   }
}
```

3. The code follows:

```
public class Lecturer extends Person{
   private String employeeNumber;
   private double salary;

   public String getEmployeeNumber() {
      return employeeNumber;
   }

   public void setEmployeeNumber(String employeeNumber) {
      this.employeeNumber = employeeNumber;
   }

   public double getSalary() {
      return salary;
   }

   public void setSalary(int salary) {
      this.salary = salary;
   }
}
```

Unit 5:

Stacks and Queues

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
5.1 Introduction 5.2 Stack operations	• Explain what a stack is
5.3 Stacks in Java	• Determine when to use stacks
5.4 Uses of stacks	• Create and use a stack in Java
5.5 Queues	• Explain what a queue is
5.6 Queue operations	• Determine when to use queues
5.7 Queues in Java	• Create queue in Java
5.8 Uses of queues	• Use queue in Java
5.9 Unit Summary	• Summarise topic areas covered in unit

Prescribed Textbook:

- Currie, Edward. Fundamentals of Programming using Java Cengage Learning. ISBN 978-1-84480-451-1

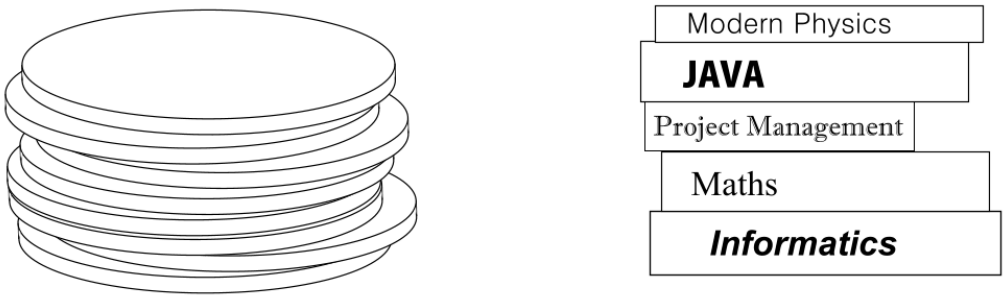
5.1 Introduction

In the last chapter you looked at arrays and linked lists, two very important internal data types. As powerful and useful as arrays and linked lists are, there are times when the software developer will need internal data storage mechanisms which can process data much faster in order to meet the ever-growing speed and efficiency demands on software.

In this chapter you will be introduced to stacks and queues, which are also internal data storage mechanisms, but are quite different from arrays and linked lists in their characteristics and their uses. We will also look at some of their applications in computer programs, so that you will be able to make informed decisions on the optimum data type for each situation.

Stacks

The stack data type derives its name from the fact that it resembles a stack of real-world items. To understand the concept of a stack, we need to think about a stack of objects like coins or books, as shown in the following diagram:



When you place objects onto a stack, the last item you placed will be at the top of the stack. As a result, this item will be the first one to be retrieved from the stack. If you wish to remove the item at the bottom of the stack, you will need to remove all the items on top of it first. For example, if you wish to retrieve the Java book from the stack in the diagram, you will need to remove the “Modern Physics” book first. Similarly, to remove the “Maths” book, you will need to remove the “Modern Physics”, “Java” and “Project Management” books.

In essence, we can say that a stack allows us to insert and remove objects from one place only: from the top of the stack. There is no way to retrieve an item directly from the middle of the stack. A stack data type is identical to a real –life stack in this sense. Using a stack, we “pile up” data objects at the top of the stack, and then we retrieve them one-by-one from the top. Unlike arrays and linked lists where we use the index number of an item to retrieve or remove it, there is no way to reference an item in the middle of the stack – we have to remove each item from the top of the stack in the reverse order in which it was placed in the stack. In other words, the last item in the stack is always the first to be removed.

Because of this characteristic of stacks, we refer to stacks as “Last-in first-out” data types. The definition of a stack follows:

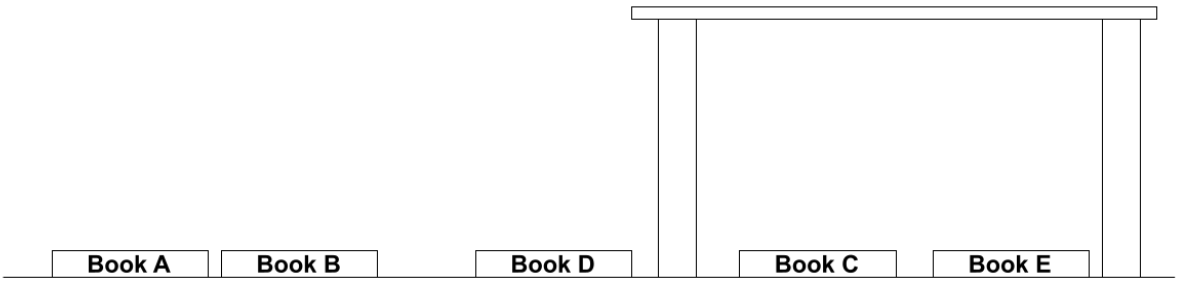
“A stack is a data structure in which the elements are added and removed from one end only; it is a Last-in First-out (LIFO) structure.”

5.2 Stack operations

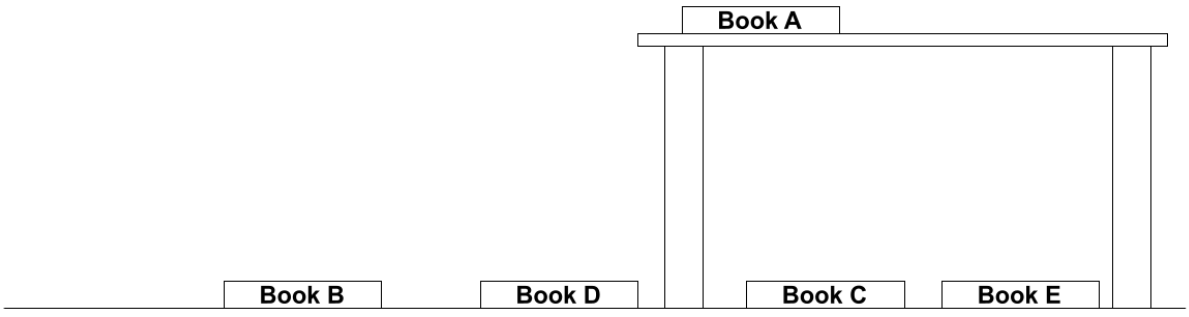
Now that you know what a stack is, let’s look at some of the operations which can be done on a stack:

- Push: the push operation adds an element to the top of a stack
- Pop: the pop operation removes (pops) the element at the top of the stack
- Peek: the peek operation retrieves the topmost element without removing it

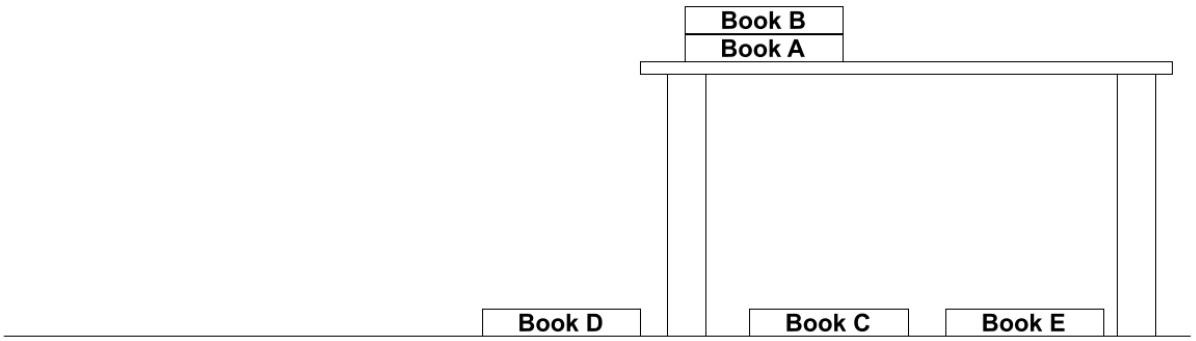
Let’s take a look at an example. Consider a scenario where we have a few books lying on the floor which need to be stacked on a table, as depicted below:



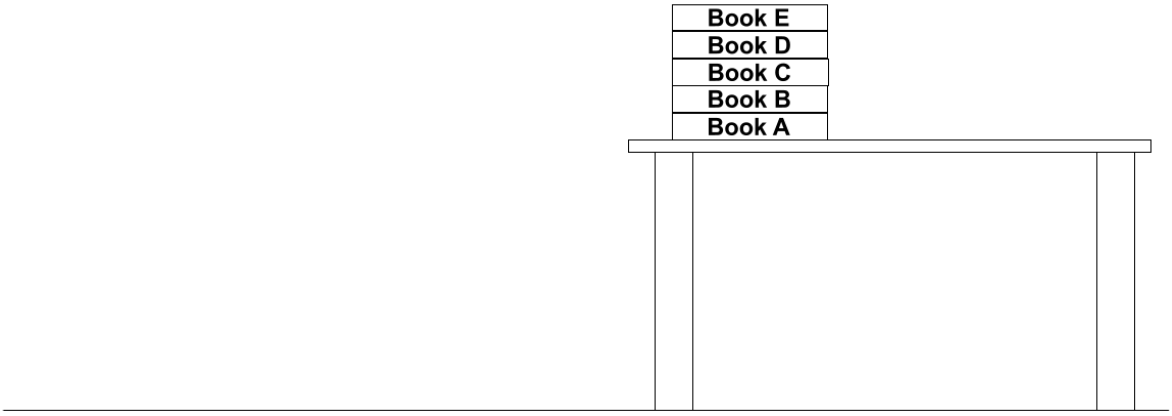
First, we add book A onto the stack by calling push(A). After this operation is complete, the stack looks as follows:



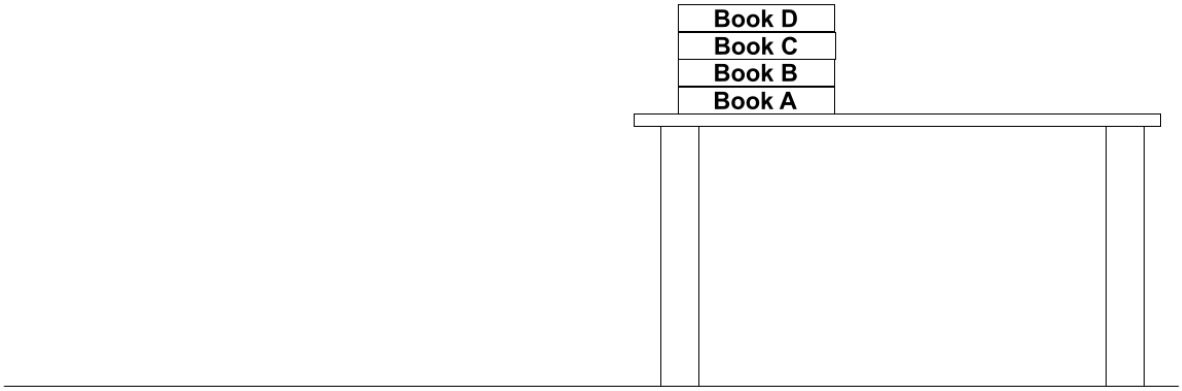
First, we add book B onto the stack by calling push(B). The result is:



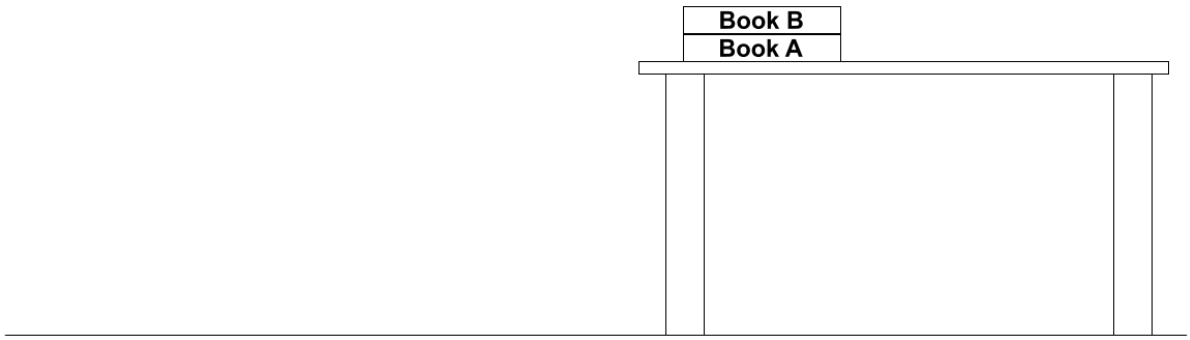
The operations push(C), push(D) and push(E) will result in the following:



Now we call the pop() operation. This will result in the top-most book, book E to be removed from the stack:



Calling the peek() operation will yield book D, but there will be no change in the stack. Calling pop() two more times will yield the following:



Activity 5.1

What will be the outcome of the following stack operations?

1. Empty stack
2. Push(45)
3. Push(23 - 12)
4. Pop()
5. Push(56 + 5)
6. Peek()
7. Push(5 * 7)
8. Pop()
9. Push(8 * 2 + 6)
10. Push(6 * 12)
11. Peek()
12. Pop()
13. Pop()

Now that you understand what a stack is, and how it works, let's look at how stacks are actually implemented in Java and other programming languages.

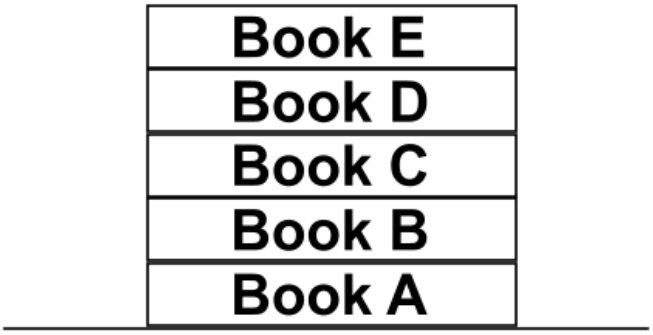
5.3 Stacks in Java

The following code demonstrates how a stack is implemented in Java. The code implements the example shown above.

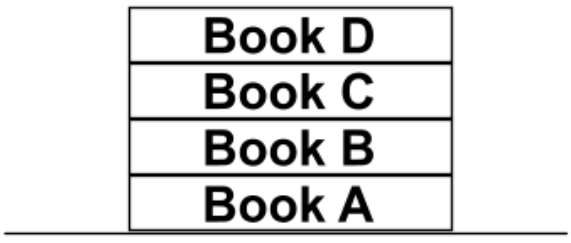
```
1 import java.util.Stack;
2
3 public class StackTest{
4    public static void main(String[] args){
5       Stack s1 = new Stack();
6       s1.push("Book A");
7       s1.push("Book B");
8       s1.push("Book C");
9       s1.push("Book D");
10      s1.push("Book E");
11      s1.pop();
12      System.out.println(s1.peek());
13      s1.pop();
14      s1.pop();
15      System.out.println(s1.peek());
16   }
17 }
```

The code explained

- **Line 1:** In order to use a stack we need to import the java.util.Stack class which provides all the methods necessary for working with stacks.
- **Line 5:** Here we are creating an instance of the Stack class and assigning it to the variable *s1*. In essence we are creating an empty stack.
- **Line 6 to Line 10:** In these lines we are populating our stack with five String elements, "Book A" through "Book E". The result is illustrated below:



- Line 11: This line removes "Book E" from the stack, resulting in the following:



- **Line 12:** This line takes a peek at what the topmost element in the stack is, without removing it. The result will be "Book D", which will be output to the screen.
- **Line 13 and Line 14:** The two topmost elements in the stack, that's is "Book D" and "Book C", are removed. The result will be:

Line 15: This line will output "Book B" to the screen.

Activity 5.2

Write a Java program to implement the stack in the previous exercise.

5.4 Uses of stacks

You can use stacks in any situation where you need to quickly store large amounts of data and to retrieve them in the reverse order. In this section we will describe some of the major uses of stacks in computer programs.

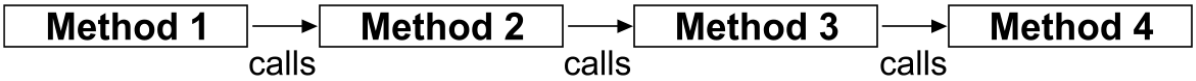
The Undo button

We've all used the "Undo" button in various programs such as word processors and spread sheet applications. When you click the undo button while you are in a program, it reverses the last action that you had performed. Clicking the Undo button again reverses the action done before that; and clicking it a third time reverses the one before that, and so forth, until there are no more actions left to undo.

This behaviour might seem familiar to you by now, because it resembles a stack very closely. In fact, this is one of the places where stacks come in handy – to record each activity the user does while in the program.

Method chaining

Often in your computer programs you will encounter method chaining: this is where a method is called, and before it completes, that method calls a second one, the second calls a third before completing, and so forth, as shown in the diagram below:

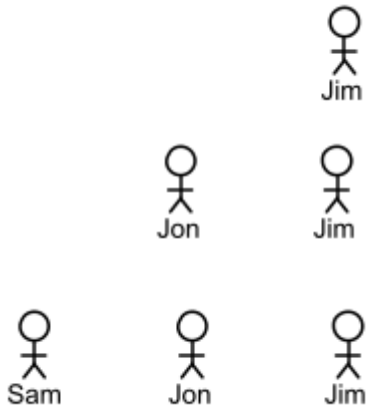


In the above scenario, your program will use a stack to keep track of the methods being called. The first method will be placed onto the stack; the second method will be placed above it, followed by the third and then the fourth.

When the fourth method completes execution, it is removed from the stack, and execution returns to the third; when the third method completes, execution returns to the second, and lastly the first.

5.5 Queues

As its name suggests, a queue data structure resembles a queue in real life. Think about a queue at a supermarket, for example: the first person to arrive will always be the first to be processed and to leave. In other words, people will leave the queue in the order in which they arrived. Look at the diagram below:



Jim is the first to join the queue, followed by Jon and then Sam. As a result, the first person to leave the queue will be Jim, then Jon, and finally Sam.

Queue data structures work in exactly the same way – elements are retrieved from the queue in the order in which they were added. It is for this reason that a queue is known as a first-in-first-out (FIFO) data structure.
A queue will hence be defined as follows:
"A queue is a data structure in which the elements are added at one end, called the rear, and deleted from the other end, called the front; it is a first-in-first-out (FIFO) data structure."

5.6 Queue operations

Now that you know what a queue is, let's look at some of the operations which can be done on a queue:

- Add: the add operation adds an element to the front of a queue
- Remove: the remove operation removes the element at the front of the queue
- Peek: the peek operation retrieves the front element without removing it

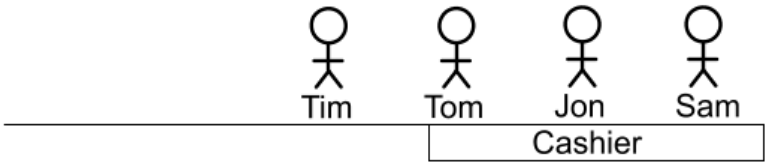
You will understand this better by means of an example. Let's say we have a group of people doing their shopping in a supermarket. As they complete their shopping, they arrive at the cashier to pay for their things:

First we will add Jim to the queue by calling **add(Jim)**:

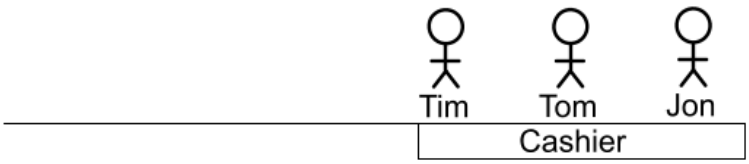
The commands **add(Sam)** and **add(Jon)** will place Sam behind Jim in the queue, and Jon behind Sam, as shown below:

Next, we call **remove()**. This will remove Jim from the queue and place Sam at the front of the queue:

Calling the **peek()** method will tell us that Sam is at the head of the queue, but will not remove him from the queue. Next, we call **add(Tom)** and **add(Tim)**, which will result in the following scenario:



Calling **remove()** once will remove Sam from the queue:



Calling **remove()** three times will empty the queue:

Activity 5.3

What will be the result of the following queue operations?

add(9+5)

add(5*3)

add(7+3)

remove()

add(8/2)

add(8+5)

peek()

remove()

remove()

add(5+4)

5.7 Queues in Java

The following code demonstrates how a queue is implemented in Java. The code implements the "supermarket" example shown above.

```
1 import java.util.Queue;
2 import java.util.LinkedList;
3
4 public class QueueTest{
5    public static void main(String[] args){
6       Queue q1 = new LinkedList();
7       q1.add("Jim");
8       q1.add("Sam");
9       q1.add("Jon");
10      q1.remove();
11      System.out.println(q1.peek());
12      q1.add("Tom");
13      q1.add("Tim");
14      q1.remove();
15      q1.remove();
16      q1.remove();
17      q1.remove();
18   }
19 }
```

The code explained

- **Line 1 and Line 2**: Here we import the necessary package's, which are the Queue interface the LinkedList class. The reason we need the LinkedList class is because our queue will be implemented as a LinkedList.
- **Line 6**: Here we are creating out queue, which is actually an LinkedList instance.
- **Line 7**: The LinkedList class has a add() method, so we use the **add()** method in its place. The **add()** method performs the exact same function as the **push()** method.
- **Line 10**: The LinkedList class does have a remove**()** method, so we use its **remove()** method to do the exact same thing.
- **Line 11**: The **peek()** method simply tells us which element will be removed with the next call to **remove()** without removing the element from the queue.

Activity 5.4

Implement the queue in exercise previous in Java

5.8 Uses of queues

Queues are used mostly by operating systems to schedule certain tasks which need to be executed in a first-come-first-served order. Some of the practical applications of queues are:

1. Sending requests to a single shared resource such as a printer, disk or the CPU of the computer. For example, if multiple printing jobs are sent to a printer, these jobs are placed into a queue in the printer's memory and processed in a FIFO manner.

2. Call centre phone systems will use a queue to hold people in line until a service representative is free. As calls are attended to, they will be "popped" from the queue.

3. MP3 players, portable CD players and iPods will place the user's playlist into a queue. The tracks are then played in the order in which they were inserted into the queue.

5.9 Unit Summary

A stack can be defined as a data structure in which the elements are added and removed from one end only; it is a Last-in First-out (LIFO) structure.

Some of the operations which can be done on a stack are:

- Push: the push operation adds an element to the top of a stack
- Pop: the pop operation removes (pops) the element at the top of the stack
- Peek: the peek operation retrieves the topmost element without removing it

Stacks can be created in Java by instantiating the Stack class.

Stacks have numerous uses in computer programs, such as in the "Undo" button and in method chaining.

A queue can be defined as a data structure in which the elements are added at one end, called the rear, and deleted from the other end, called the front; it is a first-in-first-out (FIFO) data structure.

Some of the operations which can be done on a queue are:

- Add: the add operation adds an element to the back of a queue
- Remove: the remove operation removes the element at the front of the queue
- Peek: the peek operation retrieves the front element without removing it

A queue can be implemented in Java as a LinkedList.

Queues have many uses in computer programs, such as queuing up printing jobs in the memory of a printer, and creating a playlist in an MP3 player or an iPod.

5.10 Review questions

1. Differentiate between a stack and a queue.

2. Study the operations below:

```
push(3*4)
push(7+8)
pop()
push(6*7)
peek()
pop()
push(9*3)
```

2.1 What would be the outcome if the above operations were to be performed on a stack?

2.2 What would be the outcome if the above operations were to be performed on a queue?

3.1. Implement the above as a stack in Java.

3.2 Implement the above as a stack in Java.

5.11 Answers to Revision Questions and Activities

1. A stack is a data structure in which the elements are added and removed from one end only; it is a Last-in First-out (LIFO) structure.

A queue is a data structure in which the elements are added at one end, called the rear, and deleted from the other end, called the front; it is a first-in-first-out (FIFO) data structure.

2.1 The outcome would be 27 12

2.2 The outcome would be 27 42

3.1 The Java code follows:

```
1    import java.util.Stack;
2
3    public class StackTest{
4       public static void main(String[] args){
5          Stack s1 = new Stack();
6          s1.push(3*4);
7          s1.push(7+8);
8          s1.pop();
9          s1.push(6*7);
10         System.out.println(s1.peek());
11         s1.pop();
12         s1.push(9*3);
13      }
14   }
```

3.2 The Java code is as follows:

```
1 import java.util.Queue;
2 import java.util.LinkedList;
3
4 public class QueueTest{
5    public static void main(String[] args){
6       Queue q1 = new LinkedList();
7       q1.add(3*4);
8       q1.add(7+8);
9       q1.remove();
10      q1.add(6*7);
11      System.out.println(q1.peek());
12      q1.remove();
13      q1.add(9*3);
14   }
15 }
```

Unit 6:

Files

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
6.1 The Problem with Internal Data Types	• Differentiate between a data file and a database • Understand the concept of Primary Key
6.2 What are data files?	• Classify what data files are
6.3 Sequential file organisation	• Illustrate knowledge on sequential file organisation
6.4 Advantages of sequential files	• Outline the advantages of sequential files
6.5 Disadvantages of sequential files	• Examine the disadvantages of sequential files
6.6 Direct access files	• Explain direct access files
6.7 Advantages of direct access files	• Determine the advantages of direct access files
6.8 Disadvantages of direct access files	• Determine the disadvantages of direct access files
6.9 Indexed files	• Establish what Indexed files are
6.10 Advantages of indexed files	• Classify the advantages of indexed files
6.11 Disadvantages of indexed files	• Classify the disadvantages of indexed files
6.12 Unit Summary	• Summarise topic areas covered in unit

Prescribed Textbook:

- Currie, Edward. Fundamentals of Programming using Java Cengage Learning. ISBN 978-1-84480-451-1

6.1 The Problem with Internal Data Types

The data types you have been exposed to thus far, are fine for storing data temporarily in the memory of the computer, but they all have one major drawback: when the program exits or the computer shuts down, all data will be lost. The result is that every time you need to work with a data set, you will need to retype it in. This can be cumbersome for even modestly-sized data sets, let alone large data sets with hundreds or thousands of elements.

Often you will want to store data permanently on a storage device in such a way that the data is not lost when the program exits. Then, whenever you need the data, you can retrieve it, use it, and then store it back into the storage medium such as a hard disk, a DVD, a flash memory stick, magnetic tape or an SD card. There are essentially two ways to store data permanently:

1. A database
2. A file system

A database is an efficient and powerful way to store, manipulate and retrieve large amounts of data. However, databases are quite complex, and require a totally separate discussion; as such, they will be discussed in detail in the Informatics 2A module.

Another way to persist data is by storing it in data files. Although files are not suitable for large amounts of data, they can be more suitable than databases when there is a small amount of data to store. This will be particularly true in systems where resources are limited, such as in mobile devices.

6.2 What are data files?

A data file is any file that you store on your computer. By this definition, a word processing document, a spread sheet, an image and a text file can all be described as data files, simply because they store data; for example, the word processor document and the text file will store text, and the spread sheet will typically store facts and figures. When it comes to storing data from your applications, you may use data files. Data files are essentially ordinary text files which store data in ways which make it a little easy for your application to retrieve and manipulate. There are three types of data files:

1. Sequential files
2. Direct files
3. Indexed files

6.3 Sequential file organisation

Sequential files are files which store data in a sequential manner, that is, one after another. The data in these files can only be accessed from beginning to end sequentially. As an example, to read or write the 1000th record, you must first read the 999 records that come before it; there simply isn't another way to retrieve that particular record. Sequential files work a bit like songs on a cassette tape. If you want to listen to the last song on the tape, you must wind through all the songs that come before it. Sequential files are usually sorted into some order, e.g. ascending

order of customer number. Note, however, that the sorting is done by an application, and then the data is stored into the file.

Sequential File organisation is typically used when a large number of records are processed at the same time, e.g. a wages file.

In order to understand how data is stored in, and retrieved from, a sequential file, consider the following example: Assume that a business stores the names of its salespeople in a sequential file, in alphabetical order of the surname, as shown below:

Adams, John
Bartlett, Ed
Chow, Lucy
Clancy, William
Johnson, Michael
McAdam, Jenny
Zulu, Sipho

Now assume that a new salesperson by the name of Mary Franklin has joined the company. Mary's name will have to be stored in the file, and will be stored between William Clancy and Michael Johnson.

Because the file in use is a sequential file, there is no way to insert Mary's name directly into the correct position. Instead, the following steps will have to be carried out:

1. All the names in the existing file will be retrieved and stored into a data type like an array.
2. Mary Franklin's name will be stored in a random position in the array, probably at the end of the array.
3. The array will be sorted by alphabetical order of the surname, so that Mary's name falls into place between William Clancy and Michael Johnson.
4. The entire array will then be written to the file, overwriting the existing contents.

   As you can see, in order to add a single record to the file, we had to read and rewrite the entire file. In essence, we need to create an entirely new file simply to insert a single record. Clearly, sequential files can become quite cumbersome to work with.

6.4 Advantages of sequential files

Despite their shortcomings, sequential files do have a few advantages. These are summarised below:

- Simple file design: sequential files are very simple by design, and do not require extensive planning and designing to create.
- Very efficient when all or most of the records must be processed e.g. Payroll
- Very efficient if the data has a natural order, such as alphabetical or numerical order.

- Can be stored on inexpensive devices like magnetic tape.

6.5 Disadvantages of sequential files

Some of the major disadvantages of sequential files are listed below:

- The entire file must be processed even if a single record is to be changed.
- All transactions have to be sorted before processing.
- A completely new file has to be created if even a single new record is added or changed.
- Overall processing is slow.

6.6 Direct access files

Direct file organisation addresses the problems associated with sequential files. Direct file organisation is a type of file organisation in which each data entry can be accessed directly, without having to go through every entry that comes before it. Direct access files are also known as random access files, because the entries can be accessed in a random order – there is no way to predict in which order they will be accessed. This is in contrast to sequential files, where the order of access is always the same: from the beginning to the end.

To access an entry in a direct access file, you open the file, seek a particular location, and read from or write to that file. To understand the difference between a sequential file and a direct access file, think of a scroll and a book. In order to access specific content in a scroll, you will need to unwind the entire scroll up until that content. You cannot access the content directly. This resembles a sequential file.

On the other hand, if you need to access specific content in a book, you can look up the content in the table of contents or in the index and then flip to the page that the content is on. This is how direct access files work.

6.7 Advantages of direct access files

The advantages of direct access files can be summarised as follows:

- Any record can be directly accessed.
- Speed of record processing is very fast.
- Up-to-date file because of online updating.
- Concurrent processing is possible.

6.8 Disadvantages of direct access files

The disadvantages of direct access files can be summarised as follows:

- More complex than sequential
- Does not fully use memory locations
- More security and backup problems

6.9 Indexed files

Indexed files offer another way to access entries in a file directly without having to iterate through every entry preceding it. Indexed files make this possible via indexes: unique reference numbers that identify each entry uniquely.

Each entry in an indexed file contains reference numbers, like employee numbers, that identify a record in relation to other records. These reference numbers are unique to each entry, and are called the “primary keys”. If you know the primary key of an entry, you can access the entry directly by specifying the record's primary key.

Take a look at the following table:

Emp_Number	Name	Surname
1001	Tom	Smith
1002	Jane	Harris
1003	Peter	Jones

In the above table the **Emp_Number** column is the unique primary key. By knowing this number you will be able to access the employee’s record directly. For example, if you are looking for information about Peter Jones, you only need to know his employee number, which is 1003. By specifying the employee number, you can access his information.

6.10 Advantages of indexed files

The advantages of indexed files can be summarised as follows:

- Easy, efficient data access via the indexes
- Faster than sequential files for storing and retrieving data

6.11 Disadvantages of indexed files

Indexed files may present just one disadvantage, which is that they require more disk storage space than sequential files. However, considering the massive storage capacities of modern storage media, this may be insignificant.

6.12 Unit Summary

- In order to persist data after a program exits or after the computer shuts down, the data may be stored in a database or a file.
- There are three types of file storage systems:
  - Sequential files
  - Direct access files
  - Indexed files

- Sequential files are files which store data in a sequential manner, that is, one after another.
- Direct file organisation, also known as random access file organisation, is a type of file organisation in which each data entry can be accessed directly, without having to go through every entry that comes before it.
- Indexed files access entries in a file via indexes, which are unique reference numbers that identify each entry uniquely.
- Indexed files allow quick access and editing of data.

6.13 Review questions

1. Explain what a direct access file is.
2. What are the advantages of sequential files?
3. In which scenario will a sequential file be a better solution than an indexed file?
4. What is a primary key?

6.14 Answers to Revision Questions and Activities

1. A direct access file is a type of file in which each data can be accessed directly, without having to go through every entry that comes before it.

2. The advantages of sequential files are as follows:
   a. Simple file design: sequential files are very simple by design, and do not require extensive planning and designing to create.
   b. Very efficient when all or most of the records must be processed e.g. Payroll
   c. Very efficient if the data has a natural order, such as alphabetical or numerical order.
   d. Can be stored on inexpensive devices like magnetic tape.

3. Sequential files are well-suited for bulk data processing. If you have a scenario where there is a lot of data, and all or most of it needs to be processed, then a sequential file method of storage will be ideal.

4. A primary key is a unique identifier for each record in a data file.

Unit 7:

Ethical and Professional Issues in Information Technology

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
7.1 Introduction 7.2 IT Technicians Fired After Reporting Child Porn	• Display IT ethical responses to the scenarios explained

Prescribed Textbook:

- Currie, Edward. Fundamentals of Programming using Java Cengage Learning. ISBN 978-1-84480-451-1

7.1 Introduction

There is no doubt that the great advancements in information technology that we have seen in recent decades have brought with them massive ethical, social and professional challenges. Computers, mobile devices, the internet, email, instant messaging; all of these technologies have had a tremendous impact on our lives, but in the wrong hands they have been, and continue to be, used for many illegal and unethical purposes.

In this chapter we will look at some of the social and ethical issues affecting information technology professionals and information technology users. We will focus primarily on two issues:

1. Is there any justification to intrude into the privacy of another person?
2. Should IT professionals disclose information about illegal activities that they've detected on client's computers?

Many of the issues relating to information technology cannot be labelled as good or bad, ethical or unethical; they are open for debate. Take, for example, the act of digitally spying on people by intercepting their emails, text messages and telephone calls – will this be considered unethical and illegal? Almost everyone will agree that it is indeed unethical and illegal. However, if these acts are done in the interest of national security, the matter is no longer as clear; many will argue that since it is done for good, it is acceptable; others will condemn it, stating that the acts are illegal and unethical no matter who does it.

Consider a slightly different scenario: an IT professional working on a client's system discovers that the client is involved in illegal activities. Should they report this to the relevant authorities, or will it be better to ignore it, for fear of reprisals?

The case studies below deal with these very same issues. Read them carefully, and answer the questions that follow.

Echelon—Top-Secret Intelligence System

Echelon is a top-secret electronic eavesdropping system that is managed by the U.S. National Security Agency (NSA) and known to be used by the intelligence agencies of England, Canada, Australia, and New Zealand. It can intercept and decrypt almost any electronic message sent anywhere in the world via satellite, microwave, cellular, or fiber-optic telecommunications, including radio and TV broadcasts, phone calls, computer-to-computer data transmission, faxes, and e-mail. It may have been in operation since the 1970s, but it wasn't until the 1990s that journalists, using the FOIA, were able to confirm its existence and gain insight into its capabilities.

Although Echelon is the world's largest and most sophisticated surveillance network, it is by no means the only one. Russia, China, Denmark, France, the Netherlands, and Switzerland operate Echelon-like systems to obtain and process intelligence by listening in on electronic communications.

Which electronic transmissions are captured and what Echelon can do with the messages is subject to much conjecture. Even if all electronic messages worldwide were unencrypted, finding the messages that warranted further attention would be an enormous, computer-intensive task. As a result, Echelon probably targets communications to and from specific people and organisations rather than trying to assimilate all electronic messages. Thus, some subset of all possible messages is forwarded to the massive U.S. intelligence operations at Fort Meade, Virginia, where powerful computers look for code words or key phrases. Intelligence analysts peruse any conversation or document flagged by the system, and significant messages are then forwarded to the agency that requested the information.

A number of intelligence satellites in orbit are used to detect signals that normally dissipate into space—radio signals, mobile phone conversations, and microwave transmissions. In addition, at least six ground-based stations throughout the world monitor the communication satellites of Intelsat, the world's largest service provider of commercial satellite communications.

Computer processing speeds and the science of speech recognition probably are not advanced enough for a real-time global listening system capable of transcribing the hundreds of thousands of calls that are happening at any instant in time. However, Echelon is capable of voice-pattern matching and can identify speakers if their voice patterns are stored in its database. Also, it employs recording systems that can automatically trigger tape recordings based on "hearing" key words.

Echelon's special software and speech recognition technology can convert any audio communications into formatted, searchable text. A half-hour broadcast can be processed and stored in searchable format in 10 minutes. Currently, the software understands only American English, but the CIA is enhancing it to handle Chinese and Arabic. Other Echelon software alerts intelligence analysts any time a new page goes up on a Web site of interest. CIA personnel use special software to perform searches in English of Web sites developed in Chinese, Japanese, Russian, and eight other languages. The software then translates the text of the Web site into English.

This immense, highly sophisticated surveillance system apparently operates with little oversight, and the various agencies that run Echelon have provided few details about legal guidelines governing the project. In fact, the governments of the countries believed to be involved have not officially acknowledged Echelon's existence. Because of this, there is no way of knowing its true capabilities and exactly how it is used.

Echelon intercepts both sensitive government data and corporate information. It also provides the opportunity to illegally spy on private citizens. It is no wonder that privacy advocates are upset with the secrecy surrounding the system and its great potential for misuse. They feel that Echelon can be directed against virtually any citizen in the world with the full knowledge and cooperation of their government.

In the U.K., Echelon has already been accused of spying on organisations such as Amnesty International—an international agency that seeks to ensure fair and prompt trials for political prisoners and that opposes human rights abuses. In addition, in September 1999, the European Union released a report that accused Echelon of intercepting confidential company information and divulging it to favoured competitors to help win contracts. The report alleged that Airbus Industries of France lost valuable contracts because information intercepted by Echelon was forwarded to the Boeing Company to help it obtain a competitive advantage.

In the United States, the ACLU and others are concerned that Echelon may be used without a court order to intercept communications involving Americans. The Foreign Intelligence Surveillance Act prohibits interception of certain communications for intelligence purposes without a court order, unless the Attorney General certifies that certain conditions are met. These conditions include a limitation that "there is no substantial likelihood that the surveillance will acquire the contents of any communication to which a United States person is a party."

Echelon supporters know that communications surveillance is successful in gathering enemy intelligence and was a key to the success of the allied military effort in World War II. They also argue that tragedies such as the attacks on September 11, 2001, and the Oklahoma City bombing are proof that surveillance systems are necessary to warn authorities and combat terrorism. Within that context, the United States agreed to share highly classified material from Echelon with the Spanish government to aid in its battle against the Basque separatist group ETA. As a result, the Spanish are now receiving decoded intercepts relating to the ETA's plans for terrorist operations.

Questions:

1. Are you for or against the use of Echelon for eavesdropping on electronic communications? Why or why not? Is your opinion affected by the terrorist attacks of September 11?
2. Develop a set of plausible conditions under which the directors of Echelon would authorise using the system to listen to specific electronic communications.
3. What sort of expanded or new capabilities might Echelon develop in the next 10 years as information technology continues to improve? What additional privacy issues might be raised by these new capabilities?

7.2 IT Technicians Fired After Reporting Child Porn

In 2002, Dorothea Perry was an employee of Collegis, an IT outsourcing firm that implemented and supported software for colleges and universities. She worked on the IT help desk at New York Law School, a private law school in lower Manhattan founded in 1891. When she arrived at her desk on a Sunday afternoon in June 2002 she had a voice mail from Dr. Edward Samuels. Dr. Samuels, a highly respected professor at the school for over a quarter century, asked that she check his computer for a possible virus. Perry went to Dr. Samuels' empty office to check things out but was unable to fix the problem. She left a message for her co-worker. Robert Gross—also a Collegis employee—to take a look at the computer on Monday. Gross examined the machine, but he, too, was

unable to resolve the problem, so Perry and Gross, following their employer's guidelines, gave the professor a "loaner" machine while they worked on his.

Again, following their employer's procedures, they began backing up Samuels' files to the school's network so that they could transfer them to Samuels' loaner machine. In the process, they uncovered more than two dozen photos of nude, young girls—several in very sexually provocative poses. Perry immediately reported the incident to Collegis's executive director who in turn alerted her liaison at the school, Fred DeJohn. DeJohn met with Dean Richard Matasar three days later to discuss the photos and determine the appropriate next steps. (Possession of child pornography is both a federal and a New York state crime.) The New York Law School sought advice from counsel, alerted authorities, and cooperated fully in the investigation. A subsequent search of Dr. Samuels' apartment by the New York Police Department uncovered more man 100 000 similar photos—many of them depicting violent sex acts involving young girls. On advice of the district attorney, Dean Matasar did not alert Dr. Samuels of the findings of legal authorities.

Following the investigation, Dr. Samuels was arrested in August 2002. Dean Matasar sent an e-mail to New York Law School faculty, staff, and students, saying "I'm saddened to report to you that I learned this afternoon that our colleague, Professor Edward Samuels, was arrested on charges relating to possession of child pornographic images. The Law School has placed Professor Samuels on paid administrative leave so that he may attend to his defence ... Our hearts go out to Ed and his family as they face the difficult time ahead."

Dr. Samuels was protected by tenure and remained on paid administrative leave for months. Some of the faculty and senior staff were critical of Dean Matasar for turning the photos over to the authorities without first warning Dr. Samuels. Some claimed that Dr. Samuels had committed a victimless crime and there was no proof that he had done anything other than view the photos. Even Dean Matasar said, "When there's no purchase or sale of these materials, I don't know. As a lawyer, I'm ambivalent on these issues."

In April 2003, Samuels pled guilty to 100 counts of possessing Child pornography. The sentencing judge received 20 testimonial letters expressing great respect for Dr. Samuels' integrity as a professor and stating that the incident did not affect their respect for his professionalism. In June, Samuels was sentenced to six months in jail and 10 years' probation. Samuels eventually served four months.

Perry and Gross were placed on probation (or job-performance issues shorty alter they reported finding the photos. In October 2002, four months after the discovery, Gross and Perry were fired. Gross had worked at the school just over a year, and his evaluation three months prior to the incident rated him as "fully competent plus." Perry had worked at the school for 12 years, and her previous performance appraisal rated her work as "excellent."

Dean Matasar stated that he had asked Collegis to improve the help desk and that the firing of Gross and Perry was likely a result of this request. Matasar claims he received countless complaints from students and faculty at the school about poor service. The terminations came just prior to the renewal of Collegis's multimillion dollar contract with the school.

Tom Huber, CEO of Collegis, defended his firm's actions, saying, "Employment of the technicians ended due to issues completely unrelated to this isolated incident. It is the policy of Collegis to treat all of its employees in a fair and impartial manner, and it would never dismiss an employee for doing the right thing. Our employees are instructed and expected to uphold the law and to enforce the policies of the client institutions they serve.

Perry and Gross claimed they had done nothing wrong. They filed a $15 million lawsuit that charged New York Law School and their employer, Collegis, with retaliation for their reporting of the incident. In their suit, the two contend that the child pornography was a form of sexual harassment that violated New York City's "human rights" law. Their attorney stated that Perry and Gross "had a right to a workplace free from degrading and offensive pornography." In October 2004, a judge dismissed some of the claims and ruled that the action did not constitute sexual harassment. However, the judge stated: "That they were terminated shortly after they reported finding child pornography, and despite unblemished employment records, raises a substantial question as to whether the defendants were fired for reporting the professor's alleged criminal activities."

Gross and Perry, after several months, found new jobs in IT support. Gross settled the Suit, Perry continued to fight for years, and in October 2008, six years after she was fired, the Appellate Division of the Supreme Court of Now York ruled that Perry's employment with Collegis was terminable at will and that Perry had no tenable claim that New York Law School acted for the sole purpose of harming her.

Questions to Consider

1. What message is sent to IT workers by the actions of New York Law School and Collegis - even if unrelated job-performance issues justified their actions in firing Gross and Perry?

2. Since this incident, a number of states have enacted laws that require workers to report immediately any child pornography found while servicing equipment. Most of the laws state that a worker who reports such a discovery is immune from any criminal, civil, or administrative liability. Failure to report the discovery can result in a fine, imprisonment, or both. Do you think such laws will encourage reporting? Why or why not?

Conclusion

In this module you studied various mechanisms for storing data within your computer program as well as for storing data externally on storage media such as hard disks, magnetic tape, DVD's etc. You should now have a sound understanding of these data types; their characteristics, their usage scenarios, as well as the way to implement them in an object oriented programming language like Java.

You should at this stage be fully aware of the benefits of utilising the correct data types for storing data, and you should be equally aware of the dire consequences to your program should inappropriate data types be utilised.

You have learned to use arrays and Linked Lists, two popular and useful data storage types; you have covered object oriented programming, and should now be able to comprehend why it is so popular among software developers; you have learned about advanced object oriented concepts like inheritance and composition, and how these save a lot of time and money; you covered the concepts of stack and queues, and should be aware of their correct application in computer programs; you were also exposed to the concept of external data storage via data files.

In the next module, Informatics 2A, you will take your understanding of external data storage much further via the study of databases. Databases are without doubt the backbone of the information age, because it is in various databases around the world that the bulk of the world's data is stored.

We sincerely hope that you found this module informative and stimulating, and we look forward to working with you in the next module.

MODULE REVIEW AND FEEDBACK FORM

Name of Student:

Programme:

Module:

MANCOSA is committed to continuous upgrading and improving the content and presentation of our module guides. Your constructive feedback is appreciated.

Please e-mail to : Curriculum Development and Review

E-mail : modulefeedback@mancosa.co.za