CHAPTER 1
INTRODUCTION TO DATA STRUCTURE

FUNDAMENTALS OF DATA STRUCTURES (CSC248)

PREPARED BY:

MISS MAZNIE MANAF

TOPIC COVER

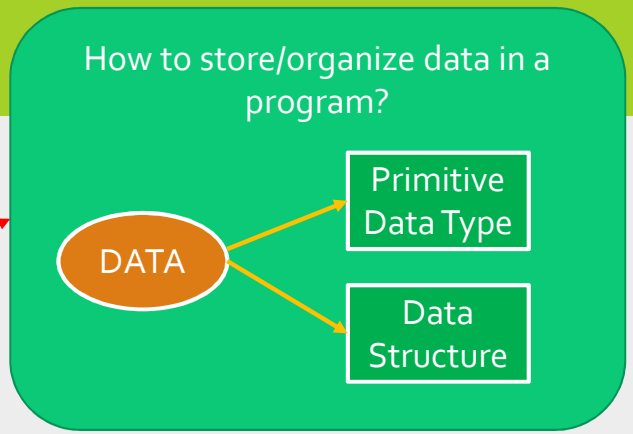
- Abstract Data Type concept
- Data Structure concept
- Application of structure data
- Implementation of Generic classes

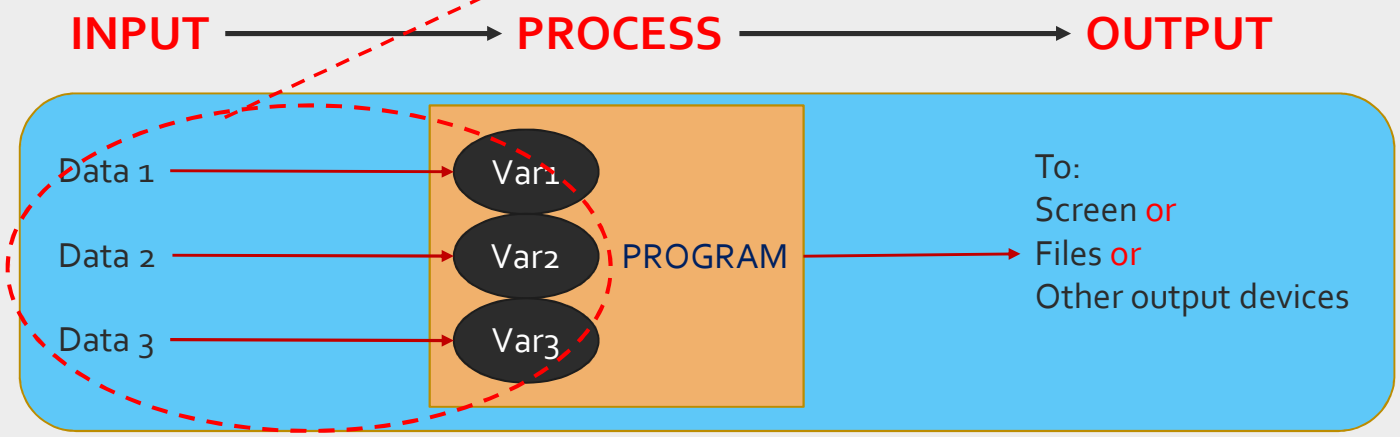
OBJECTIVE OF THIS CHAPTER

- At the end of this chapter, student should :
  - be able to define data structure
  - be able to define Abstract Data Type (ADT)
  - be able to associate between ADT and Data Structure

WHAT IS DATA?

- Raw material that processed by computer programs
- Many data types
  - Different operations allowed on the data
  - Different internal representation





INTRODUCTION TO DATA TYPE

- In programming, all variables are required to have data type
- In Java, there are have different Data Types :
  - Primitive /fundamental/base type
    - int, long, short, double, float, char, boolean and byte
    - a variable that stores a value of that type
  - Reference data type
    - An object or array
    - The address is stored in a variable not a value itself.
  - Abstract data type(ADT)
    - Specifies the properties and behaviors but implementation will be done later

REVISION

- Abstraction
  - the act of representing essential features without including the background details or explanations.
  - Use to manage complexity
- Interface
- Abstract Class – a class contains an abstract method

EXAMPLE: ARRAYS STATIC DATA STRUCTURE

- Consists of data items of the same type and data is stored contagiously in memory (adjacent to each other).
- Arrays are fixed-length entities with the capacity that is specified during the creation / declaration time and the length remains until end of program.
- Arrays in JAVA are objects and its considered as reference type.
- The elements of a JAVA array can be either primitive types or abstract data types (ADT).
- Example

PRIMITIVE DATA TYPE

```
// declare an int array
int  c[ ];

// create an array of  int with 10 elements
c  = new int[10];
```

ADT

```
public class Student{
        private String name;
        private double gpa;
        .....
}
```

```
public class Process{
    public static void main(String[] arg)
   { .....
     //declares and create the array object
     Student  Stu[ ];
    Stu = new Student[12];
     ......
}
```

WHAT IS DATA STRUCTURE ?

- A data structure is an arrangement of data in a computer's memory or even disk storage.
- Many algorithms require that we use a proper representation of data to achieve efficiency.
- This representation and the operations that are allowed for it are called data structures.
- Algorithms, on the other hand, are used to manipulate the data contained in these data structures as in searching and sorting (deletion, retrieval, insertion, etc).

DEFINITION of Data Structure:

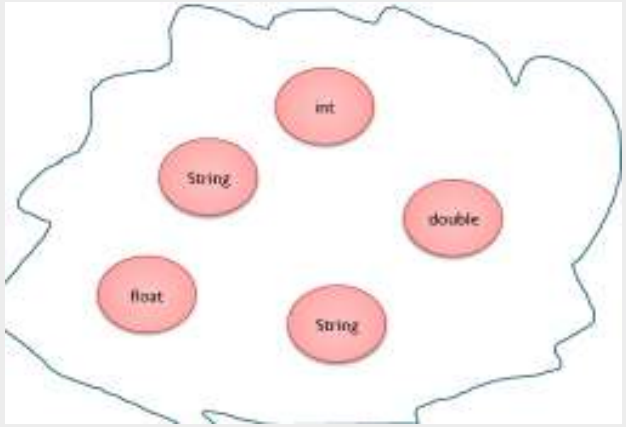
An organization of information, usually in memory, for better algorithm efficiency, such as queue, stack, linked list, heap, dictionary, and tree, or conceptual unity, such as the name and address of a person. It may include redundant information, such as length of the list or number of nodes in a subtree.

Paul E. Black (2004)

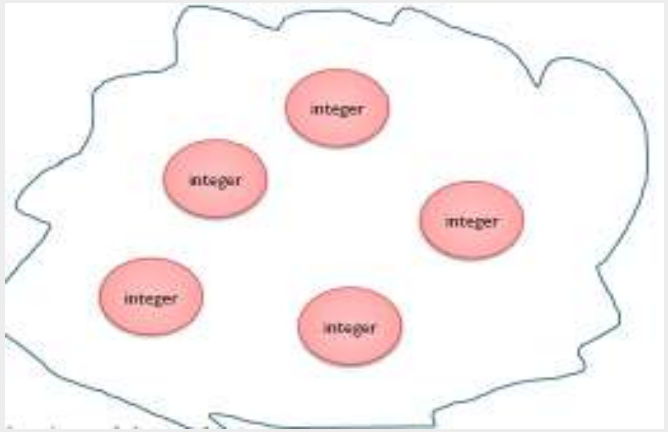
EXAMPLE OF DATA STRUCTURE

- Data structure can only be accessed with defined operations described in Abstract Data Type (ADT)

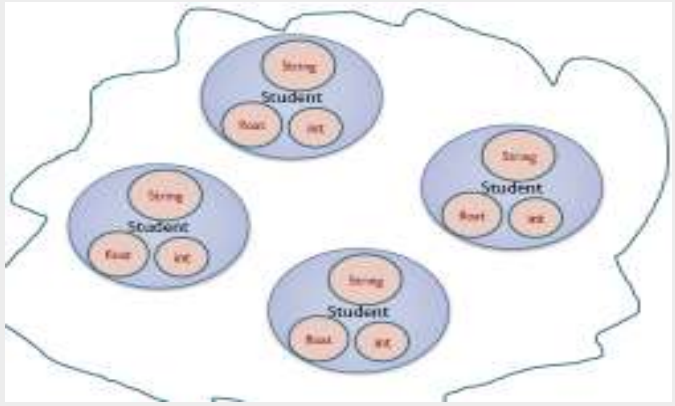
- An example of several common data structures are:
  - Arrays
  - Linked Lists
  - Queues
  - Stacks
  - Binary Trees
  - Hash Tables



a collection of data of different types eg. objects



a collection of data of the same primitive types eg. arrays



a collection of data of the same type of objects

- within the collection, the data has to be **organized** to form a Data Structure

CHARACTERISTIC OF DATA STRUCTURES

Data Structure	Advantages	Disadvantages
Array	Quick inserts Fast access if index known	Slow search and deletes Fixed size
Ordered Array	Faster search than unsorted array	Slow inserts and deletes Fixed size
Stack	Last-in, first-out access	Slow access to other items
Queue	First-in, first-out access	Slow access to other items
Linked List	Quick inserts and deletes	Slow search
Binary Tree	Quick search Quick inserts and deletes (If the tree remains balanced)	Deletion algorithm is complex
Hash Table	Fast access (if key known) Quick insert	Slow deletes Slow access (if key not know) Inefficient memory usage

SO…WHAT IS THE PURPOSE?

- Searching for a particular data item (or record).
- Sorting the data. There are many ways to sort data. Simple sorting, or using advanced sorting.
- Iterating through all the items in a data structure. (Visiting each item in turn so as to display it or perform some other action on these items)

ABSTRACT DATA TYPE CONCEPT

ABSTRACT DATA TYPE (ADT)

- A Data Structure is the implementation of an abstract data type.
- In OO languages, a developer implements an interface with class while user is interested in ADT, given as interfaces.
- ADT specifies the operations and behaviors to be done but not how to be done(defer the implementation) – concept of interface
- In another word is as following association :

General Term	Object Oriented Term
Abstract Data Type	interface
Data Structure	class

ABSTRACT DATA TYPE (ADT) CONTINUE

- ADT is also referred to as an object.
- ADT contains more than one fundamental data types.
- ADT also contains the methods as an interface to manipulate the data.
- Each ADT is an encapsulation of data and methods.
- In Java, ADT or object is defined using class.

DEFINITION of ADTs:

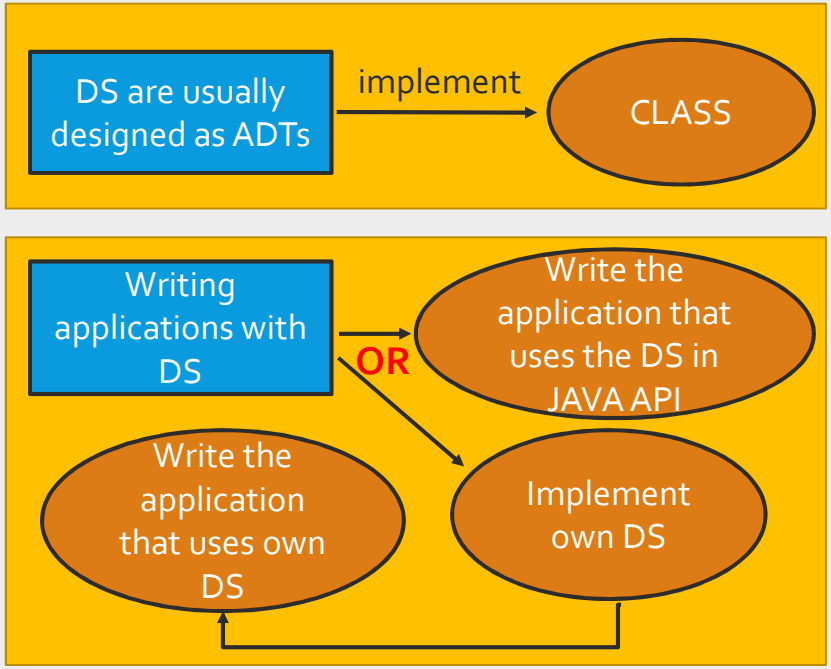
A data type that specifies the logical properties without the implementation details.

D. S Malik & P. S Nair, 2003

15

ABSTRACT DATA TYPE (ADT) CONTINUE

JAVA classes are implementations of ADTs



EXAMPLE:

```
class Computer {
    private int code;
    private String brand;
    private double price;

    public Computer () { }
    public Computer (int c, String b, double p) {...}
    public int getCode () { return code; }
    public String getBr () { return brand; }
    public double getPr() { return price; };
}
```

ABSTRACT DATA TYPE (ADT) CONTINUE

Using class w/o create ADT

- ADT can be used by any application program, it eliminates rewriting the code if there are changes in the implementation of the data / object.
- ADT provides the implementation-independent interfaces by having the set of methods in the classes.
- ADT hides the implementation of the methods in the encapsulation feature of the object.

Using class to create ADT

- Classes are used to represent objects in Object Oriented Programming (OOP) approach.
- Class contains :
  - Member data - a set of field (data)
  - Member Methods - manipulates fields
- OOP uses classes to encapsulate data (attributes) and methods (behaviors).
- Encapsulate enables objects to hide their implementation from other objects.

DATA STRUCTURE CONCEPT

DATA STRUCTURE CONCEPT

- Data Structure is a concept on how to structure (represent) and manipulate data (store, process, access and etc) in the program.
- Systematic way of organizing a collection of data
- Data structure is known as a collection of related data items.

TYPE OF DATA STRUCTURE IMPLEMENTATIONS

Besides time and space efficiency another important criterion for choosing a data structure is whether the number of data items it is able to store can adjust to our needs or is bounded.

Static Data Structure (*fixed-size*)	VS	Dynamic Data Structure
▪ Memory space allocation for the DS is done before the DS is built ▪ Static data structures are fixed at the time of creation. ▪ e.g., a structure used to store a postcode or credit card number (which have a fixed format).		▪ Memory space for the DS is done during execution and usually during insert operation. ▪ Dynamic data structures grow or shrink during run-time to fit current requirements. ▪ e.g., a structure used in modeling traffic flow.

TYPE OF DATA STRUCTURE IMPLEMENTATIONS CONTINUE

STATIC

ADVANTAGES

- Ease of specification
- Programming languages usually provide an easy way to create static data structures of almost arbitrary size
- No memory allocation overhead
- Since static data structures are fixed in size, there are no operations that can be used to extend static structures; such operations would need to allocate additional memory for the structure (which takes time)

DISADVANTAGES

- Must to make sure there is enough capacity
- Since the number of data items we can store in a static data structure is fixed, once it is created, we have to make sure that this number is large enough for all our needs
- more elements? (errors), fewer elements? (waste)
- However, when the program tries to store more data items in a static data structure than it allows, this will result in an error (e.g. ArrayIndexOutOfBoundsException)
- Thus, if fewer data items are stored, then parts of the static data structure remain empty, but the memory has been allocated and cannot be used to store other data.

21

TYPE OF DATA STRUCTURE IMPLEMENTATIONS CONTINUE

DYNAMIC

ADVANTAGES

- There is no requirement to know the exact number of data items
- Since dynamic data structures can shrink and grow to fit exactly the right number of data items, there is no need to know how many data items we will need to store
- Efficient use of memory space if we only extend a dynamic data structure in size whenever we need to add data items which could
- Otherwise not be stored in the structure and if we shrink a dynamic data structure whenever there is unused space in it,
- Then the structure will always have exactly the right size and no memory space is wasted

DISADVANTAGES

- Memory allocation/de-allocation overhead
- Whenever a dynamic data structure grows or shrinks, then memory space allocated to the data structure has to be added or removed (which requires time)

ADT vs DATA STRUCTURE

Abstract Data Type	Data Structure
▪ Is the logical picture of the data and the operations to manipulate the component elements of the data ▪ is in the logical level	▪ is the actual representation of the data during the implementation and the algorithms to manipulate the data elements ▪ is in the implementation level.

APPLICATION OF STRUCTURE DATA

DATA STRUCTURE APPLICATION

There are many data structure applications, some of the major ones are:

LIST

- List is an ordered sequence elements.
- contains a number of elements that is less than or equal to its capacity.
- Two types of list :
  - ***Sequential list*** (*array list*) – arrays that store data in a contiguous memory (elements next to each other)
  - ***Linked list*** – a linked structure that store data where one element refers to another, anywhere in memory

STACK

- used in compiler and Operating System
- insertion and deletions are made from the top of a stack

QUEUE

- represent a waiting lines and used in any queue eg. printing queue and request queue
- insertion are made at the end of a queue and deletions are made from the front of a queue.

25

DATA STRUCTURE APPLICATION CONTINUE

BINARY TREE

- Facilitate high speed searching and sorting of data.
- Store high volume of data with the links from one element to another anywhere in memory
- Eliminate duplicate data efficiently
- Represent file directories and compile expressions into machine language.

IMPLEMENTATION OF GENERIC CLASSES

DATA STRUCTURES IN THE JAVA API

- In Java 5.0 and above, data structures are implemented as generic classes
- A generic class is a general class which allows the same organization and operations on different ADTs
- Eg: ArrayList – which is an implementation of the sequential lists – is a generic class

IMPLEMENTATION OF GENERIC CLASSES

Generics provides a way for you to communicate the type of a collection to the compiler, so that it can be checked. Once the compiler knows the element type of the collection, the compiler can check that you have used the collection consistently and can insert the correct casts on values being taken out of the collection.

java.sun.com

IMPLEMENTATION OF GENERIC CLASSES

- Generic programming involves the design and implementation of data structures that work for multiple types.
- "To-be-specified-later" types
- In Java, it can be achieved with type variables.
- For a generic class, it has one or more type variables.

IMPLEMENTATION OF GENERIC CLASSES

- The ArrayList is an example of generic class.
- It implements "genericity" by using type variables.

```
public class ArrayList <E> {
                        ……
                        public <E> get(int index){
                                ……..
                        }
}
```


31

IMPLEMENTATION OF GENERIC CLASSES

- Type variables can be instantiated with class or interface types.
- Example:

```
GenericClassName <Type1, Type2..>
```

- Declare as:

```
ArrayList <Student>
ArrayList <Book>
```

Specify type variable of a generic class

32

IMPLEMENTATION OF GENERIC CLASSES

- Defining a generic class :

```
modifier class GenericClassName<type var1,…>
        {
                data members
                constructors
                methods
        }

    public class ArrayList<E>
        {
                public ArrayList(){…}
                public void add(E element){…}
                …..
        }
```

WHY GENERICS

- Stronger type checks at compile time.
- A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety.
- Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.
- Elimination of casts.
- Enabling programmers to implement generic algorithms.
- By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.

EXAMPLE

- The following code snippet without generics requires casting:

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

- When re-written to use generics, the code does not require casting:

```
List<String> list = new
ArrayList<String>();
list.add("hello");
String s = list.get(0); // no cast
```