Fundamental Data Structures

Madiba Hudson-Quansah

Contents

Chapter 1

Introduction to Data Structures

1.1 Data Structures

Definition 1.1.1: Data Type

Describes the set of values that a variable can hold and the types of operations that can be performed on it.

Definition 1.1.2: Data Structure

A way of storing and organizing data in computer memory / The logical mathematical model of a particular organization of data.

In choosing a data structure, we consider the following in the following order:

1. The relationships of the data in the real world
2. The type and amount of data to be stored
3. Cost (time complexity) of operations
4. Memory occupations
5. Ease of implementation

1.1.1 Types of Data Structures

Data structures can be broadly classified into two types:

- Primitive Data Structures
- Non-Primitive / Composite Data Structures

With data structures under the primitive category being:

- Variables
  - Integer
  - Float
  - Character
  - Boolean
  - Enumerated
  - Reference / Pointer

And data structures under the non-primitive category being:

- Arrays
- Structure / Record
- Union
- Class
- Abstract Data Type
  - Linear
    - List
    - Stack
    - Queue
  - Non-Linear / Associative
    - Tree
    - Graph
    - Hash Table

Data structures can also more simply be classified in two ways:

- Linear / Sequential Data Structures
  - Linked List
  - Array
  - Stack
  - Queue
  - Set
- Non-Linear / Associative Data Structures
  - Graph
  - Tree
  - Hash Table

In this way data structures are defined by an implementation of a particular abstract data type.

1.1.1.1 Primitive Data Structures

These are the basic data structures that are directly operated on by machine / CPU instructions. They are the atomic data type, i.e. they cannot be divided further

1.1.1.2 Non-Primitive Data Structures

These are data structures composed of or derived from primitive data structures.

1.1.1.3 Linear Data Structures

These data structures are characterized by the fact their elements can be accessed in a sequential / linear manner. This means every element, excluding the first and last elements which only have a successor and predecessor respectively, has both a predecessor and successor, allowing bi-directional traversal.

1.1.1.4 Non-Linear Data Structures

These data structures are characterized by the fact that their elements are accessed based on some relationship / association between elements, therefore non-linearly.

1.2 Non-Primitive Data Structures

1.2.1 Arrays

Definition 1.2.1: Array

An indexed collection of a fixed number of homogeneous data elements. Elements are stored in contiguous memory, i.e. sequentially and can be accessed by their index.

1.3 Generics

Definition 1.3.1: Type Parameter

A placeholder for a type that is used in the definition of a generic type.

Definition 1.3.2: Generic Type

A type that is defined with one or more type parameters, i.e., `<T>`. These type parameters are used to define the type of the data that the generic type can store.

Definition 1.3.3: Generics Class and Method

A class ,interface or method that is defined with one or more type parameters.

Generics enable classes and functions to be type safe yet reusable for multiple data types. Generic types also resolve type-casting problems, for example when a class is defined to store integers, it cannot store any other data type. However, with generics, the class can be defined to store any data type.

Definition 1.3.4: Type Safety

Prevents the program from compiling if the data type of the variable is not compatible with the data type of the object or if an unsupported operation is performed on the object. / Guarantees that the data type of the variable is compatible with the data type of the object.

1.3.1 Wildcards

Definition 1.3.5: Wildcard Character

The wildcard character, `<?>` , is used to denote that a generic type can be of any type, with the usual restrictions of parametrized types still applying.

A parametrized wildcard type on its own is called an unbounded wildcard type and essentially means the same as the upper bounded wildcard type `<? extends Object>`. When unbounded wildcard types in methods act similar to an equivalent generic method, for example:

```
public void print(ArrayList<?> list) {
  for (Object obj : list) System.out.println(obj);
}

public <T> void print(ArrayList<T> list) {
  for (T t: list) System.out.println(t);
}
```

These two methods perform exactly the same and work with any parametrized type.

```
public void printZip(ArrayList<?> first, ArrayList<?> second) {
    for (int i = 0; i < first.size(); i++) {
        System.out.println(first.get(i) + " " + second.get(i));
    }
}

public <T> void printZip(ArrayList<T> first, ArrayList<T> second) {
    for (int i = 0; i < first.size(); i++) {
        System.out.println(first.get(i) + " " + second.get(i));
    }
}
```

This listing shows the main difference in the use of normal parametrized types and wildcard types. The wildcard implementation can accept two difference types for each of the ArrayLists but the same type `T` is used for both lists in the generic method implementation.

1.3.1.1 Upper Bounded Wildcards

Definition 1.3.6: Upper Bounded Wildcard

A wildcard parametrized type that only accepts types of the bounded type or any subclass of the bounded type, denoted by:

`<? extends T>`

```
public double sum(ArrayList<? extends Number> numbers) {
    double sum = 0;
    for (Number number : numbers) sum += number.doubleValue();
    return sum;
}
```

In this listing the method `sum` only accepts an ArrayList of any subclass of the `Number` class, i.e. `Integer`, `Double`, `Float`, etc.

1.3.1.2 Lower Bounded Wildcards

Definition 1.3.7: Lower Bounded Wildcard

A wildcard parametrized type that only accepts types of the bounded type or any superclass of the bounded type, denoted by:

```
<? super T>
```

```java
public int sum(ArrayList<? super Integer> numbers) {
    int sum = 0;
    for (int i = 0; i < 10; i++) {
        sum += numbers.get(i);
    }
    return sum;
}
```

In this listing the method `sum` on accepts an ArrayList of type `Integer`, or superclasses, `Number` and `Object`.