Chapter 1 – Overview & Java Review

**DATA STRUCTURES:**

**DATA:** Data is Information

* The Source of the information
* any device attached to a computer system – keyboard, mouse, modem, disk drive
* the program itself can be the source of the data if, during processing, it generates data
* The Input or Output data set can be empty – but more often than not, it is not
* sometimes the I/O info appears to be empty but isn’t

🡪 random number algorithms – require a seed value

\*Studies show that programs spend 80% of execution time searching memory for the data they process

* The speed at which a program locates the data to be processed is usually a concern when designing programs that operate on large data sets
* It must also be considered when designing programs that operate on small data sets under stringent time constraints

\*Efficient use of main memory

* Main memory 🡪 limited resource
* Important to minimize the data storage requirements
* the amount of storage required in order for the data to be processed (& also the program instructions) must be kept to a minimum

**WHAT IS A DATA STRUCTURE:**

An organization of information, usually in memory, for better algorithm efficiency

\*Study of Data Structures

* The study of how to organize the information that a program processes in a way that improves the program’s performance
* organized in a way that *facilitates the operations* the programs algorithm performs on the data

Overhead: extra/additional storage over & above the size of the size of the program’s data set

\* A good data structure is one that organizes the data in a way that facilitates the operations performed on the data, while keeping its total storage requirement at, or close to, the size of the data set \*

\*Two Types of Data Structures:

* Built-In: Schemes for storing data that are part of a programming language
* An instance of a built-in type 🡪 variable
* Programmer-Defined: schemes for storing data that are conceived and implemented by the programmer of a particular program
* An instance of a class 🡪 object

**Data Structures & Performance:**

Programs that process large sets of data or have stringent speed constraints must consider:

* efficient use of storage – both main memory & external storage
* speed of execution

\*The choice of a data structure can have a large effect on the execution speed & memory of a program\*

Evaluating Performance:

It is too time consuming and costly to build applications in order to test them

* Instead – detailed calculations are made early in the design process to evaluate candidate designs
* While ideal, rapid operation & minimal storage are usually mutually exclusive
* Thus the selection of the best structure for an app is usually a compromise

Trade-Off Process:

* Select the least expensive data structure that satisfies the speed requirements & storage constraints of the application
* 3 factors to consider 🡪 cost, speed, & memory overhead
* Speed: varies widely – display viewed by humans - 1 sec is adequate
* Cost: labor costs associated w/ developing the code to implement the structure
* Cost of software is directly proportional to the # of lines of code

**FUNDAMENTAL CONCEPTS:**

Field: indivisible piece of data

* Phone listings consist of 3 fields 🡪 name, address, number

Node: a group of related fields

* A single listing

Key Field: a designated field in a node whose contents are used to identify / refer to the node

* A name or ID number

Homogenous Structure: a set of nodes in which all the nodes have identical fields (number & type)

* The data set in the phone listing application, an array?

Linear List: an Array?

**Access Modes:**

* Access is the process by which we locate a node in memory
* Only after a node is accessed or located, can it then be operated on
* 2 generic modes used to specify how the node is to be accessed
* Node Number Mode: the # of the node to be operated on is specified
* Key Field Mode: the contents of the designated key field are specified

**Operations:** Insert, Delete, Fetch, Update

**Implementing a Programmer-Defined Data Structure:**

Data Structures are implemented in object-oriented programming languages using the class construct

* Data Members: specifies the memory required for the data structure
* Class Methods: specifies the operations performed on the information stored in the data structure

\*The best way to implement a data structure is to implement it as a Generic Data Structure\*

* These are data structures implemented in such a way that they can be used for multiple applications, even if they don’t have the same node structure (# of fields, type of info)
* They reduce the cost of software development
* Once coded for a particular app – no need to code again for subsequent apps

**Procedural Abstractions & Abstract Data Types:**

“Viewing an entity as an abstraction” refers to the idea that we do not need to know the details of how the entity is implemented in order to use it

\*Abstractions are functional views of an Entity\*

Procedural Abstraction: means we do not need to know the implementation details of a Method / Procedure in order to use it

Data Abstraction: means we do not need to know the implementation details of a data structure in order to use it

* We don’t need to know the details of how the data is physically stored in memory, nor do we need to know the algorithms of the basic operations methods
* ***Abstract Data Type***: a data structure that can be used w/ this superficial level of understanding
* To most programmers, Arrays are abstract data types

🡪 we know how to use it to store data, but we don’t know how it does it

* ***Standard Abstract Data Type***: refers to a data structure whose operation method signatures conform to a consistent format
* Application Programmer 🡪 easily change an existing application by simply changing the line of the application that declares the DS Object
* \*Standardizing abstract data types reduces the cost of software\*

**Encapsulation:**

The idea that we write code in a way that establishes *compiler enforced protocols* for accessing the data that the program processes

* These protocols usually restrict access to the data
* Encapsulation limits the *scope* of the program
* Data Not Encapsulated 🡪 all code in an app has free access to the memory cells that store the program’s data
* tomsAge = 16;
* Data Encapsulated 🡪 direct access to the data could be limited to the code of the program’s Data Structure Model
* tomsAge = 16; - anywhere outside of the data structure model would cause syntax error
* update(tom, 16); 🡪 the assignment of toms age to 16 would have to be done inside the *update method* of the data structure class

The Class Construct allows us to encapsulate more than just data..

CONFUSING MATH STUFF

**JAVA REVIEW:**

There are several Java constructs, operations, & concepts that are used in the implementation of most Data Structures

**Arrays of Primitive Variables:**

* Primitive Variable: a single instance of integral or real types of information
* Declared using the Java data types – bool, byte, short, int, long, char, float, & double

Syntax:

* dataType[] arrayName; 🡪 allocates a reference variable (one that can store a memory location
* arrayName = new dataType[arraySize]; 🡪 allocates indicated number of memory cells & stores address of the first memory cell in the array

**Classes:**

A Programmer-Defined type that consists of data definitions & methods (subprograms) that operate on that data

* The name of the class in the name of the newly defined type
* The class definition – public class ClassName { }
* Does not allocate any memory, same as w/ built-in Primitive Types
* Simply provides a template for the memory that will be allocated when an instance of this newly defined type (class) is declared

\*Public: access modifier that allows a method in any application to declare a reference variable of this class type

\*Private: access modifier that instructs the translator to enforce the encapsulation of the class’s data members

* Constructor Method will initialize data members
* Executes automatically when an instance of the class (an object) is declared
* toString Method to facilitate the output of the data members’ values

**Objects:**

Syntax

* ClassName objectName; 🡪 allocates a reference variable w/ the specified objectName
* objectName = new ClassName(ArgumentList); 🡪 creates the specified ClassName object and places its address into the reference variable of objectName

\* The name of the reference variable is considered to be the name of the object \*

Object-Oriented programs contain two types of code 🡪 *client code & class definition code*

Client Code: the sequence of code that declares the object

* After an object is declared, the client code can invoke any publicly accessible method declared for the class
* In O-O languages – the client specifies the object that the method is to operate on / access
* The client does this by mentioning the object name in the method invocation statement

\*Exception – static methods which are declared using the class name

\*Accessing Information stored in Objects is slower than accessing information stored in Primitive Variables

* Primitive: requires only 1 memory access
* Integer stored in a primitive variable at location 2000
* Must simply just access location 2000
* Object: requires 2 memory accesses
* The Reference Variable that stores the address of the object must be accessed to locate the object
* Then the integer can be accessed

**Standard Method Name Prefixes:**

Java adopted a *prefix convention* for naming methods to further promote readability

* These prefixes give insights into the source or destination of the data the method processes

Input/Show: indicates info is flowing between an I/O device & the method

* Input:
* Show:

Get/Set: indicates info is flowing between the client code & the method

* Set: typically contains a parameter list
* Get: typically a nonvoid method that contains a return statement

\*toString() method – similar to this method in that in returns information (the annotated values of a class’ data members) to the client code

\*Without Get&Set methods, the client code would not be able to access an object’s data members - since they’re usually private\*

**Shallow and Deep Copies:**

Copying information from one primitive variable to another is done with the assignment operator

* A = 2; & B = 3; 🡪 A = B; 🡪 A & B now both = 3

With Objects – there are 2 types of copy operations we can perform

\*When an object is allocated, the address of the object is stored in a reference variable\*

* Shallow Copies: only affect the contents of the Reference Variables
* ReferenceVariable1 = ReferenceVariable2; Marie = Nate;
* The above statements are the same as when we copy one primitive value into another
* The location value 100 stored in Nate is copied into the variable Marie
* Both reference variables refer to the same object
* Since the address of the object at location 200 is no longer stored in a reference variable – would be returned to the available storage pool by the Java memory manager
* Deep Copies: only affect the contents of objects – reference variable contents unchanged
* Copies the contents of the data members from one object, into the data members of the other object
* Deep copy 🡪 toString() method to output 🡪 output is coming from 2 different objects
* Since the reference variables are unaffected by the deep copy – both objects are still referred to / references in memory

**Arrays of Objects:**

Declared using a 3-step process

1. Declare a reference variable in which to store the location of the first element in the array
2. Declare an array of n reference variables – declare the 10-element array

* In the case of Primitives: this gives you 10 *storage locations*
* In the case of Objects: this gives you an array of 10 *reference variables*

1. Declare the n objects & set their locations into the array of n reference variables – 10

**Inheritance** – Classes that Extend Classes

WeightLossClient Extends Person meaning – WeightLossClient objects contain all the data members & methods of a Person along w/ additional ones.

Public class WeightLossClient extends Person {

Public WeightLossClient(int a, double w, double g) {

super(a, w) //invokes Person’s constructor to initialize a & w

goalWeight = g; //the newly added attribute

}

}

**Parent-Child-References:**

* A parent class reference variable can contain the address of a child object BUT
* A child class reference variable cannot contain the address of a parent object

\*Child reference variables cannot refer to Parent objects\*

**Generic Types:**

Allows the author of a method or class to generalize the type of information with which the method or class will deal

* The choice of the type of information is left to the invoker of the method, or the declarer of the object
* Generic Method 🡪 can be invoked to process Integer Objects, and also be invoked somewhere else in the same program to process Double Objects
* Generic Class 🡪 data member types & the parameters of class methods can be different for each object instance that the program declares

Generic Methods:

Generic Classes: