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:
* An instance of a built-in type 🡪 variable
* Programmer-Defined:
* An instance of a class 🡪 object

**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

**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:**

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