Week 1 Course Overview & Java Basics

4009 DATA STRUCTURES & ALGORITHM (DSA)

Learning Outcomes

At the end of week 1 students should be able to:

- Understand how data structures help in organizing and storing data.
- Define and discuss concepts of data structures & algorithm
- Differentiate between a data structure and an algorithm
- Understand and discuss the three categories where DSA can be applied in the real world
- Understand and discuss basic database concepts relating to DSA
- Identify basics of a Java program syntax and rules
- Implement their first Java program (hello world program)

Content Overview

- Course Overview
- Definition of a Data Structure
- Definition of an Algorithm
- What sort of problems can be solved with DSA?
- Real World Data Storage
- Programmer Tools
- Real World Modelling
- Data Structure Trade-offs
- Overall costs of structures we'll study
- Databases
  - Database Records
  - Database Fields
  - Database keys
- Java Basics
  - Getting Started with Java
  - Reserve words & Comments
  - Base Types
- Class exercise
- Lab activity

Course Overview

After mastering fundamentals of programming. You may start to question:

- What are data structures and algorithms?
- What good will it do me to know about DSA?
- Why can’t I just use arrays and for loops to handle my data?
- When does it make sense to apply what I learn here?

Provide background in data structures (arrangements of data in memory) and algorithms (methods for manipulating this data)

Why is this important? The answer is there is far more to programming than finding a solution that will work:

- Execution time
- Memory consumption

So by the end of the course, you should be more equipped to not just develop an accurate solution to a problem, but the *best* solution possible.

Definition of a Data Structure

A ***data structure*** is an arrangement of data in a computer's memory (or disk).

Data structure include arrays, linked lists, stacks, binary trees and hash tables, etc.

Questions to ponder:

- What are some examples of data structures you already know about from your Java course?

- How can the arrangement of data in memory affect performance?

Definition of an Algorithm

An ***algorithm*** provides a set of instructions for manipulating data in structures.

Algorithm involve searching particular data or sorting the data.

Many algorithms apply directly to specific data structures, you need to know how to:

- insert a new data item
- search for a specific item
- delete a specified item
- iterate through all the items in a data structure
- sort through items in a data structure

Questions to ponder:

- What's an example of an algorithm?
- How can the design of an algorithm affect performance? How can it affect memory?

Data Structure or Algorithm?

Linked List – Data structure

Sort – Algorithm

Search – Algorithm

Stack – Data structure

Vector – Data structure

What sort of problems can be solved with DSA?

Rough approximation, we divide the situation into three categories

- **Real world data storage**
- **Programmer's tools**
- **Real world Modeling**

Real World Data Storage

Real-world data: data that describes physical entities external to the computer.

Can we think of some examples?

- personal record that describes an actual human being

- an inventory record that describes an existing car part

What's an example of non-computer data storage?

- Addresses and phone #s:
- Book names, titles, ISBNs:

Real World Data Storage

Example - personal record that describes an actual human being

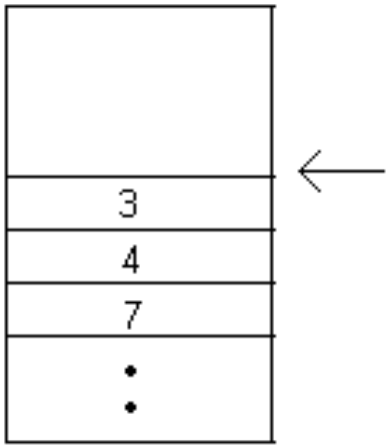
Say we wanted to convert one of these systems or the example above to a computer program, what must we consider?

- Memory consumption:
  - How would you save the data in your computer's memory?
- Scalability:
  - Would your method work for a hundred recorded? A thousand or a million?
  - Would your method permit quick insertion of a record or deletion of an old record?
  - How big will each records be and how should be stored?
- Algorithms (which ones do we care about?):
  - Would it allow for fast searching of a specified record?
  - Suppose you wanted to arrange the cards in alphabetical order. How would you sort them?

Important: Data Structures can be HUGE!!! What are some example scenarios where we need large data structures?

Now the issue of scalability comes into play. Suppose I design a reusable component for (i.e.) sorting an array of records by last name. I'm going to distribute this on the internet.

Programmer Tools



Not all data storage structures are used to store real-world data.

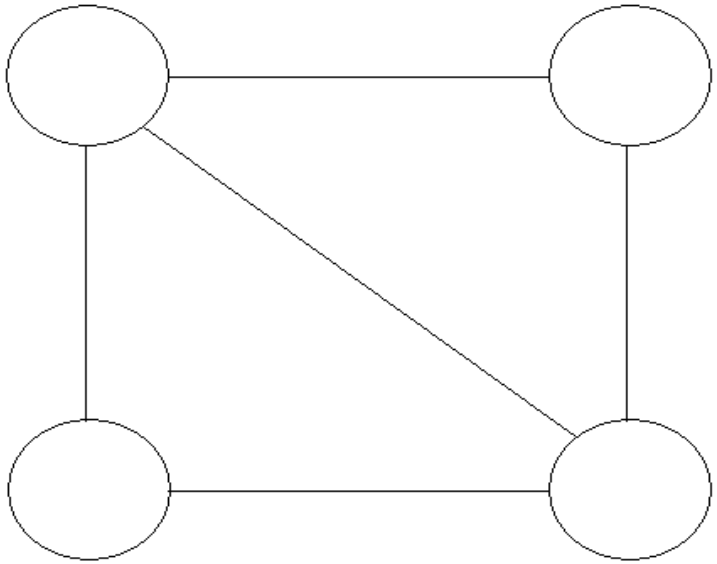
Real world data is accessed more or less directly by a program's user

- Some data structures are not meant to be accessed by the user but by the program
- A programmer uses such structures as tools to facilitate some other operation. Use case of stacks, queues and priority queues.
- Example - A 'stack' of data, where I can only insert or remove elements from the top:

Real World Modeling

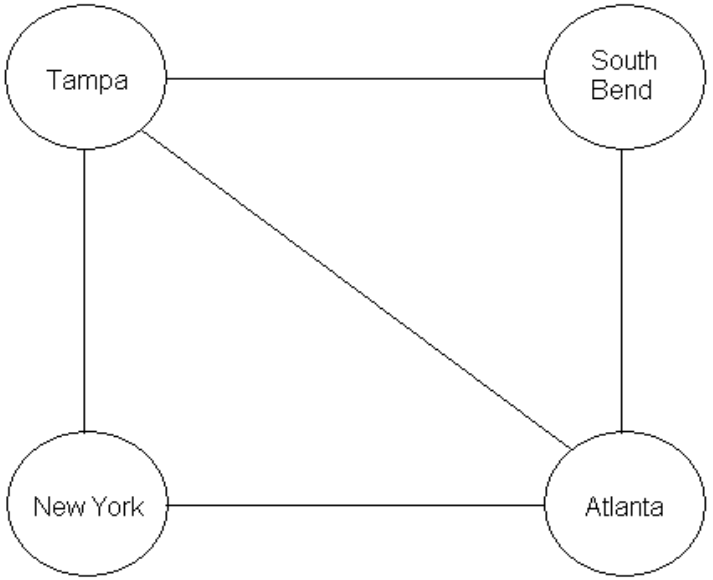
Effectively, ‘simulate’ a real-world situation.

For example, what could the following represent:



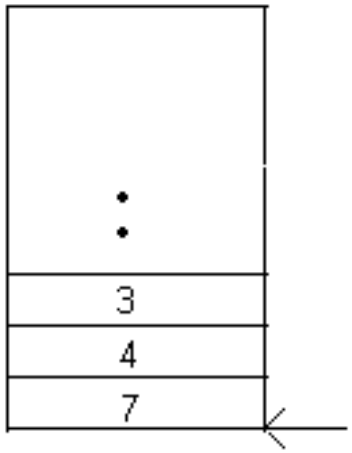
How about airline routes?

This type of structure is called an ‘undirected graph’



Real World Modelling

How about a ‘queue’ of data, where the first element in is the first element out (termed ‘FIFO’):



Example applications:

- Grocery store lines
- Traffic (Queues are actually used when determining timing of traffic lights!! How? Let’s think about it)

Data Structure Trade-offs

A structure we have dealt with before: arrays

Requirement that is enforced:

- Arrays store data sequentially in memory.

Data structures	Advantages	Disadvantages
Array	Quick insertion, very fast access to index known	Slow search, slow deletion, fixed size
Ordered Array	Quicker Search than unsorted array	Slow insertion and deletion, fixed size
Stack	Provide Last-in First-out access	Slow access to other items
Queue	Provide First-in First out access	Slow access to other items
Linked List	Quick Insertion, Quick deletion	Slow search
Binary Tree	Quick search, insertion, deletion (if tree balanced)	Deletion algorithm is complex
Hash Table	Very fast access if key known, fast insertion	Slow deletion, access slow if key not known, inefficient memory usage
Heap	Fast insertion, deletion, access to large item	Slow access to other items
Graphs	Model real world situations	Some algorithms are slow and complex

Overall Costs for Structures We'll Study

Structure	Access	Search	Insert	Delete	Impl.	Memory
Array	Low	High	Med	High	Low	Low
Ord. Array	Low	Med	High	High	Med	Low
Linked List	High	High	Low	Low	Med	Med
Stack	Med	High	Med	Med	Med	Med
Queue	Med	High	Med	Med	Med	Med
Bin. Tree	Med	Low	Low	Low	High	High
R-B Tree	Med	Low	Low	Low	Very High	High
234 Tree	Med	Low	Low	Low	Very High	High
Hash Table	Med	Med	Low	High	Low	High
Heap	Med	Med	Low	Low	High	High
Graph	High	High	Med	Med	Med	Med

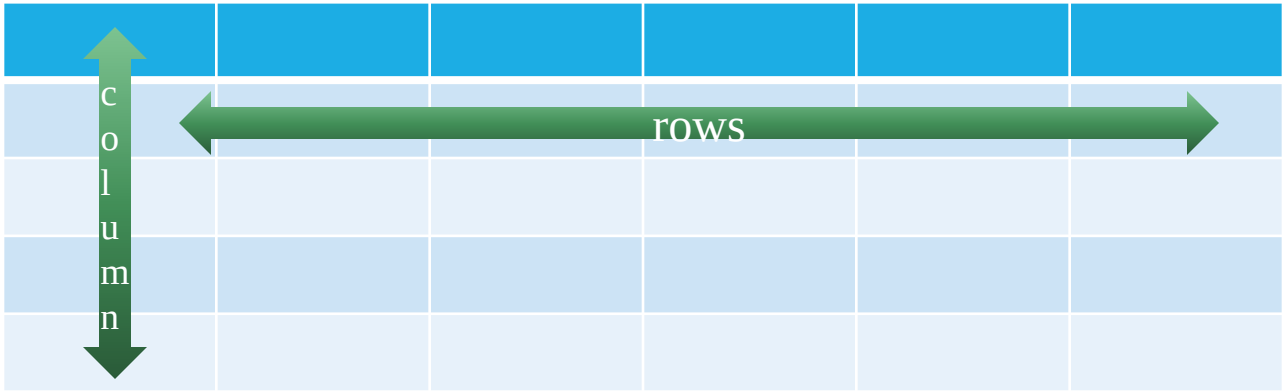
Point that you should be getting: No 'universal' data structure!!!

Algorithms We'll Study

- Insertion/Searching/Deletion
- Iteration. Java contains data types called iterators which accomplish this.
- Sorting. Believe it or not, there LOTS of ways to do this! So much, you get two chapters on it.
- Recursion.
  - Involves a method calling itself.
    - Java uses the term method while other languages use term of function, procedure or subroutine.
  - Import step to note: What's the key attribute of a recursive function in Java?
  - This technique will be used to develop some of the afore mentioned algorithms.

Databases

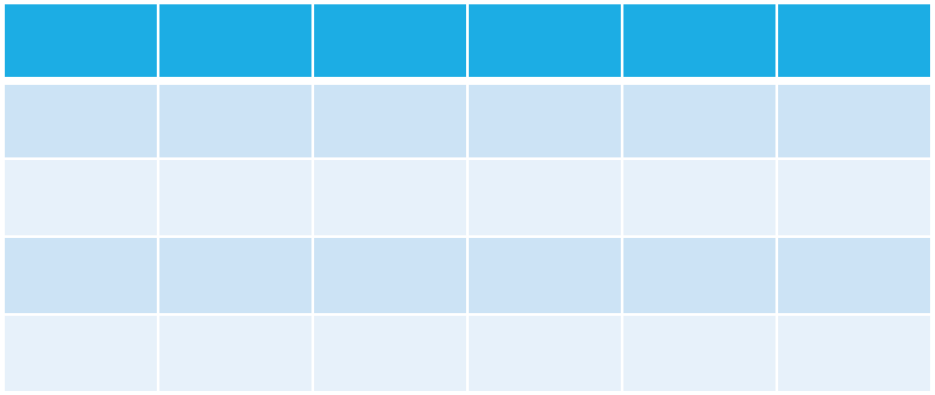
A database refers to all data that will be dealt with in a particular situation. We can think of a database as a table of rows and columns:



What is the ‘database’ in the case of a collection of index cards and names/phone #s?

Database Records

A ***record*** is the unit into which a database is divided. How is a record represented in a table:



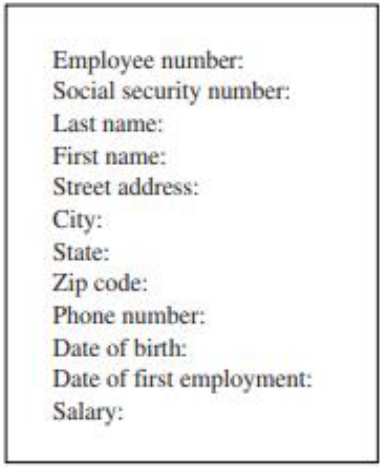


FIGURE 1.1 A record with multiple fields.

What would be a 'record' in a stack of index cards?

What about a banking system? How about a cookbook?

Note: In Java, records are usually represented by objects of an appropriate class.

Database Fields

- A field is the unit into which a record is divided. A field holds a particular kind of data.
- Figure 1.1 shows such a record where each line represents a distinct field

Employee Number					
1					
2					
3					
4					

- What would some example fields be for a banking system?

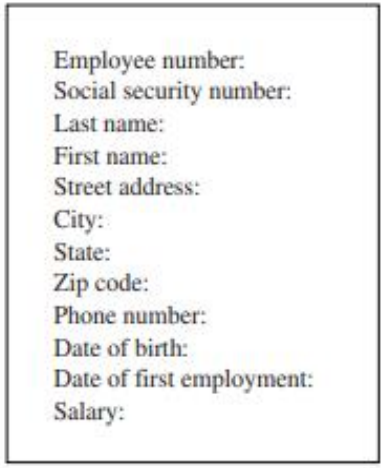


FIGURE 1.1 A record with multiple fields.

Note: In Java, individual variables within an object represent data fields. Also fields within a class object are called fields in Java

Database Keys

Given a database (a collection of records), a common operation is obviously searching. In particular we often want to find a single particular record. But what exactly does this mean? Each record contains multiple fields, i.e.:

Last	First	Acct No.	City	State	Zip
Azuma	Goro	123456789	St. Pete	FL	33712
Smith	John	867530912	Clearwater	FL	33720
Gunnz	JT	102938475	South Bend	IN	46545
Zozzfuzzel	Ziggy	000000000	St. Pete	FL	61589

So how do we designate a record? We need something unique to each record, that's the key. What is the key above?

Database Keys

So if I wanted to search for a particular record, I would want to do it by key (i.e. Acct No).

Note that searching by anything else could yield multiple records.

Last	First	Acct No.	City	State	Zip
Azuma	Goro	123456789	St. Pete	FL	33712
Smith	John	867530912	Clearwater	FL	33720
Gunnz	JT	102938475	South Bend	IN	46545
Zozzfuzzel	Ziggy	000000000	St. Pete	FL	61589

Java Basics

PART 1 OF 2

Getting Started with Java

Building data structures and algorithms requires that we communicate detailed instructions to a computer. An excellent way to perform such communication is using a high-level computer language, such as Java.

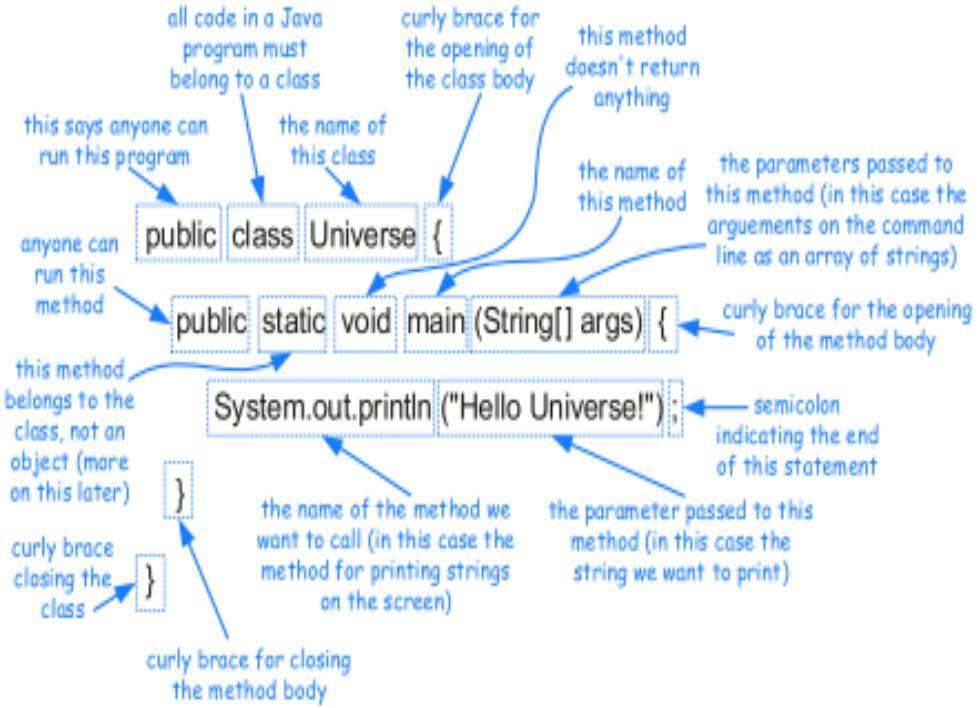


Figure 1.1: A "Hello Universe!" program.

In Java, executable statements are placed in functions, known as methods, that belong to class definitions.

The Universe class, in our first example, is extremely simple; its only method is a static one named main, which is the first method to be executed when running a Java program.

Any set of statements between the braces "{" and "}" define a program block.

Notice that the entire Universe class definition is delimited by such braces, as is the body of the main method.

The name of a class, method, or variable in Java is called an identifier, which can be any string of characters as long as it begins with a letter and consists of letters, numbers, and underscore characters.

Reserve words & Comments

Reserved Words				
abstract	default	goto	package	synchronized
assert	do	if	private	this
boolean	double	implements	protected	throw
break	else	import	public	throws
byte	enum	instanceof	return	transient
case	extends	int	short	true
catch	false	interface	static	try
char	final	long	strictfp	void
class	finally	native	super	volatile
const	float	new	switch	while
continue	for	null		

Table 1.1: A listing of the reserved words in Java. These names cannot be used as class, method, or variable names.

Comments

Java allows a programmer to embed comments, which are annotations provided for human readers that are not processed by the Java compiler.

Java allows two kinds of comments: inline comments and block comments.

Java uses a "//" to begin an inline comment, ignoring everything subsequently on that line.

For example: // This is an inline comment.

While inline comments are limited to one line, Java allows multiline comments in the form of block comments. J

ava uses a "/*" to begin a block comment and a "*/" to close it.

For example: /* * This is a block comment. */

Block comments that begin with "/**" (note the second asterisk) have a special purpose, allowing a program, called Javadoc, to read these comments and automatically generate software documentation.

Base types

The types of objects are determined by the class the come from.

Java provides the following base types (also called ***primitive types***)

Base Types	Value	Example
boolean	True / False	“True”, “False”
char	16-bit Unicode character	“c”, “a”, “B”, “*”
byte	8-bit signed two’s complement integer	21, 10, 12
short	16-bit signed two’s complement integer	100, 125, 234
int	32-bit signed two’s complement integer	1045, 1222, 1201
long	64-bit signed two’s complement integer	12456L, 10849L, 10023L
float	32-bit floating-point number	3.456789F, 1.2345678F
Double	64-bit floating-point number (I	3.456789, 1.2345678

Simple test program in Java.

```java
/**
    * 4009 DSA - 2025
    * A simple Java test program
    * @author: Mr Jerome Kris Semos
    * @Version: 1 - 2025
*/

public class Test {

    public static void main(String args[]) {
    //define variables and assignment value
    int age = 0;

    //calculate age
    age = age + 7;

        //print the message to console
        System.out.println("Puppy age is : " + age);
    }
}
```

Identity the different comments used, use of base types and avoid use of reserve words.

Topic next week

Week 2 Java Basics (part 2)

DSA Week 1 activities

Refer to print out materials.

- This week, you are required to complete two questionnaires and one lab.
  - In your DSA textbook 1, answer questions 1 to 6.
  - Refer to the print out, answer all week 1 questions.
  - Refer to the print out, complete lab one activity using the lab computers.

Note: You can complete the activities in any order, however, make afford to finish and understand everything which prepares you for well for test 1, test 2, Major Assignment, Mid Semester Exam & Final Exam.