UNIVERSITI TEKNOLOGI MARA

CSC508 Data Structures

Topic 1 : Introduction to Data Structures

Compiled & edited by: Zahid Zainal

Topic Structure

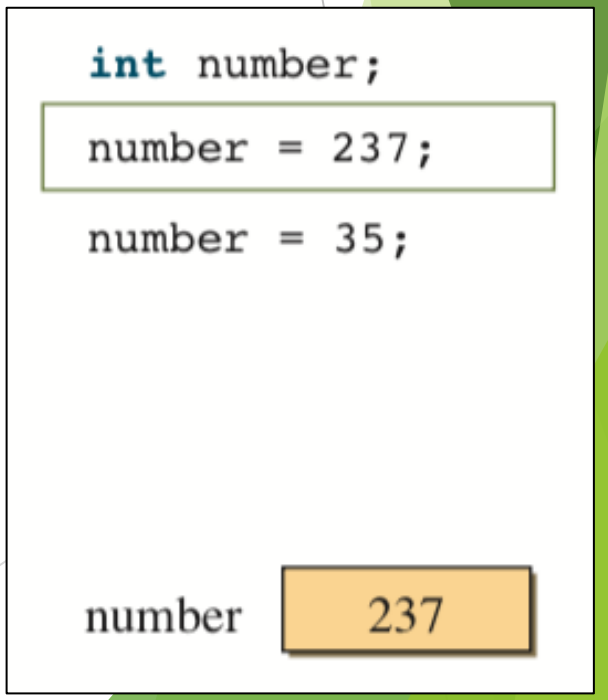
- Data types
- Definition of Data Structures
- Java Collection Framework
- Review
  - Java Abstract Classes
  - Java Interfaces

Learning Outcomes

- At the end of this lesson, students should be able to:
  - Define primitive, reference and abstract data types.
  - Describe data structures and its relation to algorithm and ADT.
  - Understand the concept Java Collection Framework.
  - Define Java abstract classes and interfaces.

Primitive Data Types

- Primitives data types are the fundamental data types built-in in any programming language.
- In Java, there are threes categories of primitive data types:
  - Integral (char, byte, short, int, long)
  - Floating point (float, double)
  - Boolean
- Data type need to be specified upon creation of a variable.

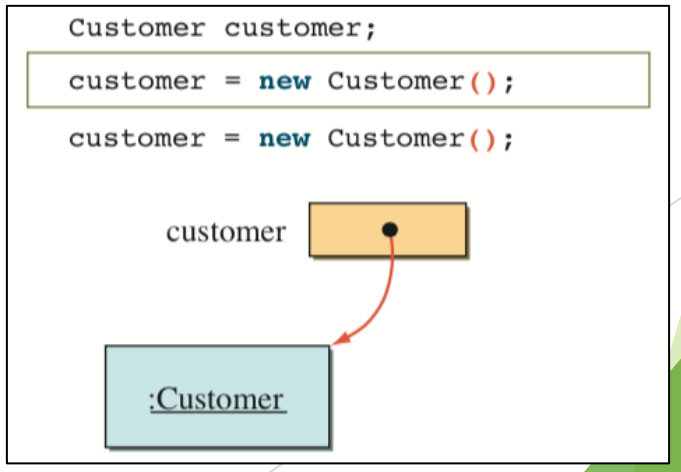


Reference Data Type

- A reference data type stores the memory location, that is, the address of the memory space where the actual data is stored.
  - Points to object that is dynamically created.
- An object is called a reference data type because the content is an address that refers to memory location where the object is being stored.
- Sample object declaration:

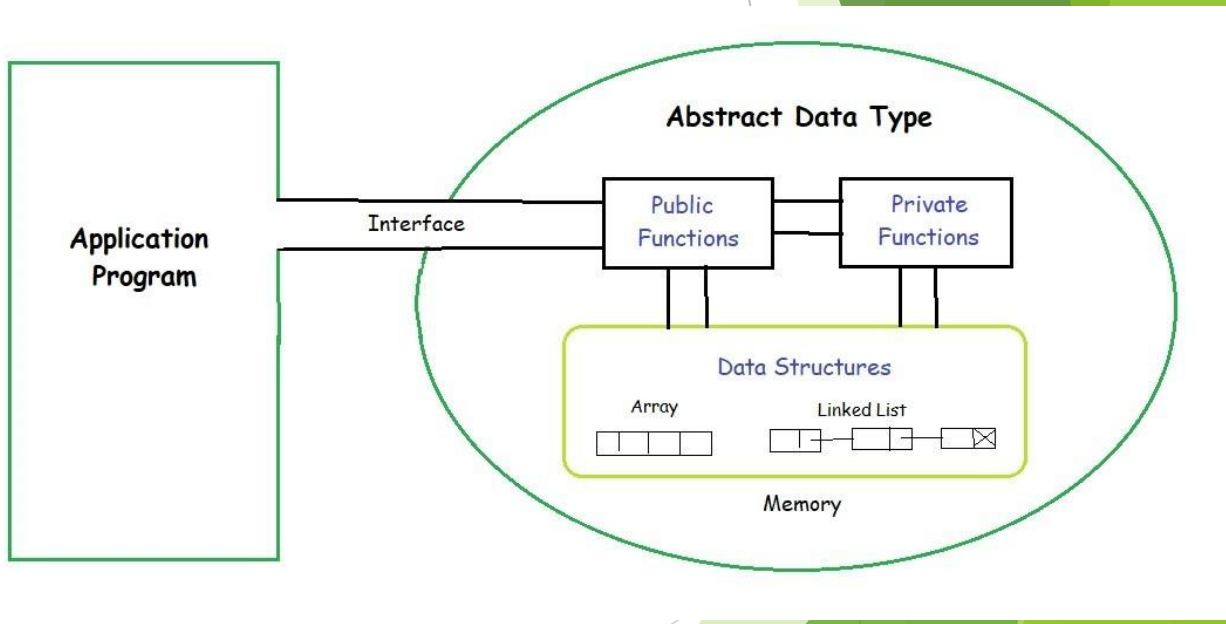
Similar to **pointer** concept in C++

```
Customer customer;
customer = new Customer();
customer = new Customer();
```



Abstract Data Type

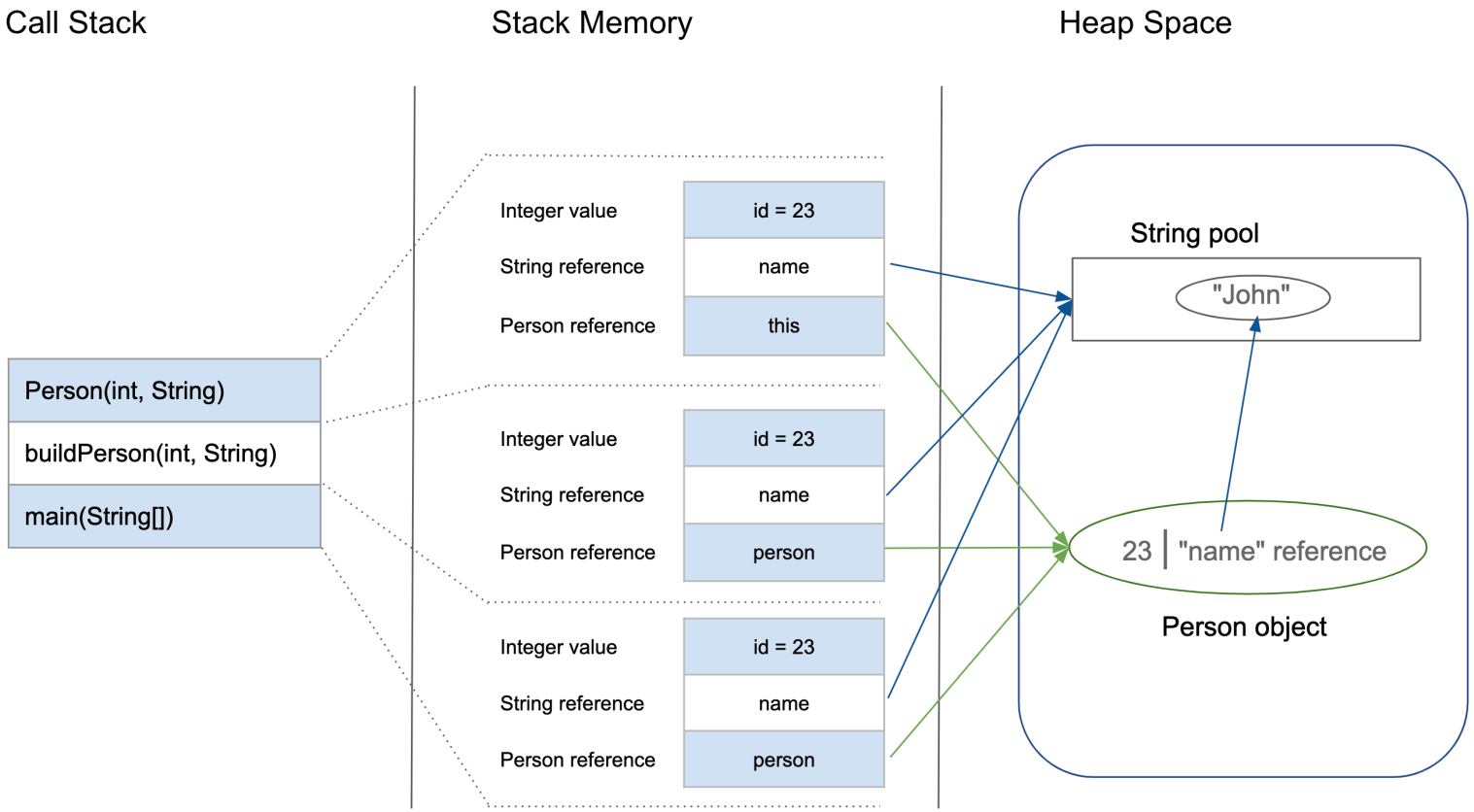
- A type (or class) for objects whose behavior is defined by a set of value and a set of operations.
  - specifies the logical properties without the implementation details.
- An ADT is presented as a class type and treated as a single entity that encapsulates its data together with its method
  - ADT kept the implementation details of the operations and the data from the users of the ADT



Abstract Data Type (cont.)

- ADTs are generic model to structure data
- An ADT consists of an abstract specification of:
  - The data (static)
  - Operations (methods) for accessing and manipulating the data (dynamic)
- It practices the software engineering principles of abstraction, encapsulation, independent modification, extension and reuse.

JAVA Memory Management



https://www.baeldung.com/java-stack-heap

Data Structure

- A representation of data and the operations allowed on that data
- In programming, the term data structure refers to a systematic way of organizing and accessing data for better algorithm efficiency
- Examples : arrays, records, files, etc.
- There are two types of data structures:
  - Linear (array, list, stack, queue)
  - Non-linear (tree, graph)

Algorithm

- Finite steps to solve a problem
- The algorithm chosen is based on the analysis of the problem
- A problem can be solved using different algorithm and data structures

Programming = Data Structure + Algorithms

- N. Wirth-

ADT vs Data Structure

- ADT is implementation independent. For example, it only describes what a data type List consists (data) and what are the operations it can perform, but it has no information about how the List is actually implemented.
- Data structure is implementation dependent, as in the same example, it is about how the List is implemented ie., using array or linked list.

Ultimately, data structure is how we implement the data in an abstract data type.

ADT and Data Structures

- A data structure that implements an ADT consist of:
  - Vvariables for storing the data described by the abstract model underlying the ADT
  - Algorithms for the methods of the ADT
- ADTs have a clean interfaces; implementation details are hidden in the classes that implements them
- E.g. The List interface is the ADT, the ArrayList and LinkedList classes are the data structures

Why Data Structure?

- Data structures: conceptual containers to organize data for efficient storage and manipulation
- Proper selection of data structures/ algorithm design is crucial to the performance of a problem solution
- Inappropriate data structures result in poor performance
- Performance is based on time and space complexity
  - Big O notation, O(n)

Database vs. Data Structures

DATABASE	DATA STRUCTURE
An organized collection of data generally stored and accessed electronically from a computer system	A data organization, management and storage format that enable efficient access and modification
Relational and Non Relational Databases are some types of databases	Linear and non linear data structures are some types of data structures
Stored in permanent memory	Stored in temporary memory
Helps to access and manage data easily	Helps to increase the efficiency related to time and space

Visit www.PEDIAA.com

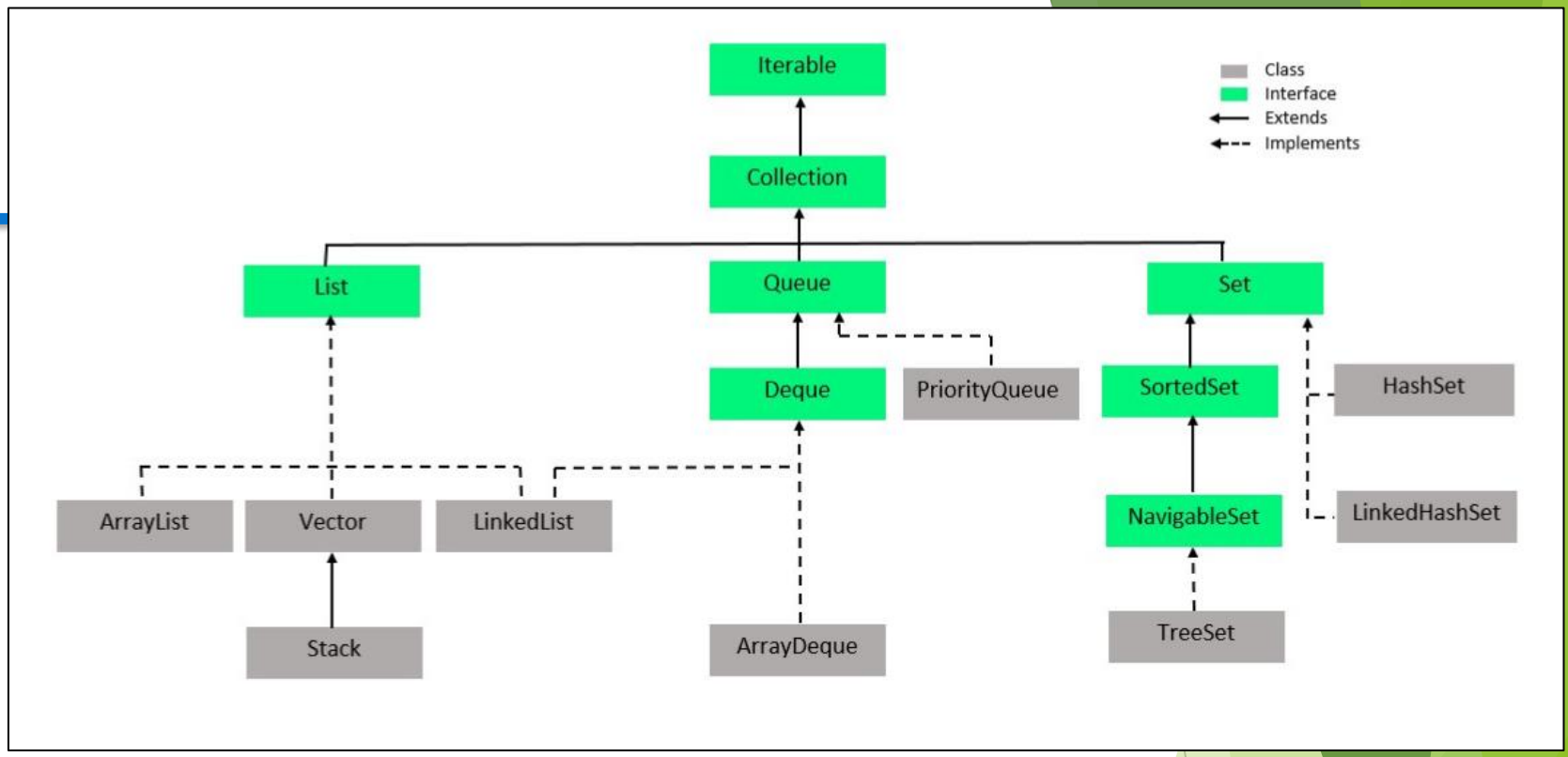
https://pediaa.com/what-is-the-difference-between-database-and-data-structure/

Java Collection Framework

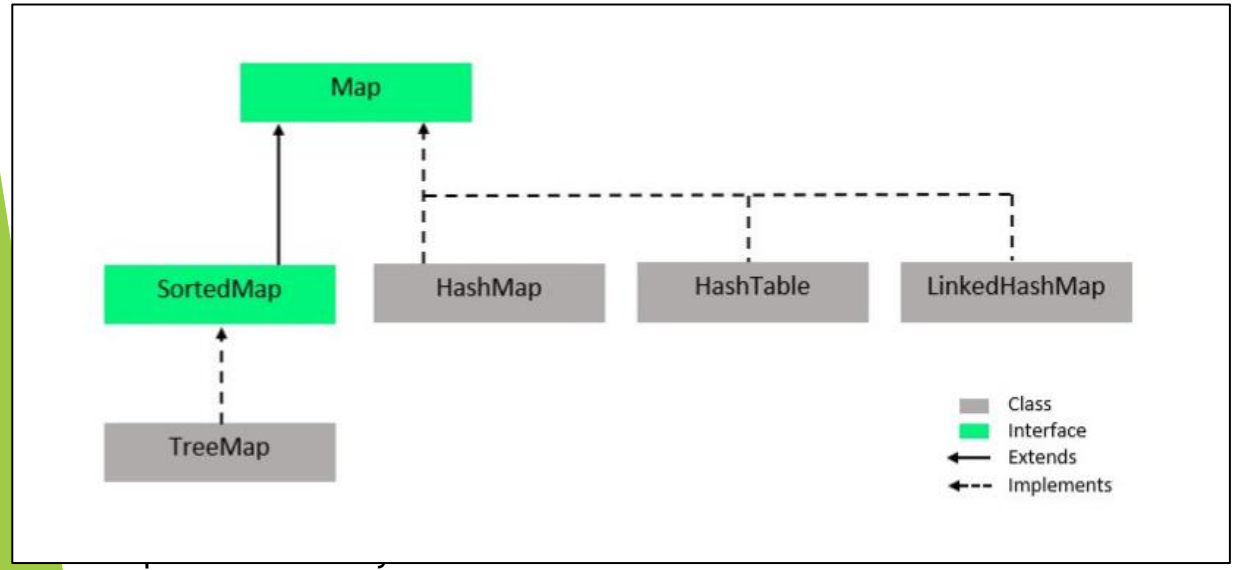
- The JAVA collection framework is part of the package java.util.
- It is basically a hierarchy, with interfaces and abstract classes at every level except the lowest. At the lowest level are classes that implement those interfaces and extend the abstract classes.
- The Collections APIs is the one library for data structures and algorithms that is guaranteed to be available.

What is Collection Framework?

- A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:
  - **Interfaces**: These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation.
  - **Implementations**: These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
  - **Algorithms**: These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces.



https://www.geeksforgeeks.org/how-to-learn-java-collections-a-complete-guide/



Why Java Collection Framework?

- Reduce programming effort
- Increases program speed and quality
- Allows interoperability among unrelated APIs
- Reduces effort to learn and to use new APIs
- Reduces effort to design new APIs
- Fosters software reuse

Collection Implementation

- Core implementation interfaces are the foundation of the Java Collections Framework.
- Classes that implements the collection interfaces typically have names of the form

  <Implementation-style><Interface>

- The general purpose implementations are summarized in the table below:

		Implementations				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Interfaces	Set	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap

Collection Implementation (cont.)

- Note that all the core collection interfaces are *generic*. For example, this is the declaration of the Collection interface.

```
public interface Collection<E>…
```

- The <E> syntax tells you that the interface is *generic*. When you declare a Collection instance you can *and should* specify the type of object contained in the collection. Specifying the type allows the compiler to verify (at compile-time) that the type object you put into the collection is correct, thus reducing errors at runtime.

Review: Java Abstract Classes

- An **abstract class** is a class that contains at least one abstract method or a class which is declared abstract.
- An **abstract class** cannot be instantiated, i.e., we cannot create objects from an abstract class.
- An **abstract method** is an undefined method. It contains no implementation -- no body, no method statements – it contains the

  abstract keyword, the method name and signature only, followed by a ";".

```
public abstract class Mammal {
    ....
    public abstract void eat();
    /* all mammals eat, but what
    each eat depends on the
    actual Mammal; thus each
    Mammal subclass must
    implement this abstract eat
    method */
}
```

Review: Java Interfaces

- The `interface` keyword - it is a "pure" abstract class, i.e., **it provides only a form/ concept for a class**

  - has no constructors, so, cannot be instantiated,

  - only abstract methods (method names, argument lists, and return types), and

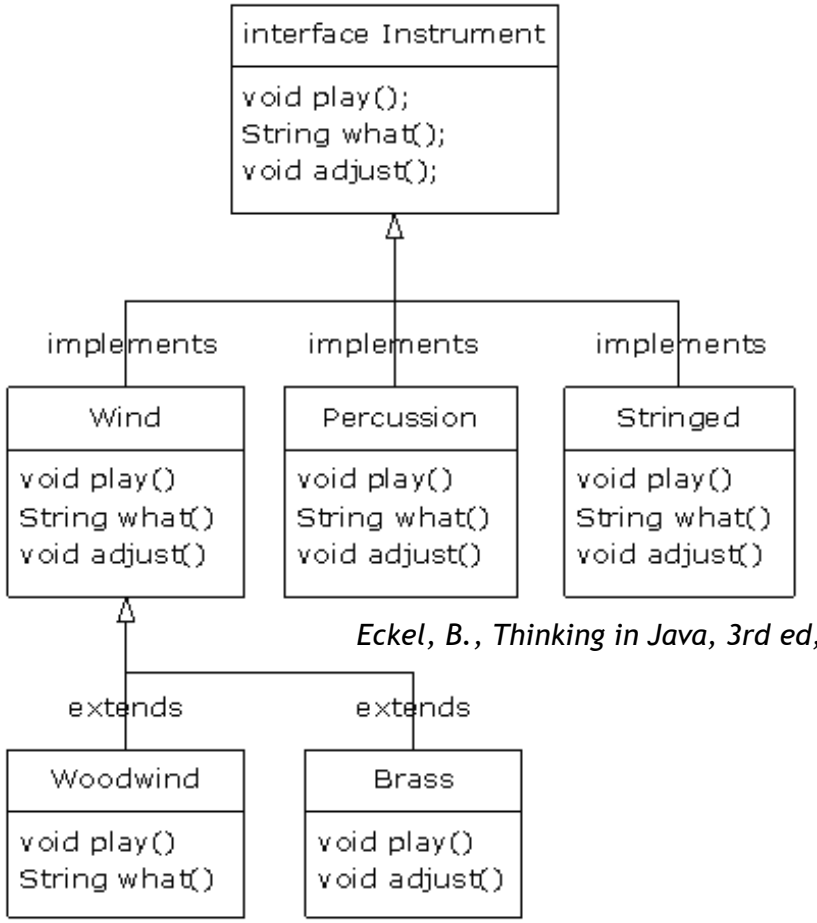
  - constants as static final fields

- **No implementation at all**

  - the interface mechanism establishes a "protocol" between classes, i.e., ***what all classes that implement this particular interface will look like***

  - the actual implementation will be done by the classes that implements the interface

Review: Java Interfaces (cont.)



Eckel, B., Thinking in Java, 3rd ed, 2002

- We **implement** interfaces; we **extend** classes
- We can only extend **one** class in java, but can implement **many** interfaces
- When we "extend" a class, any methods with the same name will override the parent class' method
- When we "implement" a class, we are adding our own method code to a method declaration (so, no overriding takes place)

Summary

- Different types of data type (primitive, reference, ADT).
- Data structure is how we implement the data in an abstract data type.
- Java Collection Framework is a hierarchy, with interfaces and abstract classes at every level except the lowest.
- Abstract class is a class that contains at least one abstract method or a class which is declared abstract.

Next Topic…

- Linear list
  - Array
    - Concept
    - Implementation
    - Application

References

- Carrano, F. & Savitch, W. 2005. *Data Structures and Abstractions with Java, 2nd ed. Prentice-Hall.*
- Malik D.S, & Nair P.S., Data Structures Using Java, Thomson Course Technology, 2003.
- Rada Mihalcea, CSCE 3110 Data Structures and Algorithm Analysis notes, U of North Texas.