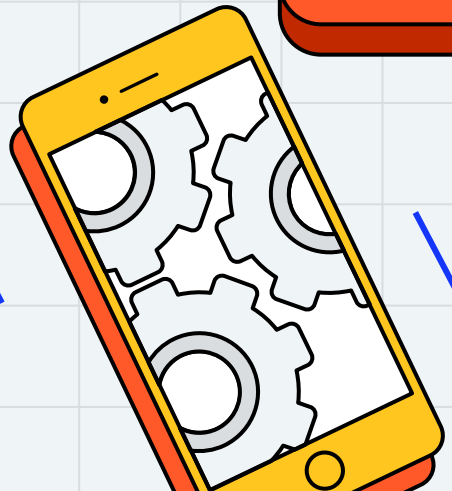
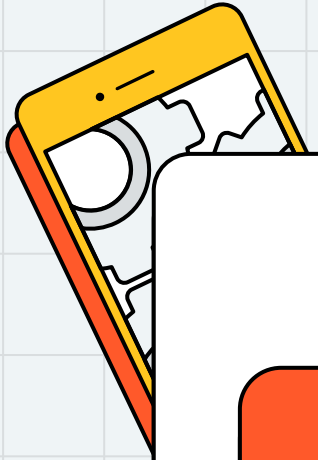
# Application Programming

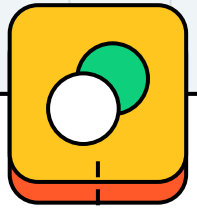


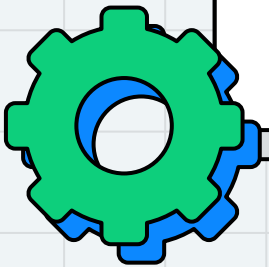
Hend Alkittawi



# Java Collections

Collections are Java’s Data Structures





# JAVA COLLECTIONS

- In Java, **Collection** is an interface. It is the root interface in the collection hierarchy. A collection represents **a group of objects**, known as its elements.
- Some collections allow **duplicate elements** and others do not. Some are **ordered** and others unordered.
- The java.util package contains many Java collections, a few of the most common are
  - List
  - Set
  - Map

# JAVA COLLECTIONS

- The **Collection** interface defines the behaviors of a collection, including typical operations such as
  - add elements to the collection .add()
  - access elements of the collection .get()
  - loop over the elements in the collection iterator
  - access an element by its index, if applicable iterator
  - test whether an element is contained in the collection .contains()
  - find out the size of the collection .size()
  - remove elements from the collection .remove()

# JAVA COLLECTIONS HISTORY

- JDK 1.0: Vector, Dictionary, Hashtable, Stack, Enumeration
- JDK 1.2: Collection, Iterator, List, Set, Map, ArrayList, HashSet, TreeSet, HashMap, WeakHashMap
- JDK 1.4: RandomAccess, IdentityHashMap, LinkedHashMap, LinkedHashSet
- JDK 1.5: Queue, …
- JDK 1.6: Deque, ConcurrentSkipListSet/Map, …
- JDK 1.7: TransferQueue, LinkedTransferQueue

# JAVA COLLECTIONS AND GENERICS

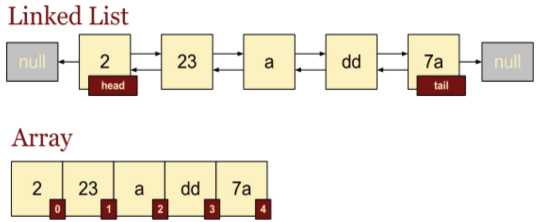
- Leveraging generics when initializing a collection is a common practice.
- Typically, a collection is declared by including the type of elements it contains within <...>, which is using Java's generics notation.
- For example

```
// declare list to be a collection of Strings
Collection<String> list;

// initialize list to a concrete class that implements Collection
list = new ArrayList<String>();
```

# JAVA LISTS

- The List **interface** represents an **ordered** collection (also known as a sequence).
- Some of the classes that implement the List interface are
  - ArrayList which is a resizable-array implementation of the List interface.
  - LinkedList which is a doubly-linked list implementation of the List and Deque interfaces.




| | add | remove | get | contains |
|---|---|---|---|---|
| ArrayList | O(1) | O(n) | O(1) | O(n) |
| LinkedList | O(1) | O(1) | O(n) | O(n) |

# JAVA LISTS

- An example for using a LinkedList

```
List<String> colorList = new LinkedList<String>();
colorList.add("red");
colorList.add("yellow");
colorList.add("blue");
```

# JAVA SETS

- The Set interface represents a collection that contains **no duplicate** elements.
- Some of the classes that implement the List interface are
    - HashSet
    - TreeSet

# JAVA SETS

- An example for using a HashSet

```
String[] colors = {"red","white","blue","green","gray","orange","tan","white","cyan","peach","gray","orange"};
List<String> list = Arrays.asList(colors);
System.out.printf("List: %s%n", list);
```

This Arrays class contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static factory that allows arrays to be viewed as lists.

prints …

```
List: [red, white, blue, green, gray, orange, tan, white, cyan, peach, gray, orange]
```

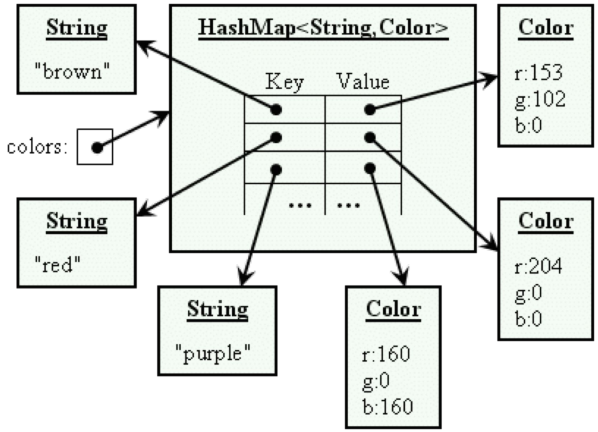
```
String[] colors = {"red","white","blue","green","gray","orange","tan","white","cyan","peach","gray","orange"};
List<String> list = Arrays.asList(colors);
Set<String> set = new HashSet<String>(list);
System.out.printf("Set: %s%n", set);
```

prints …

```
Set: [tan, green, peach, cyan, red, orange, gray, white, blue]
```

# JAVA MAPS

- The Map interface represents an object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.
- Some of the classes that implement the List interface are
  - HashMap
  - TreeMap



# JAVA MAPS

- An example for using a HashMap

```
// Not using maps, maintain two arrays, one for names and one for IDs
String[] studentNames = {"Alice", "Bob", "Carlos", "Diane"};
String[] studentIDs = {"atf123", "ght456", "liw789", "pwt012"};
// then to print out Alice, we need to know she is at index 0
System.out.println( studentIDs[0] );

// Instead, use a map!
Map<String,String> classMap = new HashMap<String,String>();
classMap.put("atf123", "Alice");    //As students register for the class,
classMap.put("ght456", "Bob");      // you can add them to the map. Then to
classMap.put("liw789", "Carlos");   // retrieve them, you only need their ID.
classMap.put("pwt012", "Diane");
System.out.println( classMap.get("atf123") );
```

# JAVA MAPS

- Another example for using a HashMap

```
// Phone book implementation
Map<String,PhoneNumber> phoneBook = new HashMap<String,PhoneNumber>();
phoneBook.put("Alice", new PhoneNumber("210-555-1234"));
phoneBook.put("Bob", new PhoneNumber("210-555-4321"));
phoneBook.put("Carlos", new PhoneNumber("210-555-4444"));
phoneBook.put("Diane", new PhoneNumber("210-555-1111"));
System.out.println( phoneBook );
```

```
public class PhoneNumber{
    private String number;

   public PhoneNumber( String phoneNumber ){
       this.number = phoneNumber;
   }

    @Override
    public String toString() {
        return "PhoneNumber [number=" + number + "]";
    }
}
```

prints …

```
{Bob=PhoneNumber [number=210-555-4321], Alice=PhoneNumber [number=210-555-1234], Diane=PhoneNumber [number=210-555-1111],
 Carlos=PhoneNumber [number=210-555-4444]}
```

# JAVA MAPS

- Another example for using a HashMap

```
Map<String, ArrayList<String>> states = new HashMap<String,ArrayList<String>>();

ArrayList<String> tx = new ArrayList<String>();
tx.add( "San Antonio" );
tx.addAll( Arrays.asList("Austin", "Dallas", "Corpus Christi", "El Paso") );
states.put("Texas", tx );

ArrayList<String> ny = new ArrayList<String>();
ny.addAll( Arrays.asList("NYC", "Albany", "Niagara", "Long Island") );
states.put("New York", ny );
System.out.println( states );
```

prints …

```
{New York=[NYC, Albany, Niagara, Long Island], Texas=[San Antonio, Austin, Dallas, Corpus Christi, El Paso]}
```

# CLASS ACTIVITY

- What collection types would you use in the following examples?
  - A phone book (name, phone number)
  - Storing user interaction history (clicks, actions, choices, etc)
  - An address book (name, phone number, address, etc)
  - User choices for character attributes in a game (hair color, shoes, etc)
  - Ordered task manager

# CLASS ACTIVITY

- What collection types would you use in the following examples?
  - A phone book (name, phone number)
    - Map
  - Storing user interaction history (clicks, actions, choices, etc)
    - List
  - An address book (name, phone number, address, etc)
    - Map
  - User choices for character attributes in a game (hair color, shoes, etc)
    - Set
  - Ordered task manager
    - List

# CLASS ACTIVITY

- Come up with 3 distinct applications that
  - Require a List
  - Require a Set
  - Require a Map

# CLASS ACTIVITY

- Come up with 3 distinct applications that
  - Require a List
    - Groceries list/High scores/List of images/To do list/Assignments/Labs
  - Require a Set
    - Enrollment UTSA/Census/UTSA IDs/Grocery list!/Medical files/Word count
  - Require a Map
    - Login info/UTSA schedule/Dictionary/Word count/parking spots

# THANK YOU!

## DO YOU HAVE ANY QUESTIONS?

hend.alkittawi@utsa.edu

By Appointment

Online