Week 3 Notes

Prof Bill - Apr 2018

Java Collections Framework = JCF

Week 3 notes covering:

A. JCF, part 1: intro, lists

B. JavaFX

C. JCF, part 2: Sets, Maps, Collections, Stream

D. Hash tables (in a separate doc)

thanks… yow, bill

A. JCF: Intro, Lists

**** Book: Muganda Ch 19.1-19.2**

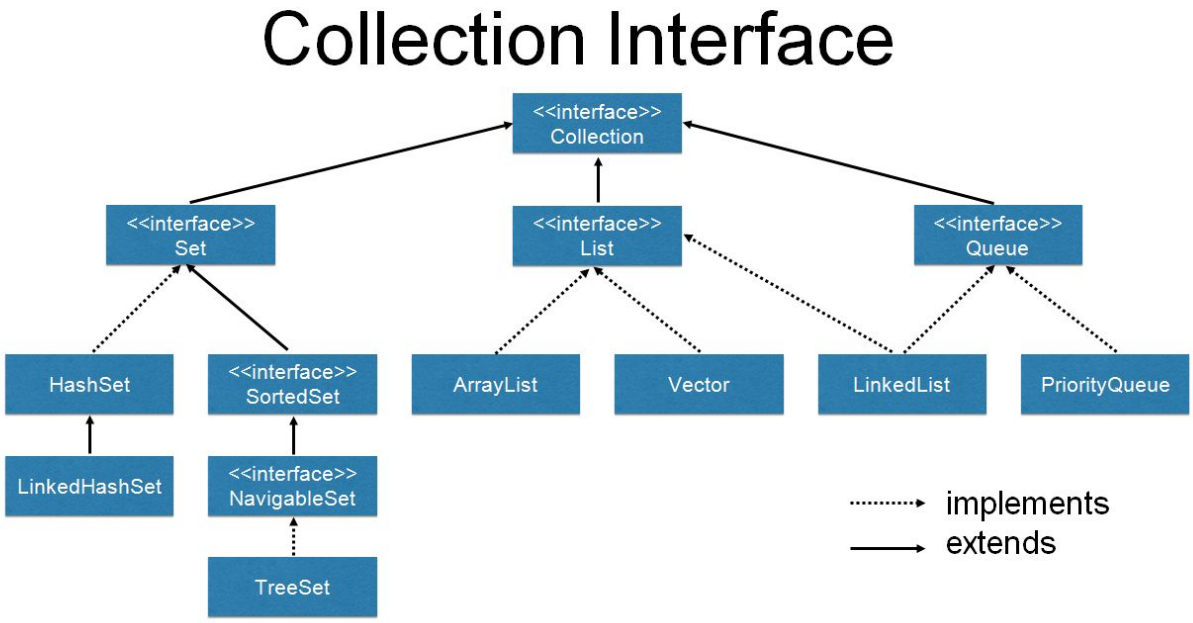
19.1 Intro

Java Collections Framework = JCF
collection: object that contain other objects
3 types of collections: list, set, map

- list - ordered collection
- set - unordered, no duplicates
- map - key-value pairs, quick retrieval by key

JCF is **generic**, so Collection<T>
JCF is part of the **util package** in Java. This Javadoc is invaluable when coding!
docs.oracle.com/javase/10/docs/api/index.html?java/util/package-summary.html

UML for class hierarchy - part 1: Collection interface, List and Set
Source: dzone.com/articles/an-introduction-to-the-java-collections-framework

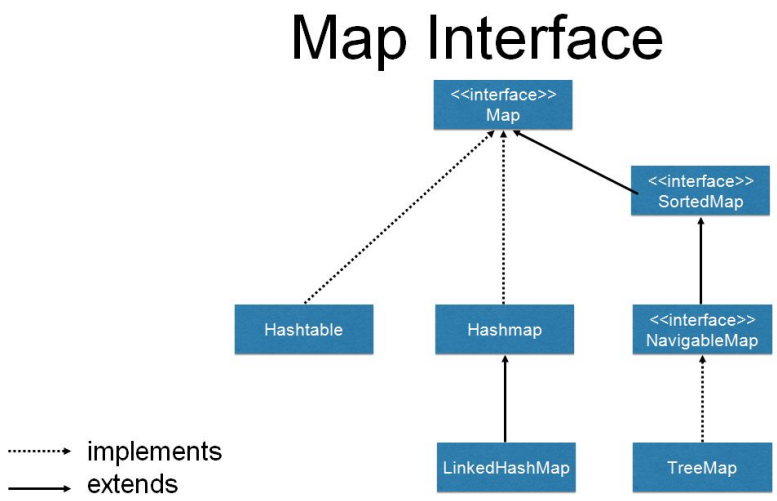


Collection interface methods:

add(Object), addAll(Collection), clear(),
contains(Object), containsAll(Collection), isEmpty(),

remove(Object), removeAll(Collection), size(), stream(), toArray()

Part 2 is Map, which has a radically different set methods; little overlap with Collection, so Map has its own interface:



Iterator for detailed control over looping

More common, foreach loop (built by Java using Iterator)

```
for( String name: nameList) {
    // do something here to each name in the list
}
```

Or of you prefer, Java functional interface to do similar things:

```
nameList.forEach(
    x ->
        {
            System.out.println("%s %d\n", x, x.length);
        });
```

19.2 Lists

Two Lists: ArrayList, LinkedList

List interface methods: 1) inherits all Collection methods, and 2) and these:

add(int pos), addAll(Collection), get(i), indexOf(Object), remove(int pos), sort(Comparator)

ArrayList, LinkedList is-a List; can use the super class in declaration (polymorphism)

```
List<String> nameList = new ArrayList<>();
```

ListIterator methods give detailed control over iteration of List

B. JavaFX

**** Book: Muganda Ch 15**

My JavaFX notes zoom by. The concepts here aren't the challenge. Gui coding is very hands-on, trial and error. Get in there, google it, try it, rinse and repeat.
thanks… yow, bill

15.1 Intro

JavaFX = Java gui library, newer and more popular (?) than Swing
Couple links:

- Javadoc for the API online, docs.oracle.com/javase/10/docs/api/index.html?javafx.graphics-summary.html

- Tutorials:
    - Loos nice, www.tutorialspoint.com/javafx/index.htm
    - The official tutorial (so it must be bad?), docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm -
    - google "javafx tutorial" to find something better!

- Hello, World, docs.oracle.com/javafx/2/get_started/hello_world.htm

event-driven gui - write event listener code, methods are called when specific user actions take place

/* if you're totally new to gui… you may want to read/skim Muganda Ch 12 and 13 */

15.2 Stages and Scenes

the metaphor... an application is a **scene** played out on a **stage**

Muganda Code Listing 15-1 - great example of the simplest JavaFX application:

```
import javafx.application.Application;
import javafx.stage.Stage;

public class SimpleJavaFXApp extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage stage) {
        stage.setTitle("Simple JavaFX Application");
        stage.show();
    }
}
```

15.3 Scene Graph and Nodes

Organization:

- “A GUI consists of a scene graph, which is itself comprised of scene graph nodes”
- “A scene is always formed from a single node called the root of the scene”

15.4 Panes and Component Layout

VBox and **HBox** - nice, simple panes; very useful!

15.5 Events and Event Handling

Important classed: EventHandler, ActionEvent
4 choices for event handlers:

- ➢ Separate class, Muganda Code Listing 15-5
- ➢ Inner class, Muganda Code Listing 15-7
- ➢ Anonymous class
- ➢ Lambda functions, Muganda Code Listing 15-8

/* style choice is yours, whatever you prefer; lambda functions are usually preferred (aka cooler) over anonymous classes */

15.6 Determining the Target of an Event

Use getTarget() method in your listener

15.7 Radio Buttons and CheckBoxes

radio buttons - choose from multiple items, **RadioButton** class
checkboxes - yes/no choice, **CheckBox** class
Use **ToggleGroup** class to group choices for radio button

15.8 Displaying Images

Use Image class
See Muganda Code Listing 15-11

15.9 Timeline Animation

Timeline class for simple frame-by-frame animation.
Muganda Code Listing 15-12 for super-simple example

15.10 Text Input Control, Panes, CSS

TextInputControl abstract class defines text input methods
CSS = Cascade Style Sheet, critical component in formatting web pages
Panes for fancy placement - TilePane, BorderPane, GridPane

C. JCF: Sets, Maps, etc

**** Book: Muganda Ch 19.3-19.6**

19.3 Sets

sets are unordered collections with no duplicates
Java says:

> A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.
>
> - docs.oracle.com/javase/tutorial/collections/interfaces/set.html

Different implementations of Set:

- **HashSet -** is-a Set, implements Set with a hash table, uses **hashcode()** method, inherited from Object
- **LinkedHashSet -** it's HashSet with a linked list added to preserve order (meh)
- **TreeSet** - implements Set with a binary tree

SortedSet - not an implementation, an interface for sorting elements in a Set, example: TreeSet is-a SortedSet.

Comparable vs. Comparator

This is important - 2 ways to compare objects (for sorting, searching, everything!):

- ➢ **compareTo()** method, inherited from **Comparable** interface
- ➢ **Comparator** interface

This is a nice example to walk through… Player class, 1) order by rank using Comparable, and 2) order by rank or age using Comparable.

www.baeldung.com/java-comparator-comparable

Comparable	Comparator
1) Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc.	Comparator provides **multiple sorting sequence**. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc.
2) Comparable **affects the original class** i.e. actual class is modified.	Comparator **doesn't affect the original class** i.e. actual class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is found in **java.lang** package.	Comparator is found in **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List,Comparator)** method.

Source: www.javatpoint.com/difference-between-comparable-and-comparator

/* HW #3 grudge match: Comparable vs. Comparator! */

19.4 Maps

maps store (key, value) pairs; each key has one value; key -> value access is fast
Java says:

> A Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.
> The Java platform contains three general-purpose Map implementations: HashMap, TreeMap, and LinkedHashMap. Their behavior and performance are precisely analogous to HashSet, TreeSet, and LinkedHashSet, as described in The Set Interface section.
>
> - docs.oracle.com/javase/tutorial/collections/interfaces/map.html

Catch that? Implementations are similar to Set.
They're **HashMap**, **TreeMap**, and **LinkedHashMap**. **SortedMap** interface, too.

Some new methods in Map (types are K=key, V=value):

> V get(K) - get value for this key
> put(K, V) - put (key, value) in map
> V remove(K) - remove (key, value) from map
> Set<K> keySet() - create set of all keys in map
> Collection<V> values() - create collection of all values in map
> And… containsKey(K), containsValue(V), clear(), isEmpty()

19.5 Collections

These are some **very useful** static methods for Collection objects!
docs.oracle.com/javase/10/docs/api/index.html?java/util/Collections.html

Most popular methods are:

- binarySearch() - with Comparable or Comparator
- sort() - with Comparable or Comparator
- max(), min() - with Comparable or Comparator

- copy()
- reverse() - reverse the order of elements
- shuffle() - randomize!

19.6 Stream

Coming soon…