Java's Collection Framework

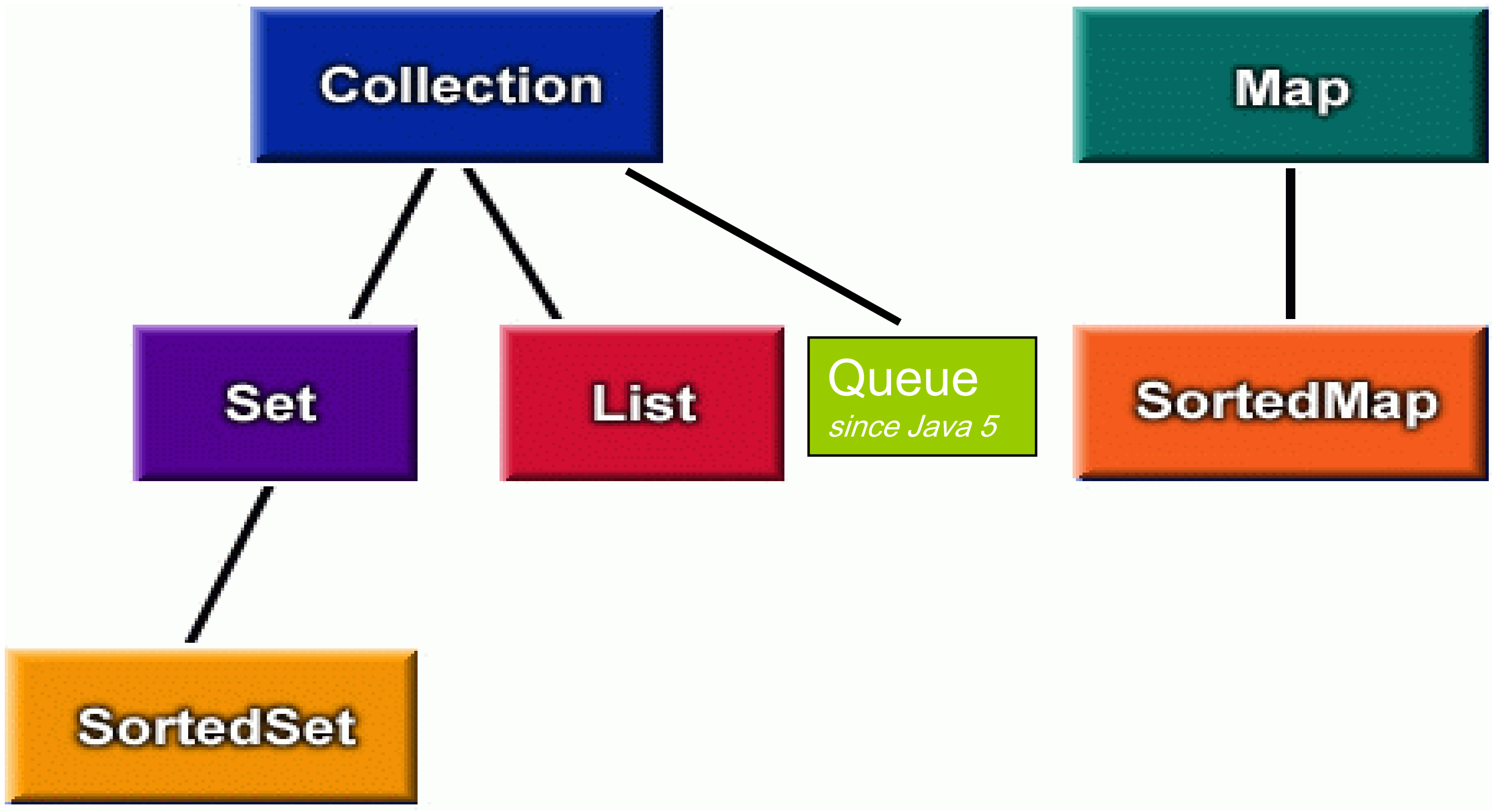
Collection Framework

- A *collections framework* is a unified architecture for representing and manipulating collections. It has:
  - **Interfaces:** abstract data types representing collections
  - **Implementations:** concrete implementations of the collection interfaces
  - **Algorithms:** methods that perform useful computations, such as searching and sorting
    - These algorithms are said to be *polymorphic*: the same method can be used on different implementations

Interfaces

- An interface describes a set of methods:
  - no constructors or instance variables
- Interfaces must be implemented by classes
  - 646 java classes implement >= 1 interfaces (‘02)
- 2 or more classes implement an interface
  - Classes guaranteed to have the same methods
  - Objects can be treated as the same type
  - Can use different algorithms / instance variables

Collection interfaces



Implementations

- A collection class
  - implements all methods of the interface
  - selects appropriate instance variables
  - can be instantiated
- Java implements interfaces with
  - **`List:  ArrayList`**, **`LinkedList`**, **`Vector`**
  - **`Map:  HashMap`**, **`TreeMap`**
  - **`Set:  TreeSet`**, **`HashSet`**

Algorithms

- Java has *polymorphic* algorithms to provide functionality for different types of collections
  - Sorting (e.g. sort)
  - Searching (e.g. binarySearch)
  - Finding Extreme Values (e.g. max)

Two Useful ADTs

- List: a collection with a first element, a last element, distinct predecessors and successors
  - duplicates that "equals" each other are allowed
- Map: maps keys to values
  - Maps cannot contain duplicate keys
  - Each key maps at most one value

List

- A list is
  - a collection of elements (numbers, strings, accounts, pictures,…)
  - ordered (a sequence): there is a first, and a last element
    - lists can be empty – no elements
  - elements with a unique predecessor and successor
  - also known as a sequence

`ArrayList` *A Java Collection Class that Implements* `List`

- ArrayList
  - stores a collection of any type of object
  - can be all the same type, or different types
  - similar functionality to an array, but does not use subscripts
  - is an indexed collection

`List` *implemented by 3 classes*

```
// Interface name: List
// Three classes that implement the List interface:
List<String> bigList = new ArrayList<String>();
List<String> littleList = new LinkedList<String>();
List<String> sharedList = new Vector<String>();

// All three have an add method
bigList.add("in array list");
littleList.add("in linked list");
sharedList.add("in vector");
```

Generics

- A **`List`** can be made to store only one type

```
List<BankAccount> accountList =
                  new ArrayList<BankAccount>();

accountList.add(new BankAccount("One", 111.11));
accountList.add(new BankAccount("Two", 222.22));
accountList.add(new BankAccount("Three", 333.33));
accountList.add(new BankAccount("Four", 444.44));
```

Iterators

- Iterators provide a general way to traverse all elements in a collection

```
ArrayList<String> list = new ArrayList<String>();
list.add("1-FiRsT");
list.add("2-SeCoND");
list.add("3-ThIrD");
Iterator<String> itr = list.iterator();
while (itr.hasNext()) {
  System.out.println(itr.next().toLowerCase());
}
```

Output

```
1-first
2-second
3-third
```

Enhanced for loop

- If a class **`extends Iterable<E>`** (like you did with your **`Set<E>`** class), you can use Java's enhanced for loop of this general form

  `for` **`(`***E refVar* **`:`** *collection<E>* **`) {`**

  refVar refers to each element in *collection<E>*

  `}`

  – example

```
ArrayList<String> list = new ArrayList<String>();
list.add("first");
for (String s : list)
  System.out.println(s.toLowerCase());
```

Hash Map

```
Set set = hm.entrySet();

Iterator i = set.iterator();
```

Note: An iterator for an map should by casted with set

Looping Map

```
while(i.hasNext()){
     Map.Entry me = (Entry) i.next();

     System.out.println(me.getKey() + " : "
          + me.getValue());
}
```

get and put with Map

```
double balance = ((Double)
hm.get("Naveen")).doubleValue();

hm.put("Naveen", new Double(balance +
1000));

System.out.println("New Balance of Naveen is
" + hm.get("Naveen"));
```