Java collections framework

Commonly reusable collection data structures

Abstract Data Type

- ADTs store data and allow various operations on the data to access and change it
- ADTs are mathematical models
- ADTs are extremely useful when designing a complex algorithms
- ADTs are not classes (well, not quite)

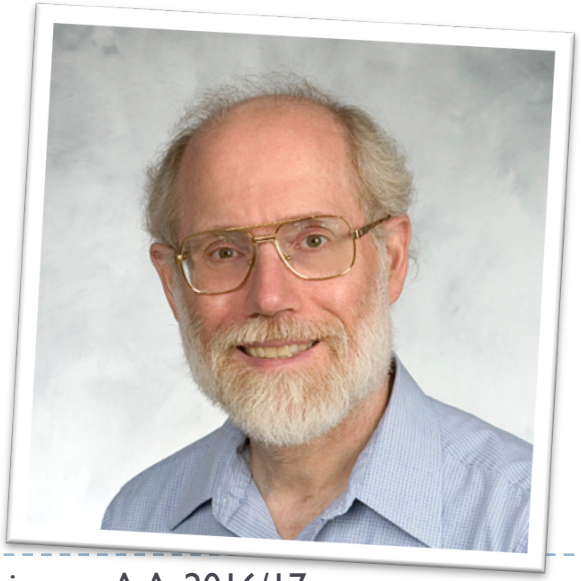
Abstract Data Type

- ADTs are "abstract" because they specify the operations of the data structure and leave *implementation details* to later
- More similar to "abstract classes" or "interfaces" (whether the language supports them)
- **Note:** Not all implementation details can be deferred!

Why study ADTs?

- How many of you will actually go out and create your own ADT from scratch?
- Different ADTs, each one with its own pros and cons
- Picking the right one for the job is an important step in design!
- ***Get your data structures correct first, and the rest of the program will write itself***

  David S. Johnson
  (winner of Knuth's Prize in 2010)

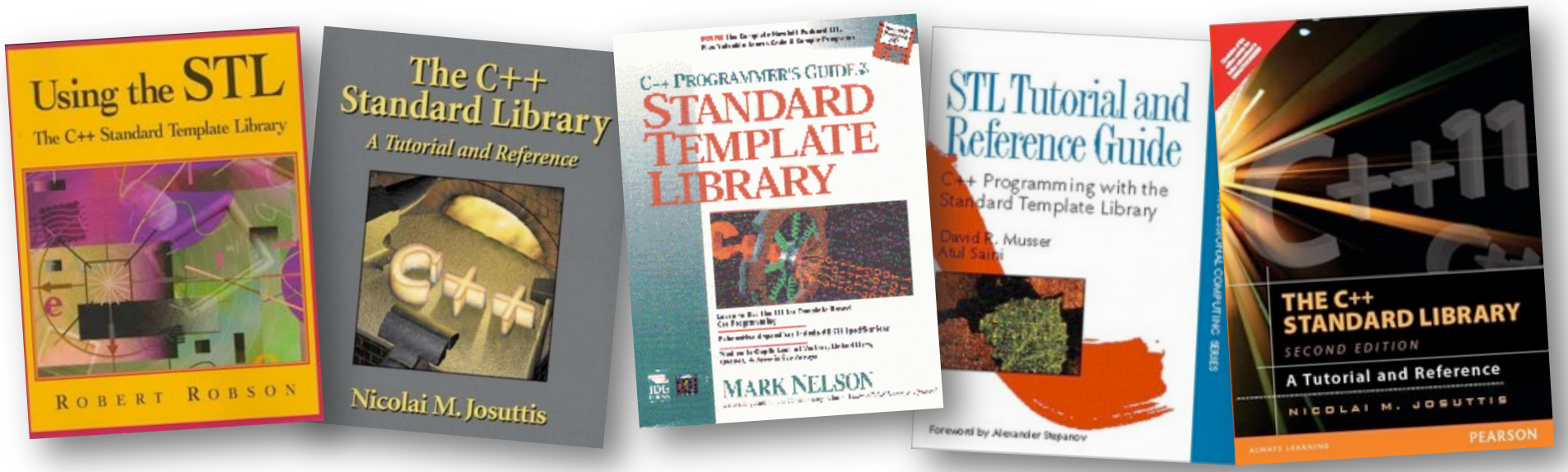


Why study ADTs?

The goal is to learn how to learn how to use and create ADTs

Built-in ADT

- High level languages often provide built in ADTs. E.g.:
  - *Standard Template Library* (C++)
  - *Java Collections Framework* (Java)



Common Ground

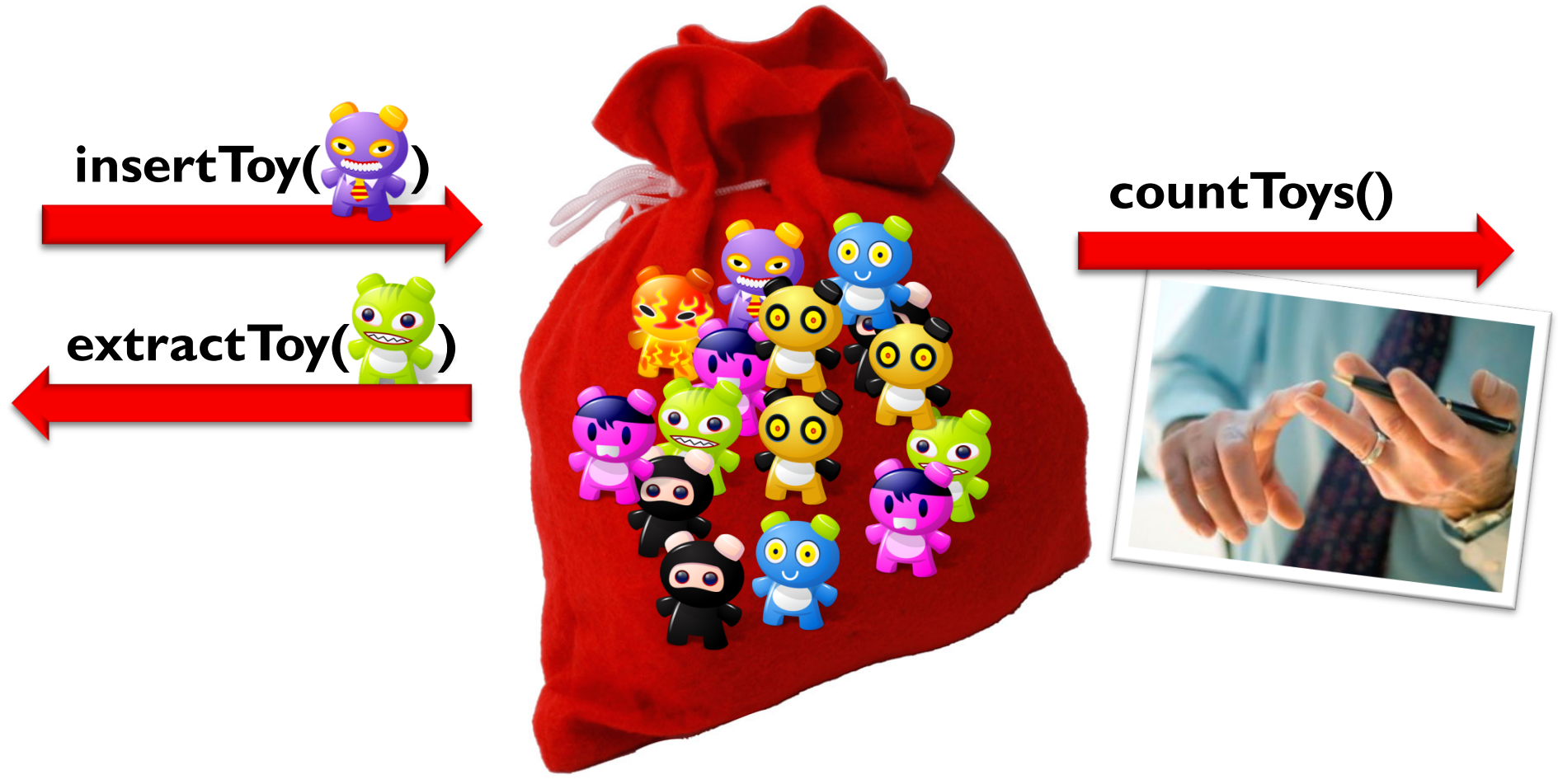
- Almost every ADT provide a way to
  - add an item
  - remove an item
  - find, retrieve, or access an item
- Most Collection ADTs provide more possibilities
  - check whether the collection is empty
  - make the collection empty
  - give me a subset of the collection
  - …

A very simple ADT: Santa's Sack

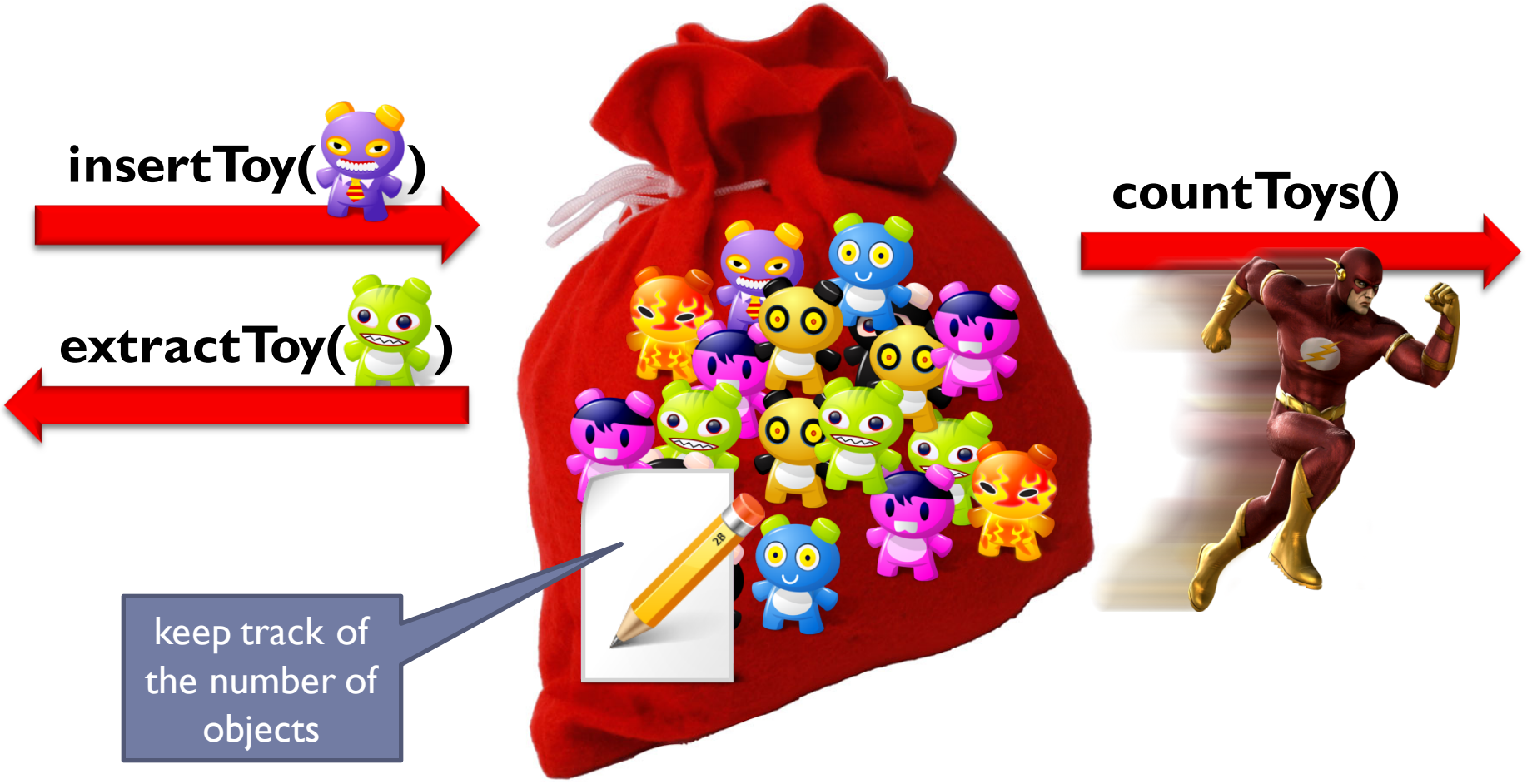
Sack's Operations

- insertToy(toy)
  - Insert a toy in the sack
  - Duplicates are – obviously – allowed
- extractToy(toy)
  - Remove the given toy from the sack
  - … and make a children happy
- countToys()
  - Count how many toys actually are stored in the sack

Santa's Sack



Santa's Sack (more efficient)

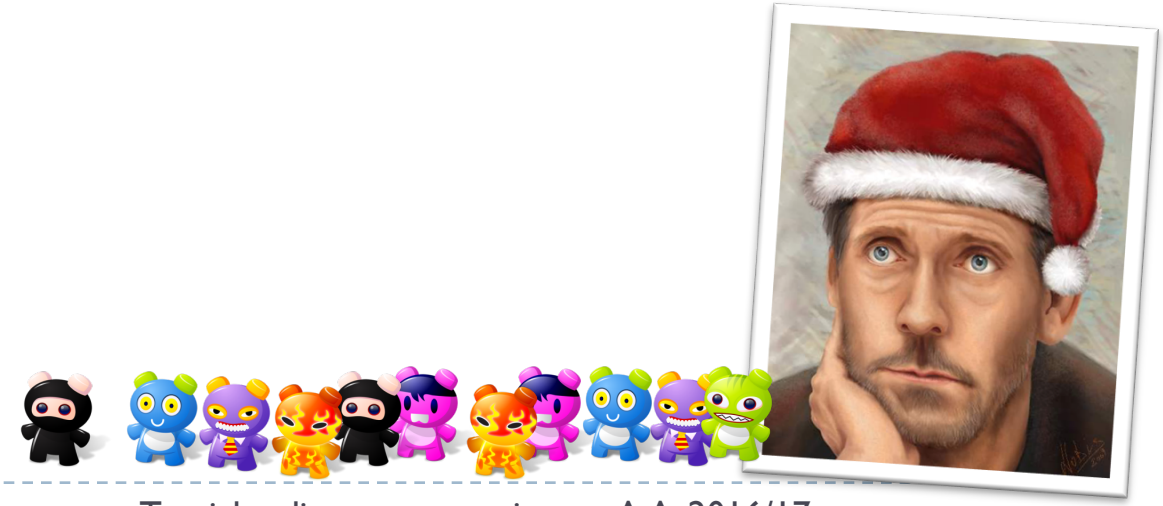


The lesson

- ADTs do not specify **the details** of the implementation

BUT

- Some information about the algorithms is essential to choose the right ADT
- Very high-level, qualitative information
- Complexity



Java Collections Framework (JCF)

- Collection
  - an object that represents a group of objects
- Collection Framework
  - A unified *architecture* for representing and manipulating collections
  - Such collections are manipulated independent of the details of their representation
  - "JCF" vs. "ADT"

A little bit of history…

- JDK < 1.2
  - Standard practice: Vector and Hashtable
  - Compatibility with C++ *Standard Template Library* (STL)
  - *Doug Lea's Collections* package
  - *ObjectSpace Generic Collection Library* (JGL)
- JDK ≥1.2
  - Sun drops compatibility with C++ STL
  - Joshua Bloch's JCF
    (now *Chief Java Architect* @ Google)

A little bit of history…

- Java 5
  - Introduction of **<generics>**
  - Clean, safe definition of the **Collection Interface**
  - **Trees**, **linked lists**, **stacks**, **hash tables**, and other classes are implementations of **Collection**
  - Arrays do not implement the Collection interface
  - **Vector** redefined to implement **Collection**

A little bit of history…

- Doug Lea later developed a concurrency package

JCF's Main Elements

- Infrastructure
  - Interfaces that provide essential support for the collection interfaces
- General-purpose Implementations
  - Primary implementations (basic and bulk) of the collection interfaces

Algorithms

- Algorithms
  - Static methods that perform useful functions on collections, such as sorting a list

ICF's Utility Implementations

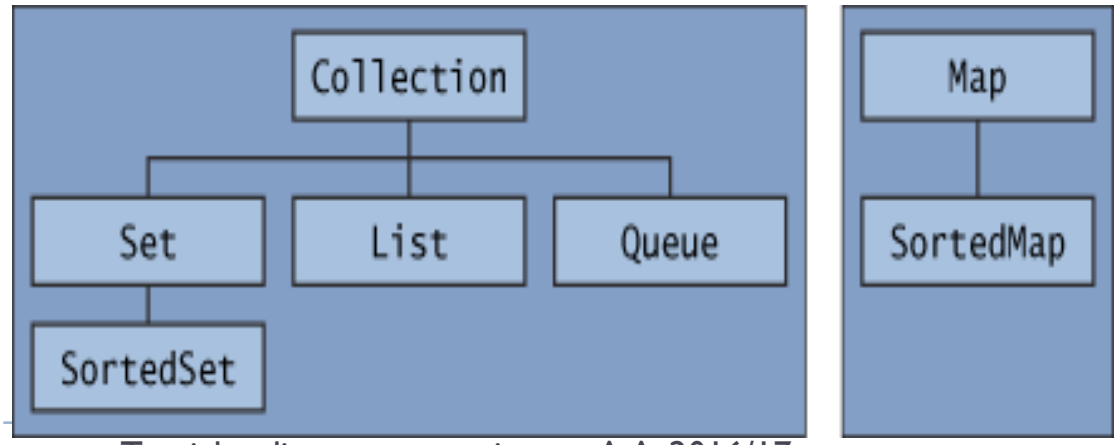
- Legacy Implementations
  - The collection classes from earlier releases, Vector and Hashtable, have been retrofitted to implement the collection interfaces
- Convenience Implementations
  - High-performance "mini-implementations" of the collection interfaces
- Wrapper Implementations
  - Add functionality, such as synchronization, to other implementations

Abstract Implementations

- Partial implementations (skeletons) of the collection interfaces to facilitate custom implementations

Infrastructure

- These interfaces form the basis of the framework
  - Some types of collections **allow duplicate** elements, others do not
  - Some types of collections are **ordered**, others are **unordered**
- The Java platform doesn't provide any direct implementations of the Collection interface, but provides implementations of more specific sub-interfaces, such as Set and List and Maps



Collection interface

- A **Collection** represents a group of objects known as its *elements*
- The Collection interface is the **least common denominator** that all collections implement.
- It is Used
  - to pass collections around
  - to manipulate them when maximum generality is desire
- **Collection** extends **Iterable**

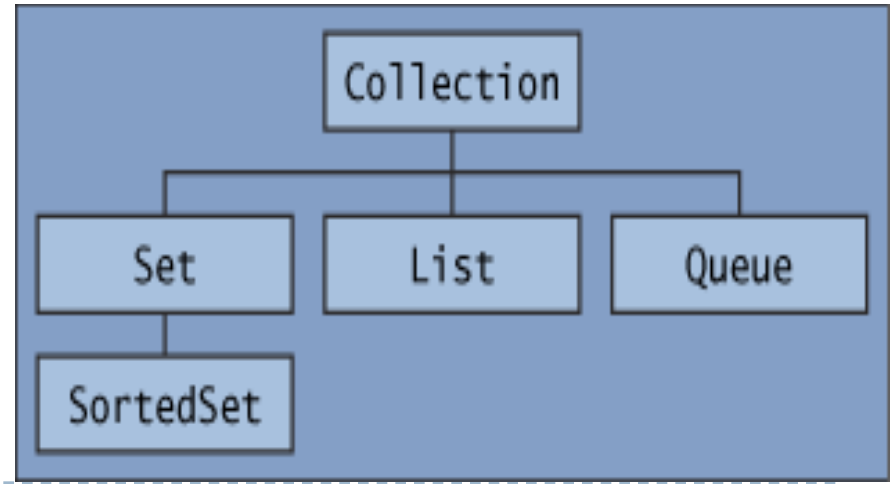
A note on iterators

- An **Iterator** is an object that enables you to traverse through a collection (and to remove elements from the collection selectively)
- You get an Iterator for a collection by calling its iterator() method.
- Several languages supports "iterators". E.g., C++, PHP, Python, Ruby, Go…

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

Main Interfaces

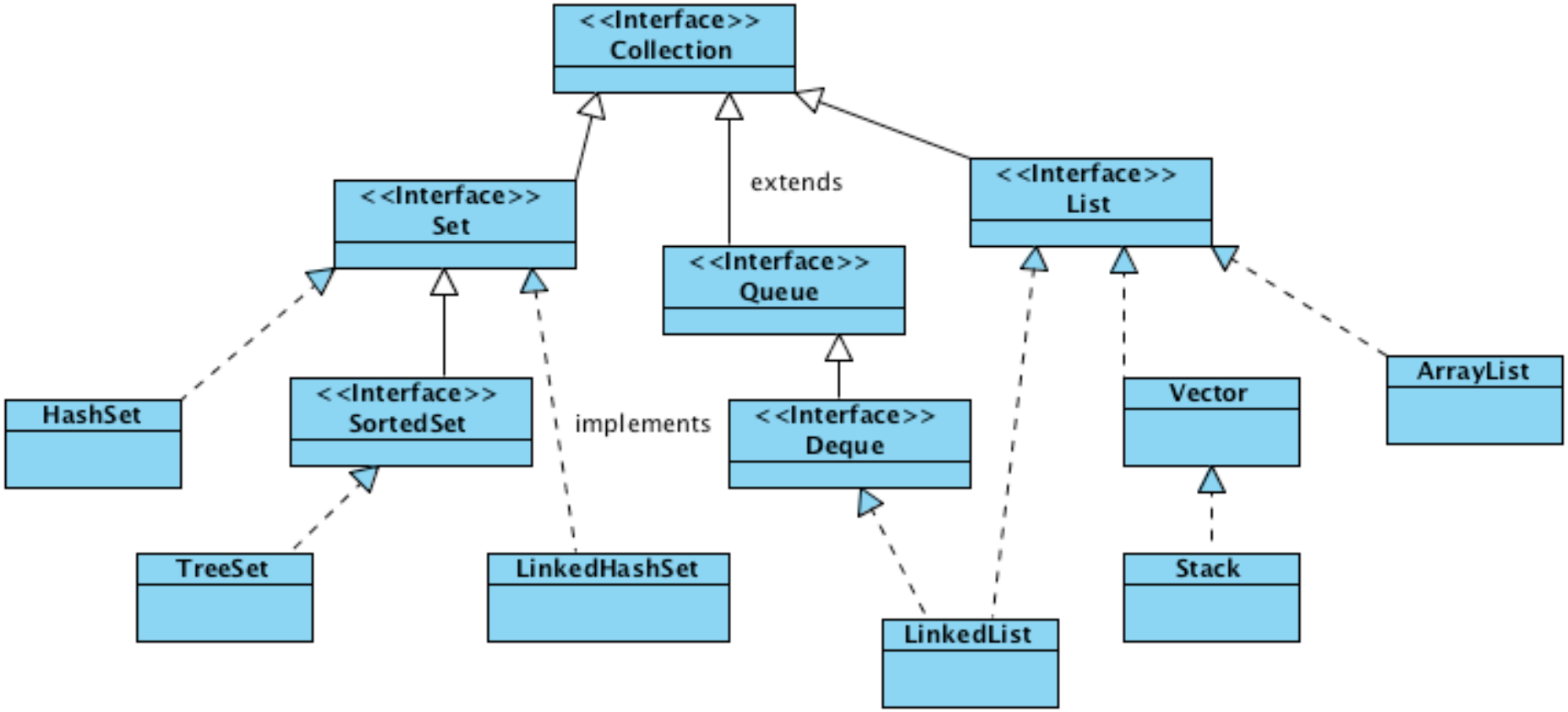
- List
  - A more flexible version of an array
- Queue & Priority Queue
  - The order of arrival does matter, or the urgency
- Set
  - No order, no duplicate elements



Map interface

- 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
- **Map** does not extend **Iterable**, but it is possible to get an iterator through **entrySet()**
- **Notez bien:** Maps do not extend from **java.util.Collection**, but they're still considered to be part of the "collections framework"

Collection Family Tree



Collection interface

```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);                          //optional
    boolean remove(Object element);                  //optional
    Iterator<E> iterator();

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);              //optional
    boolean retainAll(Collection<?> c);              //optional
    void clear();                                    //optional

    Object[] toArray();
    <T>T[] toArray(T[] a);
}
```

Collection interface

Basic Operations

```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);                          //optional
    boolean remove(Object element);                  //optional
    Iterator<E> iterator();

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);              //optional
    boolean retainAll(Collection<?> c);              //optional
    void clear();                                    //optional

    Object[] toArray();
    <T>T[] toArray(T[] a);
}
```

generics

Collection interface

```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);                          //optional
    boolean remove(Object element);                  //optional

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);        //optional
    boolean retainAll(Collection<?> c);        //optional
    void clear();                              //optional

    Object[] toArray();
    <T>T[] toArray(T[] a);
}
```

Bulk Operations

wildcard

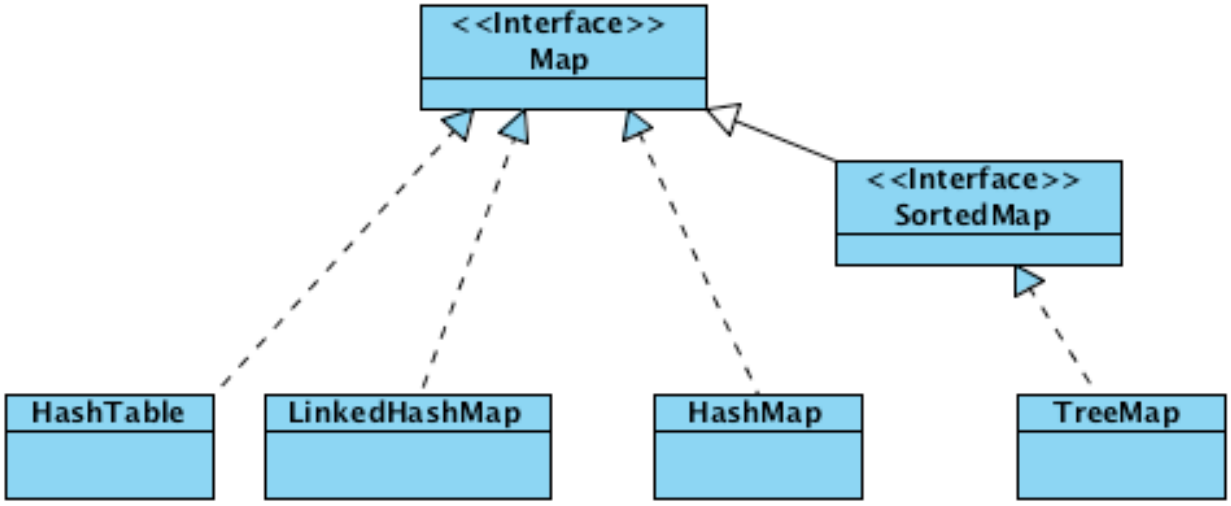
either extends
or implements

Collection interface

```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);                          //optional
    boolean remove(Object element);                  //optional
    Iterator<E> iterator();

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);              //optional
    boolean retainAll(Collection<?> c);              //optional
                                                     //optional

    Object[] toArray();
    <T>T[] toArray(T[] a);
}
```

Array Operations

Map Family Tree



Ma

Basic Operations

```
public interface Map<K,V> {
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();

    void putAll(Map<? extends K, ? extends V> m);
    void clear();

    […]
```

Map interface

```
public interface Map<K,V> {
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();

    void putAll(Map<? extends K, ? extends V> m);
    void clear();

    […]
```

Bulk Operations

Map interface

```
    […]

    public Set<K> keySet();
    public Collection<V> values();
    public Set<Entry> entrySet();

    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```

Interface for entrySet elements

Map interface

Collection Views

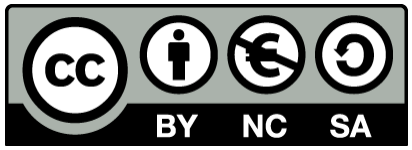
```
public Set<K> keySet();
public Collection<V> values();
public Set<Map.Entry<K,V>> entrySet();

public interface Entry {
    K getKey();
    V getValue();
    V setValue(V
}
}
```

```
for (Map.Entry<Foo,Bar> e : map.entrySet())
{
    Foo key = e.getKey();
    Bar value = e.getValue();
}
```

http://docs.oracle.com
/javase/7/docs/api
/java/util/Collection.html

http://tiny.cc/javahelp

Licenza d'uso



- Queste diapositive sono distribuite con licenza Creative Commons "Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)"
- Sei libero:
  - di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
  - di modificare quest'opera

- Alle seguenti condizioni:
  - **Attribuzione** — Devi attribuire la paternità dell'opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo i cui tu usi l'opera.

  - **Non commerciale** — Non puoi usare quest'opera per fini commerciali.

  - **Condividi allo stesso modo** — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con un licenza identica o equivalente a questa.

- http://creativecommons.org/licenses/by-nc-sa/3.0/