Java Exceptions
Java Collection API

What is an Exception

- An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions
- *exception object* - contains information about the error
- *throwing an exception* - creating an exception object and handing it to the runtime system

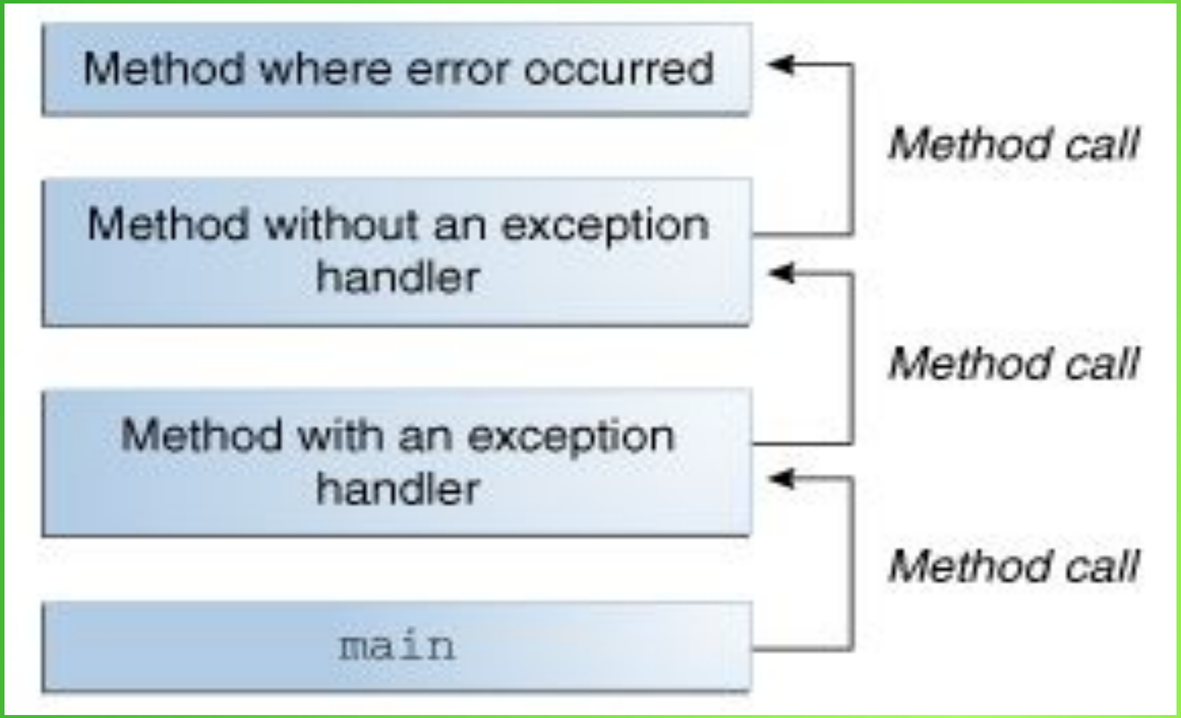
exception handler - block of code that can handle the exception

If an appropriate exception handler not found the program terminates

- try
- catch
- finally
- throw
- throws

```
public class App {
   public static void main(String[] args) throws Throwable{
   }
}
```

```
public class App {
  public static void main(String[] args) throws String {
  }
}
```



Method where error occurred

Method call

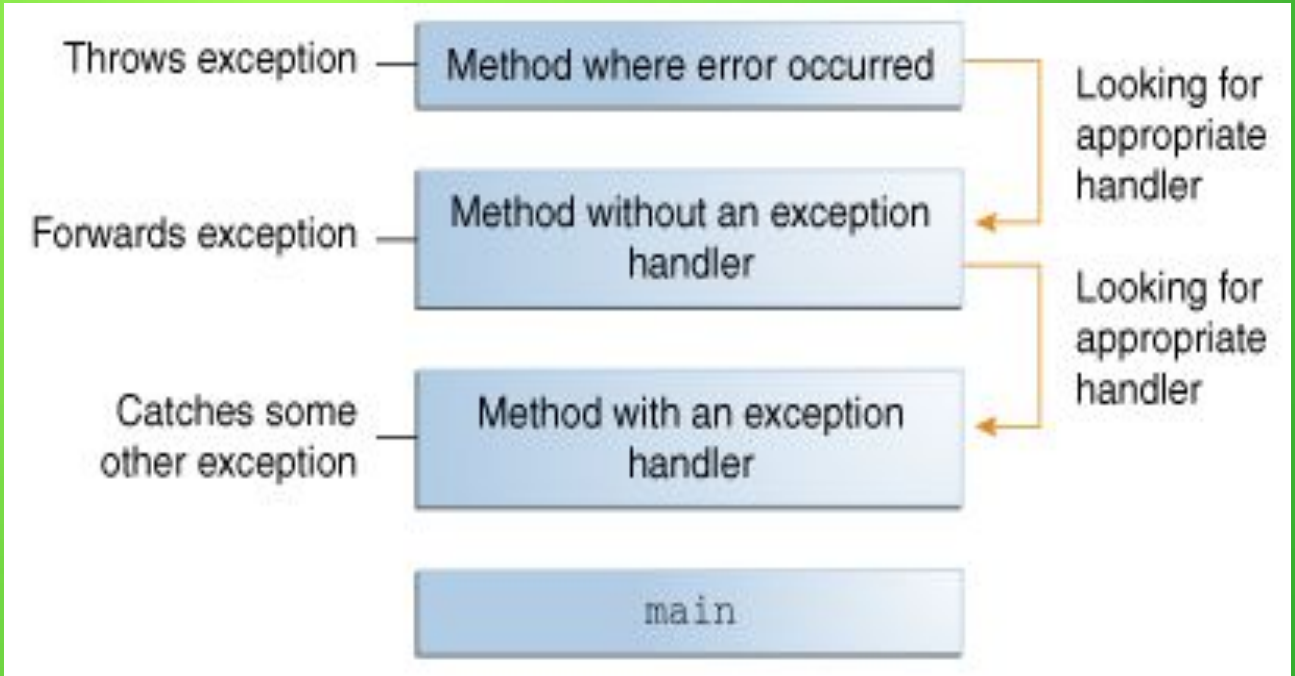
Method without an exception handler

Method call

Method with an exception handler

Method call

main



Throws exception — Method where error occurred

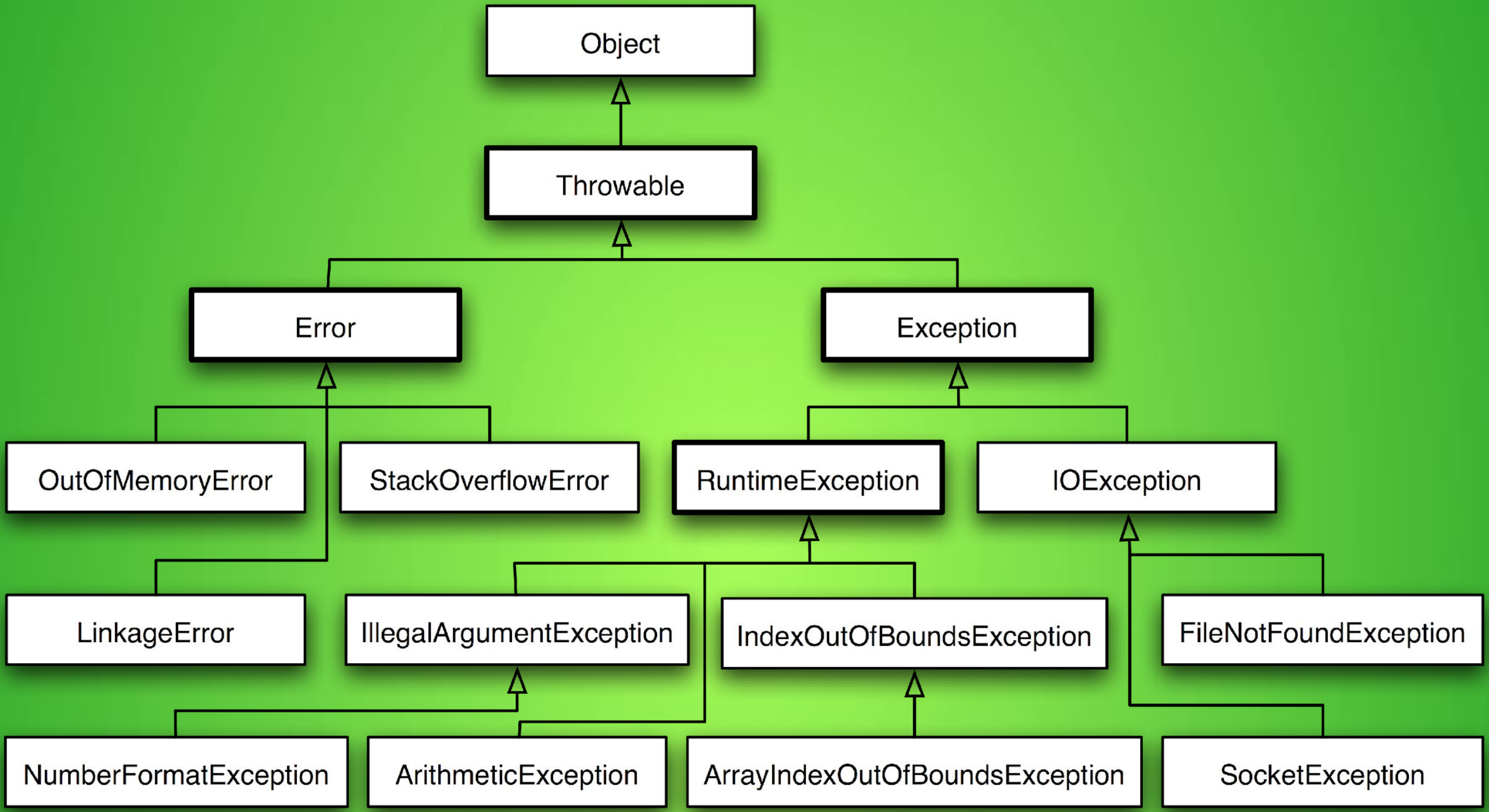
Looking for appropriate handler

Forwards exception — Method without an exception handler

Looking for appropriate handler

Catches some other exception — Method with an exception handler

main



Object

Throwable

Error

Exception

OutOfMemoryError

StackOverflowError

RuntimeException

IOException

LinkageError

IllegalArgumentException

IndexOutOfBoundsException

FileNotFoundException

NumberFormatException

ArithmeticException

ArrayIndexOutOfBoundsException

SocketException

Types

checked exception - all exceptions, except for those indicated by RuntimeException, Error, and their subclasses.

error - external to the application, which the latter can’t anticipate or recover from

runtime exception - internal to the application; usually indicate programming bugs

The code that might throw certain exceptions must be enclosed by either of the following:

A try statement that catches the exception.

A method that specifies that it can throw the exception.

```
public class App {
    public static void main(String[] args) {
        f(null);
    }

    public static void f(NullPointerException e) {
        try {
            throw e;
        } catch (NullPointerException npe) {
            f(npe);
        }
    }
}
```

```
public class App {
    public static void main(String[] args) {
        double d = sqr(10.0);
        System.out.println(d);
    }

    public static double sqr(double arg) {
        throw new Exception();
    }
}
```

```
public class App {
    public static void main(String[] args) {
        double d = sqr(10.0);
        System.out.println(d);
    }

    public static double sqr(double arg) {
        throw new RuntimeException();
    }
}
```

Collections

What Is a Collections Framework?

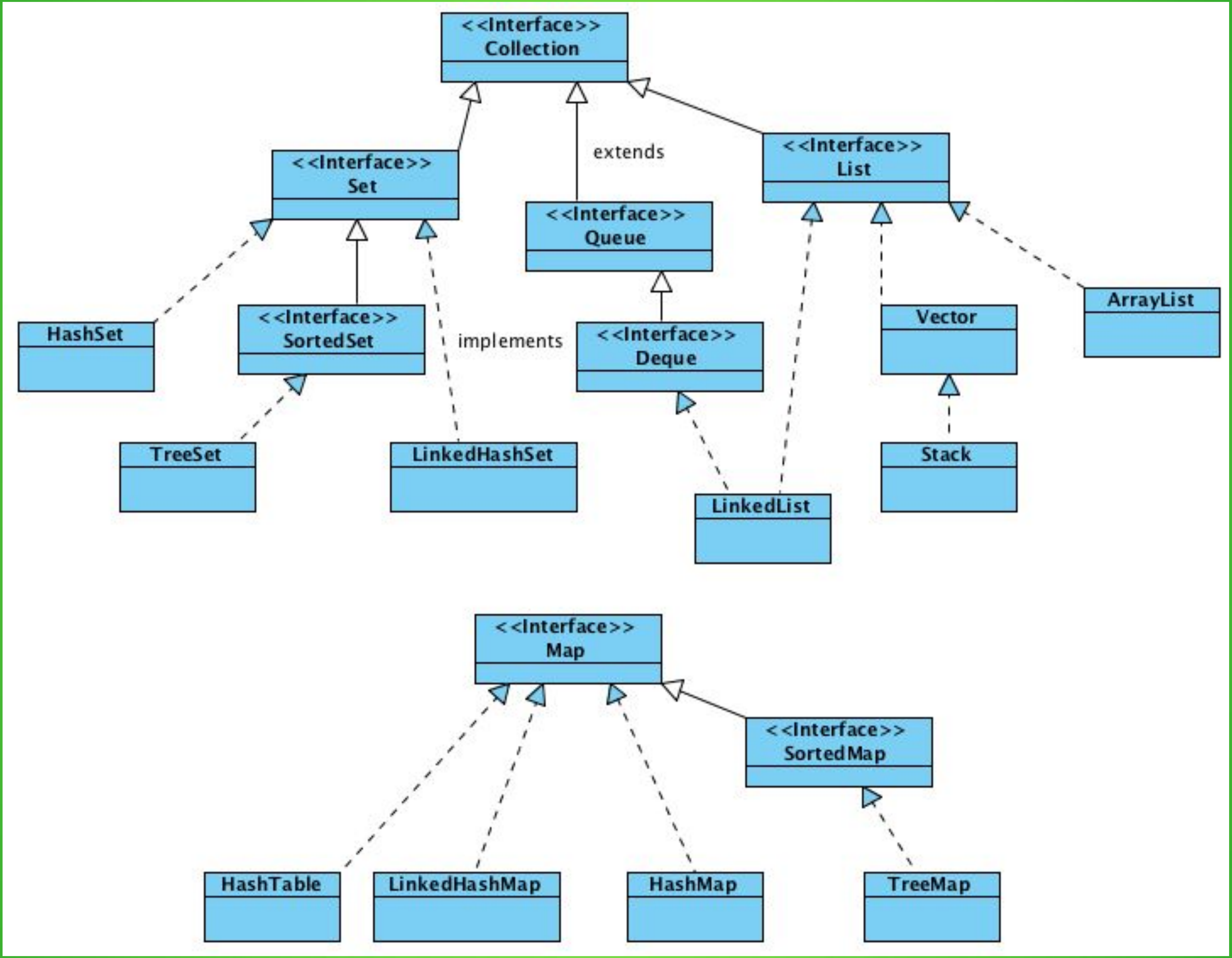
A *collection* — sometimes called a container — is simply an object that groups multiple elements into a single unit

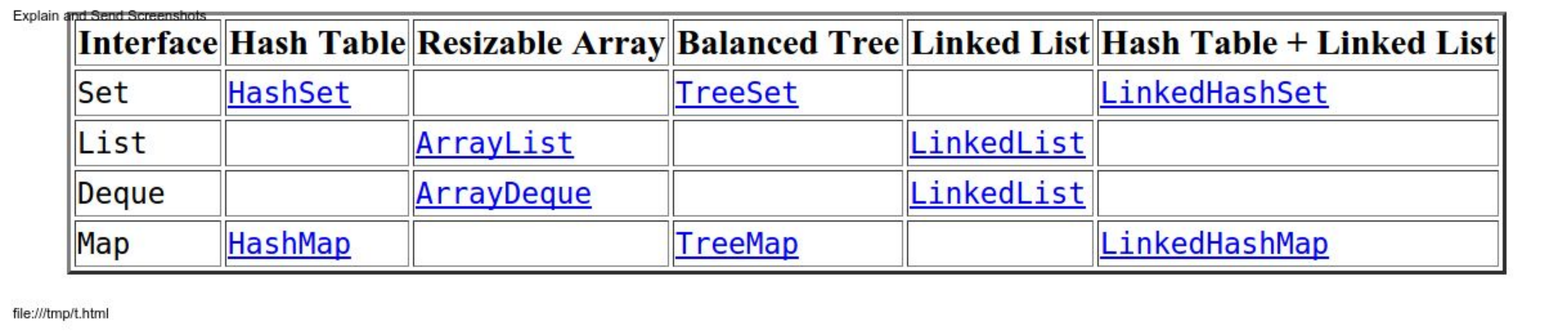
A *collections framework* is a unified architecture for representing and manipulating collections:

Interfaces - abstract data types that represent collections

Implementations - the concrete implementations of the collection interfaces

Algorithms - the methods that perform useful computations


<<Interface>>
Collection
extends
<<Interface>>
Set
<<Interface>>
Queue
<<Interface>>
List
HashSet
<<Interface>>
SortedSet
implements
<<Interface>>
Deque
Vector
ArrayList
TreeSet
LinkedHashSet
LinkedList
Stack
<<Interface>>
Map
<<Interface>>
SortedMap
HashTable
LinkedHashMap
HashMap
TreeMap



Explain and Send Screenshots

Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

file:///tmp/t.html

The Collection Interface

A **Collection** represents a group of objects known as its elements

The interface has methods:

to tell you how many elements are in the collection (size, isEmpty),

to check whether a given object is in the collection (contains),

to add and remove an element from the collection (add, remove),

provide an iterator over the collection (iterator).

The 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

The basic operations of Map:

put, get

containsKey, containsValue

size, isEmpty

Collection Implementations

ArrayList – resizable array
LinkedList – double-linked list
HashSet – unsorted set of unique values
TreeSet - sorted set of unique values

Map Implementations

HashMap – unsorted key-value set
TreeMap – sorted key-value set