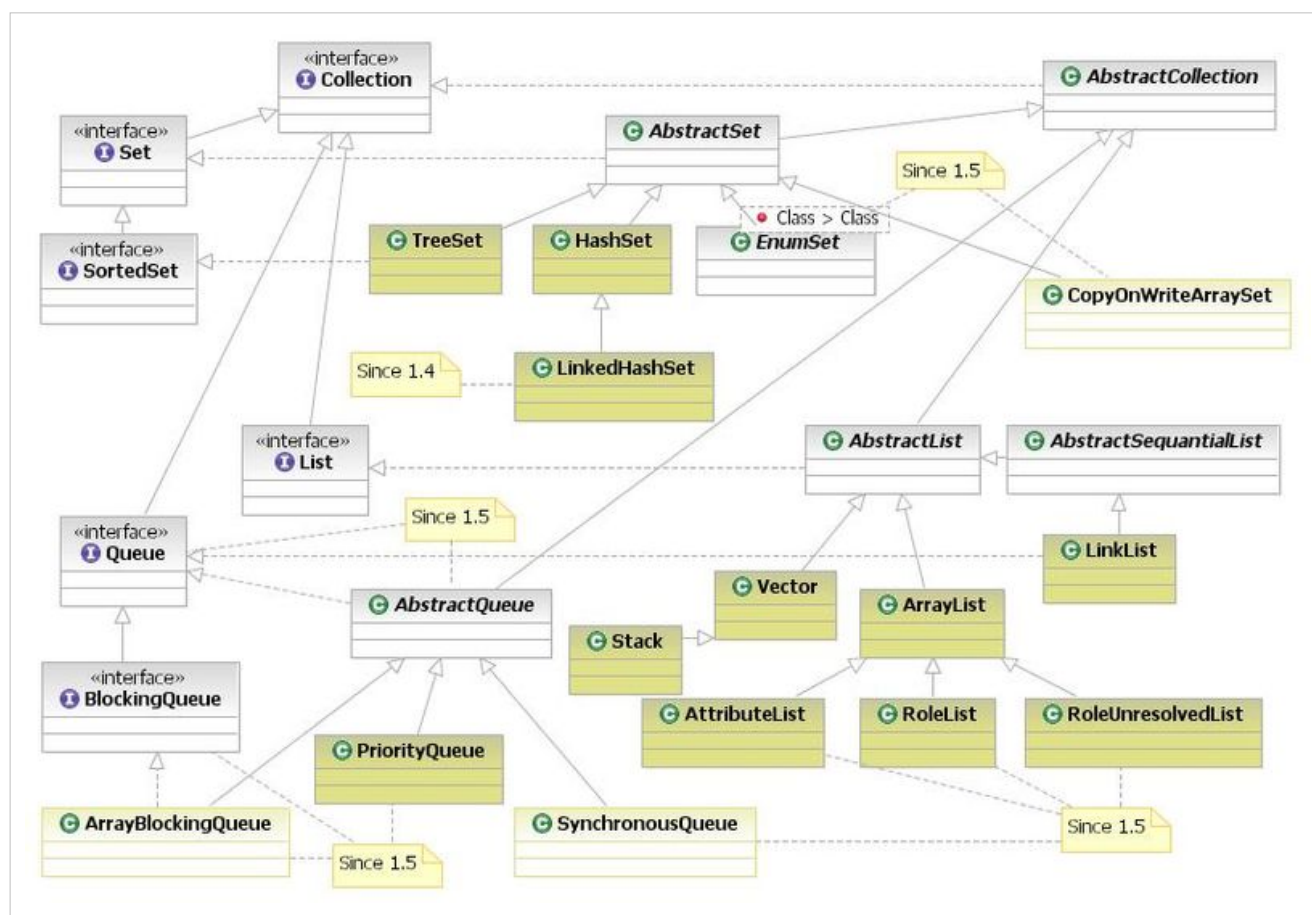


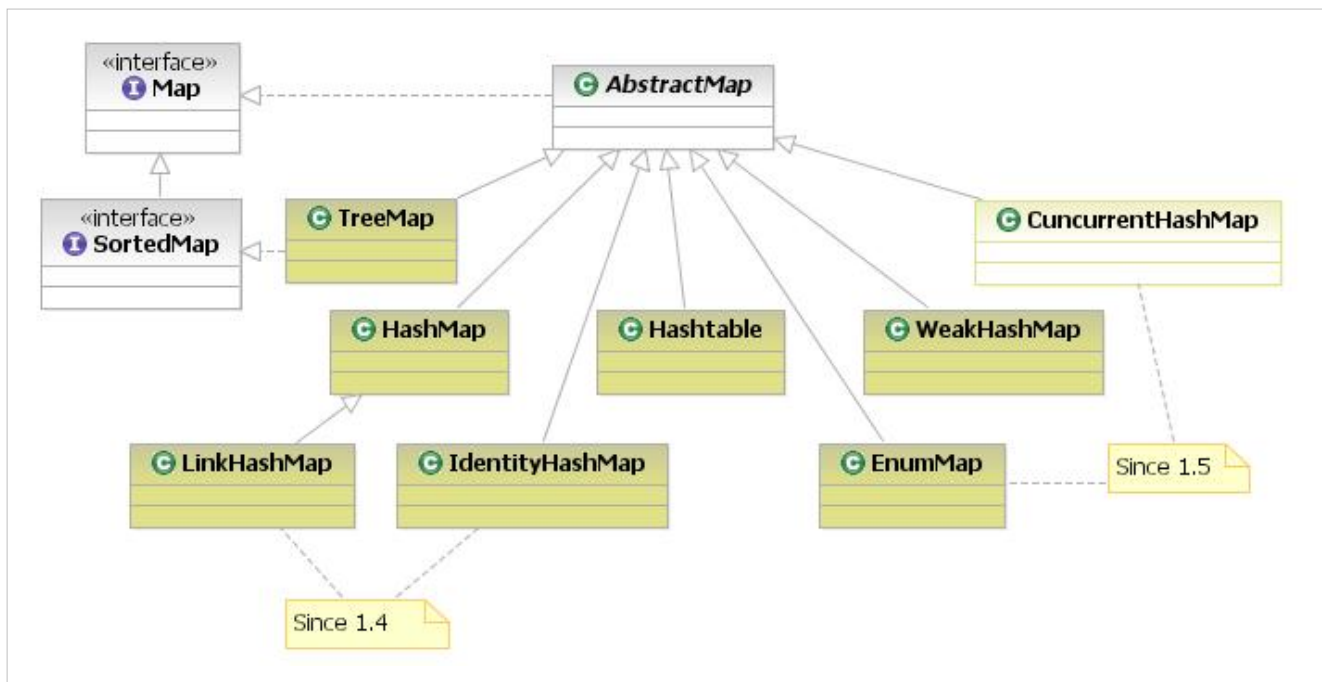
	synchronized	non-synchronized
List	java.util.Vector	java.util.ArrayList
	java.util.Stack	
		java.util.LinkedList
	java.util.concurrent.CopyOnWriteArrayList	
Set		java.util.TreeSet
		java.util.HashSet
		java.util.LinkHashSet
	java.util.concurrent.CopyOnWriteArraySet	
Map		java.util.TreeMap
	java.util.Hashtable java.util.concurrent.ConcurrentHashMap	java.util.HashMap
		java.util.LinkHashMap
		java.util.IdentityHashMap
		java.util.EnumMap

Classes Diagram (UML)

The following UML class diagram shows the **Collection** interfaces and their implementations.



The following UML class diagram shows the **Map** interfaces and their implementations.



Throwing and Catching Exceptions

Navigate **Exceptions** topic:

- Throwing and catching exceptions
 - Application exceptions
 - Runtime exceptions
 - NullPointerException
 - Main Exception classes
- Stack trace
- Checked exceptions
- Unchecked exceptions
- Preventing NullPointerException
- Chained exceptions(Nesting Exceptions)

In Java, there are two main flow of code executions.

- Normal main sequential code execution, the program doing what it meant to accomplish
- Exception handling code execution, the main program flow was interrupted by an error or some other condition that prevent the continuation of the normal main sequential code execution. After the interruption, that is when the exception is "thrown", the search for the first matching catch block/clause begins, the search is going backward against the main flow, until a match is found.

The exception handling is built in to the language. There is a keyword to **throw** an exception, there is keyword to **catch** an exception, and there is keyword(**throws**) to declare that a method could throw an exception that the caller has to deal with.

An exception can be thrown from anywhere in the code by the **throw** keyword, plus specifying a Java object, that is instance of the Throwable class. Applications should sub-class the Exception class as an exception object to be thrown.

The following is an example of throwing an instance of the `ExceptionClass`.

```
throw new ExceptionClass("Something happened, the main program flow has  
to be interrupted, the caller has to be informed about this");
```

At the above line the current method execution is interrupted and the created exception object is thrown back to the caller. The exception object should contain the information about the reason of the interruption.

The caller of the method catches the exception by the **catch** keyword.

```
catch (ExceptionClass e)  
{  
    // --- What to do with this interruption ??? ---  
}
```

Sometimes just the name of the exception class is enough to know the reason of the interruption.

Catching Matching rules

When an exception is "thrown", the search for the first matching catch block/clause begins.

There are five main catching matching rules:

1. A thrown exception object, can be caught by the **catch** keyword with specifying the exception object's class or its super-class.
2. When there are a list of catch clauses, it is evaluated sequentially, applying the first rule. If there is a catch, the following catch clauses are ignored.
3. If there is a catch clause in the list, which will never be executed, because its super-class are listed before it, the compiler will give an error message.
4. The compiler enforce that all `Exception` and its sub-class exceptions must be handled by the programmer, except the `RuntimeException` and its sub-classes.
5. If the try block would never throw an exception that is specified in the catch list, the compiler gives an error.

Rule number 1

A thrown exception object, can be caught by the **catch** keyword with specifying the exception object's class or its super-class.

The `Throwable` class is the super-class of all exception class. So specifying with the `catch` keyword we can catch all type of exceptions.

So the following piece of code would catch all type of exception.

```
<strike>  
catch (Throwable th) {  
    // --- I caught all type of exception ---  
    log_it( th );  
  
    // --- I was not suppose to catch Throwable so I throw it again ---  
    throw th;  
}  
</strike>
```

There are some special exceptions that used by the JVM, those are the sub-classes of `java.lang.Error`. We are not suppose the catch them. So we should use the following code to catch all application, and runtime exceptions.