COLECCIONES

COLECCIONES

Proveen una arquitectura para almacenamiento de grupos de objetos

- Busqueda
- Ordenamiento
- Inserción
- Manipulación
- Eliminación

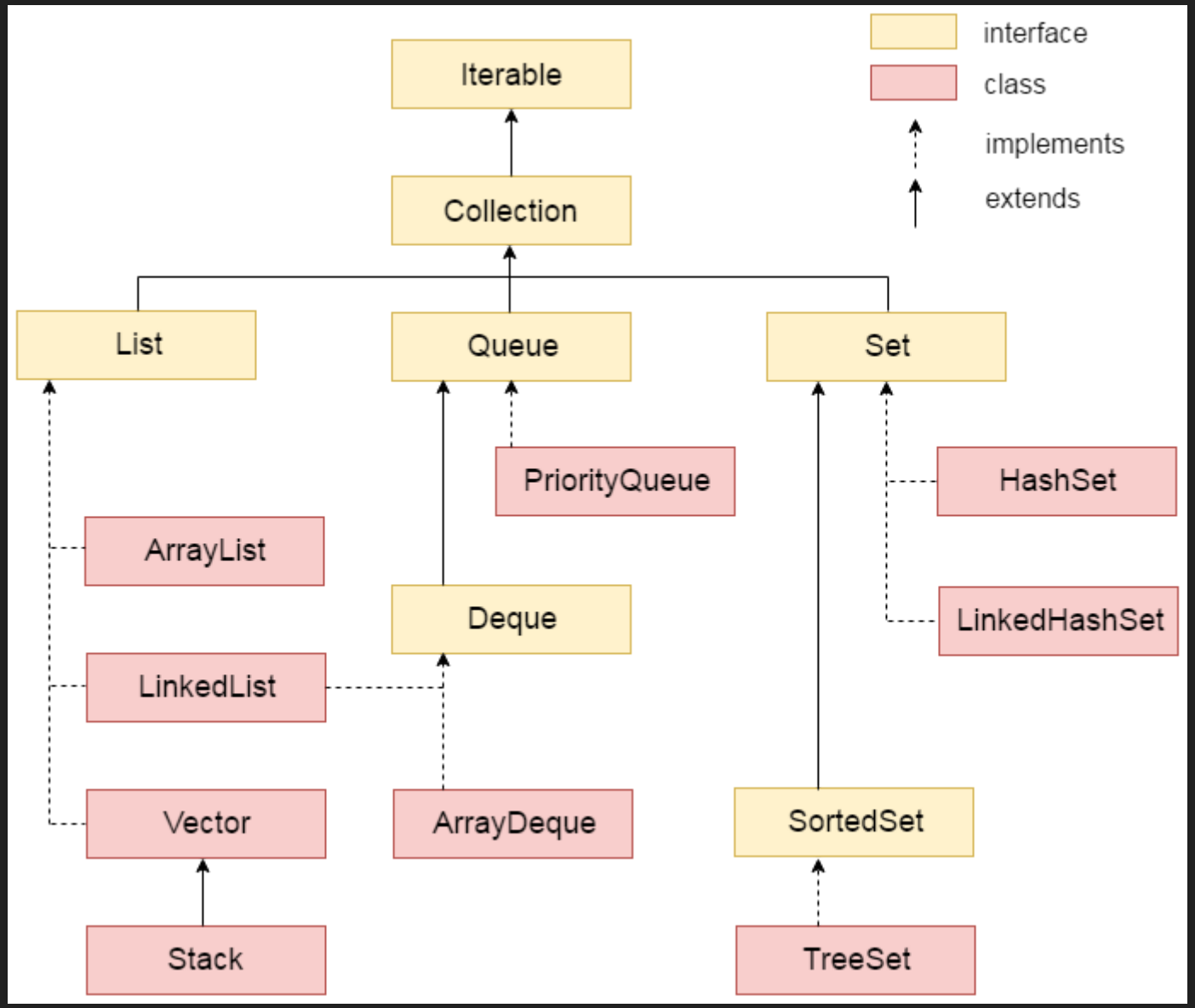
QUE ES UNA COLECCIÓN

Representa una unidad simple de objetos (grupo)

COLLECTION FRAMEWORK

Representa una arquitectura unificada para el manejo de grupos de objetos

- Interfaces
- Clases
- Algoritmos


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

INTERFAZ COLLECTION

```
public boolean add(Object element)
public boolean addAll(Collection c)
public boolean remove(Object element)
public boolean removeAll(Collection c)
public boolean retainAll(Collection c)
public int size()
public void clear()
public boolean contains(Object element)
public boolean containsAll(Collection c)
public Iterator iterator()
public Object[] toArray()
public boolean isEmpty()
public boolean equals(Object element)
public int hashCode()
```

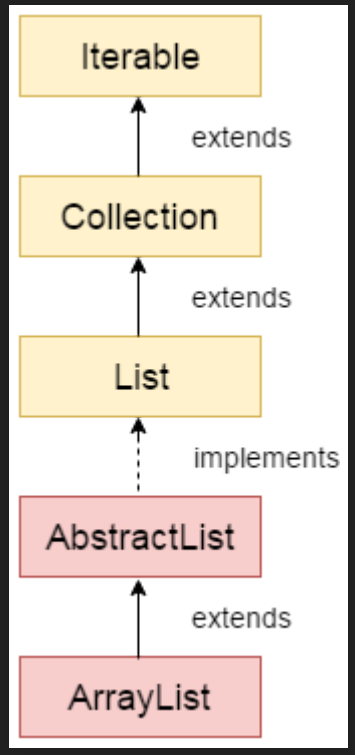
INTERFAZ ITERATOR

```
public boolean hasNext()
public Object next()
public void remove()
```

ARRAYLIST

Es una colección ordenada que utiliza arreglos dinámicos

- Puede contener elementos duplicados
- Mantiene el orden de inserción
- Utiliza un índice
- Manipulación es lenta
- Rápido acceso aleatorio
- Métodos no estan sincronizados


Iterable
extends
Collection
extends
List
implements
AbstractList
extends
ArrayList

ARRAYLIST CONSTRUCTORES

```
ArrayList(){ }

ArrayList(Collection c){ }

ArrayList(int capacity)
```

ARRAYLIST MÉTODOS

```
void add(int index, Object element)
boolean add(Object o)
boolean addAll(Collection c)
boolean addAll(int index, Collection c)

void clear()
int lastIndexOf(Object o)

Object[] toArray()
Object[] toArray(Object[] a)

Object clone()
int indexOf(Object o)
void trimToSize()
```

GENERICOS

```
// No Genericos
ArrayList myArray = new ArrayList();

// Genericos
ArrayList<String> myArray = new ArrayList<String>();

// Genericos
ArrayList<Object> myArray = new ArrayList<Object>();
```

EJEMPLO

```
import java.util.*;

class TestCollection1{

  public static void main(String args[]){

    ArrayList<String> list=new ArrayList<String>();//Creating arrayli
    list.add("Ravi");//Adding object in arraylist
    list.add("Vijay");
    list.add("Ravi");
    list.add("Ajay");

  }
}
```

ITERAR UN ARRAYLIST

```
ArrayList<String> list=new ArrayList<String>();
list.add("Ravi");
list.add("Vijay");
list.add("Ravi");
list.add("Ajay");

for(String obj: list){
  System.out.println(obj);
}

for(int i=0; i <= list.size(); i++){
  System.out.println(list.get(i));
}

Iterator iterator = list.iterator();
while(iterator.hasNext()){
  System.out.println(iterator.next());
}
```

ITERAR UN ARRAYLIST

```
ArrayList<String> al=new ArrayList<String>();
al.add("Ravi");
al.add("Vijay");
al.add("Ajay");

ArrayList<String> al2=new ArrayList<String>();
al2.add("Sonoo");
al2.add("Hanumat");

al.addAll(al2);//adding second list in first list

Iterator itr=al.iterator();

while(itr.hasNext()){
  System.out.println(itr.next());
}
```

EJERCICIOS

ArrayList: https://goo.gl/nBQmkk

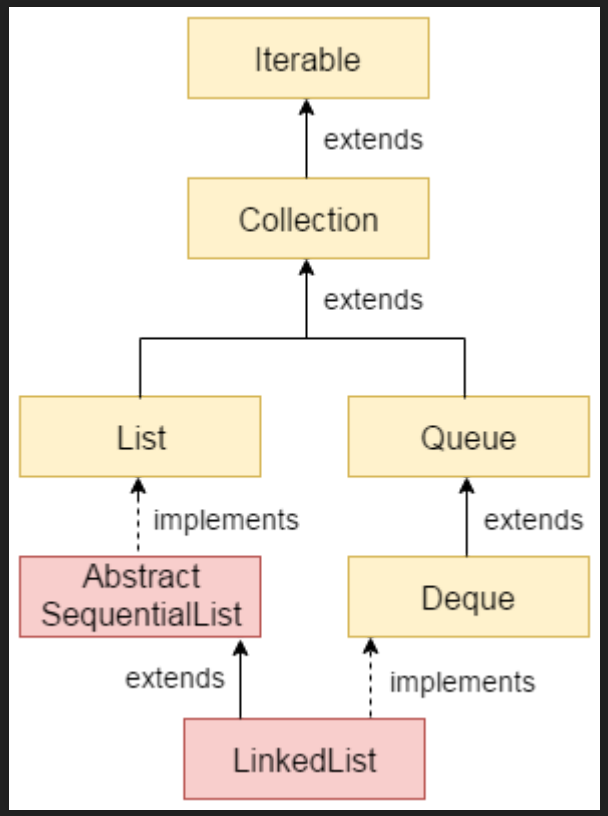
LINKEDLIST

LINKEDLIST

Son listas doblemente enlazadas para almacenar elementos

- Puede contener duplicados
- Lista ordenada
- Mantiene el orden de inserción
- Métodos no sincronizados
- La manipulación es rapida
- Pila, Cola o Lista

JERARQUÍA


Iterable
extends
Collection
extends
List
Queue
implements
extends
Abstract SequentialList
Deque
extends
implements
LinkedList

CONSTRUCTORES

```
LinkedList()

LinkedList(Collection c)
```

MÉTODOS

```
void add(int index, Object element)
void addFirst(Object o)
void addLast(Object o)
int size()
boolean add(Object o)
boolean contains(Object o)
boolean remove(Object o)
Object getFirst()
Object getLast()
int indexOf(Object o)
int lastIndexOf(Object o)
```

EJEMPLO

```
import java.util.*;

public class Test{
  public static void main(String args[]){

    List<String> list=new LinkedList<String>();
    list.add("Ravi");
    list.add("Vijay");
    list.add("Ravi");
    list.add("Ajay");

    Iterator<String> itr=al.iterator();
    while(itr.hasNext()){
      System.out.println(itr.next());
    }
  }
}
```

EJEMPLO

```
import java.util.*;

public class Test{
  public static void main(String args[]){

    LinkedList<String> list=new LinkedList<String>();
    list.add("Ravi");
    list.add("Vijay");
    list.add("Ravi");
    list.add("Ajay");

    Iterator<String> itr=al.iterator();
    while(itr.hasNext()){
      System.out.println(itr.next());
    }
  }
}
```

EJEMPLO

```
import java.util.*;

public class Test{
  public static void main(String args[]){

    Queue<String> queue=new LinkedList<String>();
    queue.add("Ravi");
    queue.add("Vijay");
    queue.add("Ravi");
    queue.add("Ajay");

  }
}
```

EJEMPLO

```
import java.util.*;

public class Test{
  public static void main(String args[]){

    Deque<String> deque=new LinkedList<String>();
    deque.add("Ravi");
    deque.add("Vijay");
    deque.add("Ravi");
    deque.add("Ajay");

  }
}
```

EJERCICIO

http://bit.ly/2FlZHxd

INTERFAZ LIST

```
void add(int index,Object element)
boolean addAll(int index,Collection c)
Object get(int index)
Object remove(int index)
ListIterator listIterator()
ListIterator listIterator(int index)
```

EJEMPLO

```
import java.util.*;

public class Test{
  public static void main(String args[]){

    List<String> list=new LinkedList<String>();
    list.add("Ravi");
    list.add("Vijay");
    list.add("Ravi");
    list.add("Ajay");

  }
}
```

INTERFAZ LISTITERATOR

```
boolean hasNext()
Object next()
boolean hasPrevious()
Object previous()
```

EJEMPLO

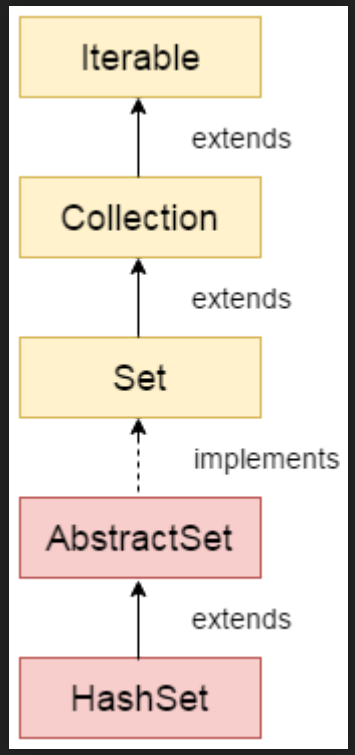
```
import java.util.*;
public class TestCollection8{
  public static void main(String args[]){
    ArrayList<String> al=new ArrayList<String>();
    al.add("Amit");
    al.add("Vijay");
    al.add("Kumar");

    ListIterator<String> itr=al.listIterator();
    while(itr.hasNext()){
      System.out.println(itr.next());
    }

    while(itr.hasPrevious()){
       System.out.println(itr.previous());
    }

  }
}
```

HASHSET

Es una colección de tipo conjunto que se utiliza para almacenar elementos

- No acepta duplicados
- No es ordenada
- Usa el equals y el hasCode para insertar
- No se usa para iterar
- Los objetos deben sobreescribir el hashCode para determinar duplicidades

HASHSET


Iterable
extends
Collection
extends
Set
implements
AbstractSet
extends
HashSet

HASHSET

```
public class MainClients{
  public static void main(String args[]){
    Set clients = new HashSet();

    clients.add( new String("Rosita Alvires"));
    clients.add( new String("Porfirio Gonzales"));
    clients.add( new String("Agustin Jaime"));
    clients.add( "Agustin Jaime" );

  }
}
```

HASHSET

```
class Client{
  String name;
  String lastName;

  Client(){ }

  Client(String name, String lastName){
    this.name = name;
    this.lastName = lastName;
  }
}
```

HASHSET

```
public class MainClients{
  public static void main(String args[]){
    Client c1 = new Client("Rosita", "Alvires");
    Client c2 = new Client("Porfirio", "Gonzales");
    Client c3 = new Client("Agustin", "Jaime");
    Client c4 = new Client("Natalio", "Reyes");
    Client c5 = new Client("Natalio", "Reyes");

    Set clients = new HashSet();
    clients.add(c1);
    clients.add(c2);
    clients.add(c3);
    clients.add(c4);
    clients.add(c5);

  }
}
```

HASHSET

```
public class MainClients{
  public static void main(String args[]){
    Client c1 = new Client("Rosita", "Alvires");
    Client c2 = new Client("Rosita", "Alvires");

    if (c1.equals(c2)){
      System.out.print("Son iguales");
    }else{
      System.out.print("Son diferentes");
    }

  }
}
```

HASH CODE

Es un número que hace referencia al objeto

HASHSET

```
class Client{
  String name;
  String lastName;

  public boolean equals(Object obj){
    if (obj instanceof Client){
      Client other = (Client)obj;

      if(this.name == other.name && this.lastName == other.lastName){
        return true;
      }else{
        return false;
      }

     }else {
       return false;
     }

   }
}
```

HASHSET

```
public class MainClients{
  public static void main(String args[]){
    Client c1 = new Client("Rosita", "Alvires");
    Client c2 = new Client("Rosita", "Alvires");

    if (c1.equals(c2)){
      System.out.print("Son iguales");
    }else{
      System.out.print("Son diferentes");
    }

  }
}
```

HASHSET

```
public class MainClients{
  public static void main(String args[]){
    Client c1 = new Client("Rosita", "Alvires");
    Client c2 = new Client("Rosita", "Alvires");

     System.out.print(c1.hashCode());
     System.out.print(c2.hashCode());

  }
}
```

HASHSET

```
public class MainClients{
  public static void main(String args[]){
    Client c1 = new Client("Rosita", "Alvires");
    Client c2 = new Client("Rosita", "Alvires");

    c1 = c2;

     System.out.print(c1.hashCode());
     System.out.print(c2.hashCode());

  }
}
```

HASHSET

```
HashSet()
HashSet(Collection c)
HashSet(int capacity)
```

HASHSET

```
void clear()
boolean contains(Object o)
boolean add(Object o)
boolean isEmpty()
boolean remove(Object o)
Object clone()
Iterator iterator()
int size()
```

EJERCICIO

Noughts and Crosses

EJERCICIO

Noughts and Crosses

http://bit.ly/2FDmEbe

https://github.com/adalarcon/NoughtsAndCrossesGame

EJERCICIO

Noughts and Crosses

http://bit.ly/2FDmEbe

https://github.com/adalarcon/NoughtsAndCrossesGame