COLECCIONES

Qué son las colecciones

Definición

Las colecciones (Collections) son el modo de agrupar objetos. También llamados contenedores, representa a un conjunto de ítems, un conjunto de objetos, que pueden ser homogéneos o no. Por ejemplo, una agenda es una colección de datos de personas.

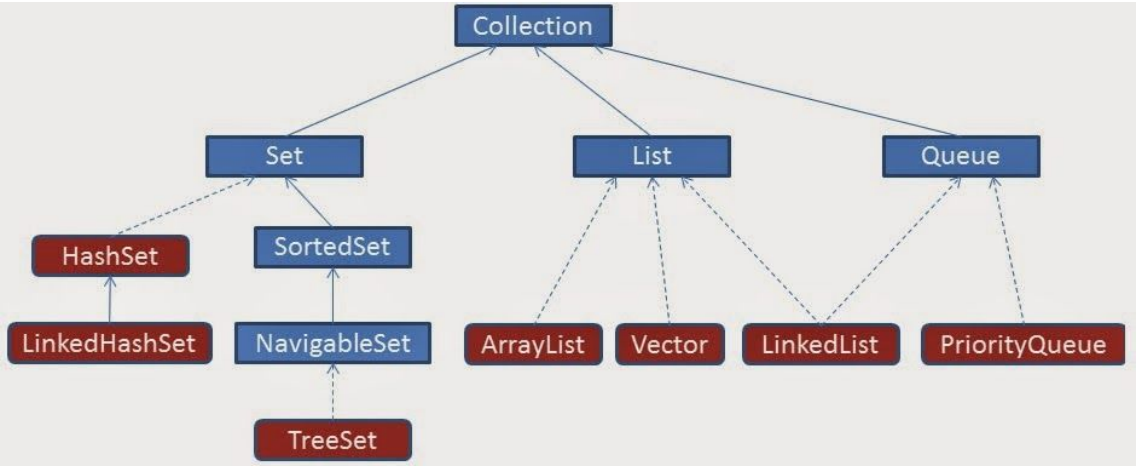
La interfaz Collection

La interfaz collection es la super interfaz de donde heredan las mayoría de las interfaces utilizadas para el manejo de las colecciones. Es la interfaz raíz de la jerarquía de interfaces.

La interfaz Collection forma parte del Collection Framework, un conjunto de interfaces y clases que representan distintos modos de agrupar objetos, según distintas políticas de manejo de memoria o acceso a ellos.

Representan un conjunto de objetos, también llamados elementos. Una clase que quiera comportarse como una Collection deberá implementar esta interfaz, por lo tanto sus métodos.

A continuación observamos un gráfico con la jerarquía de Interfaces que componen el Collection Framework, como dato adicional las interfaces correspondientes a Queue estan disponibles desde la versión 1.5 de Java.



En el gráfico se observan tres grandes grupos de interfaces que representan tres tipos de contenedores distintos en su comportamiento.

Interfaz Set

La interfaz Set, define una colección que no puede contener elementos duplicados. Esta interfaz contiene, únicamente, los métodos heredados de Collection añadiendo la restricción de que los elementos duplicados están prohibidos. Es importante destacar que, para comprobar si los elementos son elementos duplicados o no lo son, es necesario que dichos elementos tengan implementada, de forma correcta, los métodos equals y hashCode. Para comprobar si dos Set son iguales, se comprobarán si todos los elementos que los componen son iguales sin importar en el orden que ocupen dichos elementos.

HashSet

HashSet es la implementación más performante de la interfaz Set. Almacena sus elementos en una tabla Hash sin garantizar ningún tipo de orden durante la iteración.

```
[code]
public static void main(String args[]){

    HashSet<String> hashSet = new HashSet<String>();

    hashSet.add("element1");
    hashSet.add("element2");
    System.out.println(hashSet);

    hashSet.remove("element1");
    System.out.println(hashSet);
  }
[/code]
```

TreeSet

TreeSet es un poco más lenta que HashSet, almacena sus elementos en una estructura de árbol rojo-negro. TreeSet ordena sus elementos en base a sus valores.

```
[code]
public static void main(String args[]){

    TreeSet<String> treeSet = new TreeSet<String>();

    treeSet.add("element1");
    treeSet.add("element2");
    treeSet.add("element3");
    System.out.println(treeSet);

    treeSet.remove("element2");
    System.out.println(treeSet);
  }
[/code]
```

LinkedHashSet

LinkedHashSet está implementada como una HashTable con una LinkedList corriendo sobre esta. Ordena sus elementos en base al orden en el que fueron insertados en el Set.

```
[code]
public static void main(String[] args) {

    LinkedHashSet<String> linkedHashSet = new LinkedHashSet<String>();

    linkedHashSet.add("element1");
    linkedHashSet.add("element2");
```

```
        linkedHashSet.add("element3");
        System.out.println(linkedHashSet);

        linkedHashSet.remove("element1");
        System.out.println(linkedHashSet);
    }
[/code]
```

Interfaz List

La interfaz List, define una sucesión de elementos. A diferencia de la interfaz Set, la interfaz List sí admite elementos duplicados. A parte de los métodos heredados de Collection, añade métodos que permiten mejorar los siguientes puntos:

- Acceso posicional a elementos: manipula elementos en función de su posición en la lista.
- Búsqueda de elementos: busca un elemento concreto de la lista y devuelve su posición.
- Iteración sobre elementos: mejora el Iterator por defecto.
- Rango de operación: permite realizar ciertas operaciones sobre rangos de elementos dentro de la propia lista.

ArrayList

ArrayList es la implementación más performante en condiciones normales. Para definirlo de una forma simple es un array de valores u objetos expandible en tamaño.

```
 [code]
 public static void main(String args[]){

        ArrayList<String> arrayList = new ArrayList<String>();

        arrayList.add("element1");
        arrayList.add("element2");
        arrayList.add("element3");
        System.out.println(arrayList);

        arrayList.remove("element3");
        System.out.println(arrayList);
    }
[/code]
```

LinkedList

En la mayoría de los casos Linked list es un poco mas lenta que ArrayList pero puede funcionar mejor en ciertas condiciones. LinkedList está implementada con una lista doblemente linkeada mientras que ArrayList está implementada con un array que se redimensiona dinámicamente.

LinkedList permite realizar inserción y eliminaciones de forma más performante, pero el acceso a los elementos siempre es secuencial, solo se puede iterar sobre la lista para adelante y para atrás, para acceder un elemento en el medio toma tiempo proporcionalmente al tamaño de la lista.

ArrayList permite el acceso aleatorio a sus elementos, por lo que se puede acceder rápidamente a cualquier elemento a un tiempo constante, pero las inserciones y eliminaciones de cualquier lado (excepto al final) requiere mover todos los elementos, ya sea para hacer un hueco para el nuevo elemento o llenar

un espacio dejado por uno que se eliminó. Si se añaden más elementos que la capacidad del arreglo, uno nuevo con el doble de tamaño es creado, y el arreglo anterior es copiado al nuevo, con todo el tiempo de cómputo que eso significa en arreglos grandes.

[code]
public static void main(String args[]){
    LinkedList<String> linkedList = new LinkedList<String>();

    linkedList.add("element1");
    linkedList.add("element2");
    linkedList.add("element3");
    System.out.println(linkedList);

    linkedList.remove("element3");
    System.out.println(linkedList);
  }
[/code]

GENERICS

Generics es un concepto de “programación genérica” que fue introducido oficialmente a Java en 2004 dentro del lanzamiento de la versión J2SE 5.0.
Este concepto permite que un tipo o método operen con objetos de varios tipos mientras se provee una seguridad en tiempo de compilación. Esta funcionalidad especifica el tipo de objetos almacenados en una colección Java.
Generics suma estabilidad al código haciendo más detectables errores de programación en tiempo de compilación.
El siguiente bloque de código ilustra el problema que se puede dar al no utilizar generics. Primero, se declara un objeto del tipo ArrayList. Luego, se inserta un String al ArrayList. Finalmente, se intenta obtener el String insertado y castearlo a un Integer.

[code]
        List v = new ArrayList();
        v.add("test");
        Integer i = (Integer)v.get(0); // Run time error
[/code]

Aunque el código es compilado sin error, este tira una runtime exception (java.lang.ClassCastException) al ejecutar la tercer línea de código. Este tipo de problemas pueden ser evitados utilizando generics, esta es una de las principales motivaciones para utilizar generics.
Utilizando generics, el código anterior puede ser reescrito de la siguiente forma:

[code]
        List<String> v = new ArrayList<>();
        v.add("test");

```
        Integer i = v.get(0); // (type error)  compilation-time error
[/code]
```

El parámetro de tipo String dentro de los brackets mayor y menor declara que el ArrayList será constituido por objetos String. Con generics, ya no es necesario castear la tercer línea a un tipo particular, porque el resultado de **v.get(0)** es definido como String por el código generado por el compilador.
Compilando la tercer línea de este fragmento con J2SE 5.0 (o posterior) arrojara un error en tiempo de compilación porque el compilador detectará que **v.get(0)** devuelve un String en lugar de un Integer.