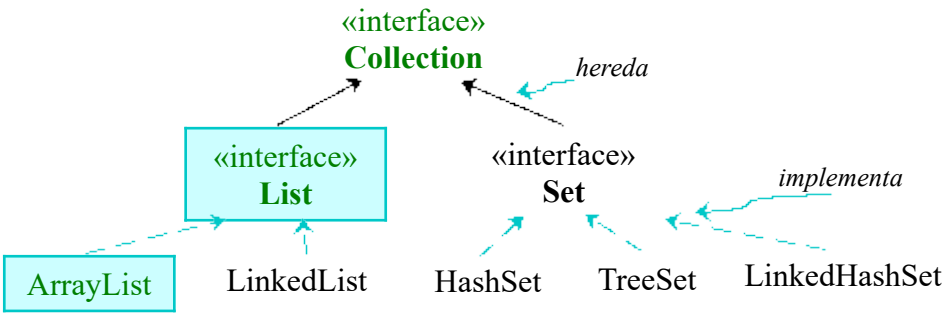
Capítulo 12

Collections

A menudo necesitamos guardar información, pero no sabemos de antemano el espacio que va a ocupar en la memoria. En estos casos, las tablas no son la solución adecuada, ya que su tamaño debe permanecer fijo una vez declaradas. En su lugar, necesitaremos **estructuras dinámicas de datos**, es decir, objetos que alberguen datos que se crean y se destruyen en tiempo de ejecución, según las necesidades de la aplicación. Para este fin, Java nos proporciona una serie de estructuras dinámicas que comparten un conjunto de métodos declarados en la «interfaz Collection». Todas ellas implementan dicha interfaz, aunque de distinta forma.



Existen tres tipos fundamentales de estructuras ligadas a la interfaz **Collection**

- Listas: responden a la necesidad de manejar sucesiones de datos que pueden estar repetidos y cuyo orden puede ser importante.
- Conjuntos: el orden de los datos no es relevante y lo que realmente importa es la mera pertenencia, o no, de un dato a la estructura, con lo que las repeticiones tampoco tienen sentido.
- **Mapas** o **Diccionarios**: relacionados con la interfaz Collection, aunque no la implementan. Sirven parar guardar datos identificados por claves que no se repiten.

Una particularidad de Collection es que trabaja con tipos genéricos de datos. Esto quiere decir que cuando declaremos una de estas estructuras, especificaremos la clase de objetos que se pueden insertar en ellas, lo que permite hacer un control de tipos más eficiente.

El conjunto de interfaces y clases del marco de trabajo Collection se halla en el paquete java.util.

LISTAS

Existen dos implementaciones de listas, las clases ArrayList y LinkedList. Las dos proporcionan los mismos métodos y funcionalidades, ya que implementan la interfaz List, que hereda de Collection.
La sintaxis general para construir un ArrayList con un tipo genérico de datos[1], es:

```
ArrayList <E> nomlis = new ArrayList<>();
```

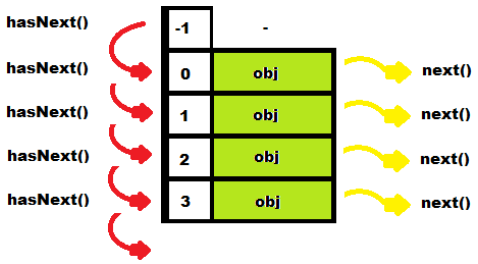
En esta lista solo se podrán insertar objetos (nodos) del tipo E. Si E es Cliente, solo se podrán insertar objetos Cliente o de una subclase de Cliente. E debe ser siempre el nombre de una clase, NUNCA de un tipo primitivo.

Métodos básicos de la interfaz Collection

- add(): Sirve para insertar un elemento al final de la lista. Si la inserción tiene éxito, devuelve true, en caso contrario, false.
- remove(): Elimina la primera ocurrencia del elemento buscado en una lista. Devuelve true si la eliminación ha tenido éxito.
- clear(): Nos permite eliminar todos los nodos de una lista y dejarla vacía.
- size(): Nos permite saber en cada momento el número de elementos (o nodos) insertados en una lista.
- isEmpty(): Permite saber si una lista está vacía, devolviendo true.
- contains(): Nos dice si un elemento determinado está en una lista, devolviendo true.
- toString(): Devuelve una cadena que representa la colección.

Iteradores

A menudo necesitamos recorrer una lista nodo a nodo. Una de las formas de hacerlo es por medio de iteradores, que son objetos que van apuntando sucesivamente a los nodos de la lista, empezando por el primero. Las funciones hasNext() y next(), que se complementan y emplean conjuntamente para recorrer la lista, ya que inicialmente el iterador apunta al principio de la lista, justo antes del primer elemento.



- hasNext(): comprueba si quedan elementos por visitar y nos devuelve true o false, según el caso.
- next(): Nos devuelve una copia del siguiente «objeto», si existe, y avanza una posición en la lista, apuntando al nodo siguiente. En caso de que no haya siguiente, porque estemos al final de la lista o porque esta estuviera vacía, next() lanzará la excepción NoSuchElementException.

En la práctica, para evitar la excepción, los dos métodos se usan conjuntamente, de forma que solo se llama al método next() si antes se ha comprobado que hay elemento siguiente, con hasNext(). «Los iteradores tienen un tercer método que permite eliminar nodos de una lista».

- remove(): Elimina de la lista, en cada momento, el último elemento devuelto por next(). No confundir con el remove de la interfaz colecction, que vimos antes.

---

[1] Introducción a los tipos genéricos (Anexo I)

1er Código Fuente ArrayList

```
package arraydinamico;
import java.util.ArrayList;
import java.util.Iterator;
public class ArrayDinamico {
    private String direcc;
    private String nombre;
    public ArrayDinamico(String d, String n){
        this.direcc=d;
        this.nombre=n;
    }
    public String toString(){
        return "Dirección: " + this.direcc + " Nombre: "+this.nombre;
    }
    public static void main(String[] args) {
        ArrayList <ArrayDinamico> lista= new ArrayList<>();
        lista.add(new ArrayDinamico("Peru 435","Daniel"));
        lista.add(new ArrayDinamico("Guatemala 343","Maia"));
        Iterator j=lista.iterator();
        while(j.hasNext()){
            System.out.println(j.next());
        }
    }
}
```

Salida por consola:

```
run:
Dirección: Peru 435 Nombre: Daniel
Dirección: Guatemala 343 Nombre: Maia
BUILD SUCCESSFUL (total time: 0 seconds)
```

2do Código Fuente ArrayList

```
package arraydinamico;
import java.util.ArrayList;
import java.util.Iterator;
public class ArrayDinamico {
    public static void main(String[] args) {
        ArrayList <String> tablis = new ArrayList<>();
        tablis.add("René");
        tablis.add("Augusto");
        System.out.println(tablis);
        tablis.remove(1);
        System.out.println(tablis.size());
        System.out.println(tablis.get(0)); //devuelve el 1er elemento
        //Metodo Iterator, util para recorrer ArrayList
        Iterator <String> i= tablis.iterator();
        String aux;
        while(i.hasNext()){
            aux=i.next();
            System.out.println(aux);
        }
        //Método indexOf, me indica la posición de un elemento
        System.out.println(tablis.indexOf("René"));
        //Eliminar toda el ArrayList
        tablis.clear();
        //Método isEmpty, indica si un arraylist esta vacio o no
        System.out.println(tablis.isEmpty());
    }
}
```

Salida por consola:

```
run:
[René, Augusto]
1
René
René
0
true
BUILD SUCCESSFUL (total time: 2 seconds)
```

Introducción a los tipos genéricos - ANEXO I

CampusMVP, 11/08/2020

Además del polimorfismo, una característica que permite a la plataforma Java tratar homogéneamente objetos heterogéneos, de los que habitualmente no se conoce su tipo concreto, el lenguaje Java cuenta con otro mecanismo con el mismo fin: los tipos genéricos.

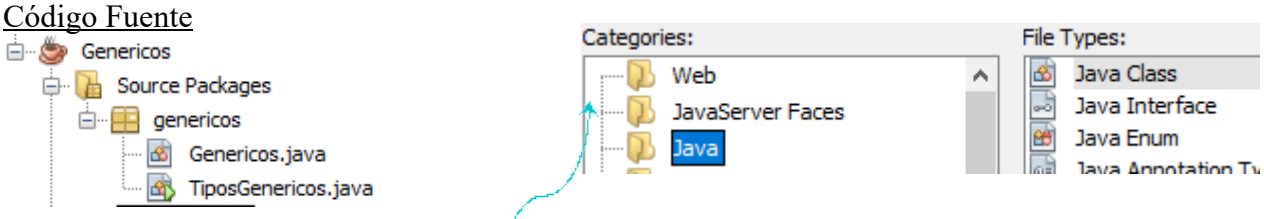
El objetivo principal de los tipos genéricos es evitar que tengamos que utilizar la clase Object[2] como tipo para ciertos atributos, parámetros o valores de retorno, como se hacía tradicionalmente en las primeras versiones de Java.

En su esencia, el término genéricos significa **tipos parametrizados**; definir una clase, una interfaz o un método cuyos tipos de datos son parametrizables.

Ejemplos:

- Se puede definir un Quicksort (algoritmo de ordenación) para una amplia variedad de tipos de datos sin ningún esfuerzo adicional.
- En lugar de definir una clase «Pila» para trabajar con números enteros, otro para trabajar con números en punto flotante y así sucesivamente, es mucho más sencillo definir una clase que sea capa de tratar con datos de diferentes tipos.

Código Fuente



Lo primero que aborde fue crear un Proyecto denominado Genericos, y dentro del paquete genericos, cree un nuevo archivo Java Class, denominado Genericos.java, en el cual especifique el siguiente código:

```
package genericos;
public class Genericos<T extends Number> {
    private T minX, maxX, minY, maxY;
    public Genericos(T miX, T miY, T maX, T maY){
        this.minX=miX;
        this.maxX=maX;
        this.minY=miY;
        this.maxY=maY;
    }
    @Override
    public String toString(){
       return this.getClass().getSimpleName() + ": " + minX + ", " + minY + ", " + maxX + ", " + maxY;
    }
}
```

Clase Genérica

Como se puede observar delante de la class Genericos, agregue el parámetro <T> que es el que va a actuar como tipo de dato parametrizable. En lugar de T podriamos asignarle cualquier otro nombre (otro identificador), pero es habitual utilizar esta notación de **un solo carácter y mayúscula**. Ahora, ya podemos utilizar este tipo parametrizable T, definiendo los atributos con los que va a contar.

[2] Object es la superclase de todas las demás clases, una referencia de Object puede referirse a cualquier tipo de objeto.

Además se le especifica <T extends Number>, donde T ha de ser necesariamente un subtipo de la clase Number, que es la clase que actúa como superclase de todos los tipos numéricos por referencia de Java, incluyendo Integer, Long y Double.

Observe que el tipo de estos cuatro atributos, de estas variables, es T. No conocemos en este momento cuál va a ser el tipo concreto. De igual forma, al definir el constructor de esta clase Genericos, definimos la lista de parámetros utilizando este mismo tipo y guardamos los datos recibidos en los correspondientes atributos. Análogamente podemos implementar el habitual método toString()[3] para mostrar los cuatros atributos por la consola.

En la clase principal TiposGenericos.java, el que contiene la función principal main(), vamos a escribir el siguiente código:

```
package genericos;
public class TiposGenericos {
    public static void main(String[] args) {
        Genericos<Integer> planoEntero= new Genericos<>(0,0,1920,1280);
        Genericos<Double> planoDouble=new Genericos<>(-1.0, -1.0, 1.0, 1.0);
        System.out.println(planoEntero);
        System.out.println(planoDouble);
    }
}
```

Vamos a comenzar instanciando un objeto planoEntero de un tipo de clase genérico, donde se le especifica el tipo concreto entre los símbolos < y >, seguidamente especificamos los parámetros. Análogamente podríamos definir un segundo objeto cuyas atributos sean de tipo numérico en punto flotante: Genericos<Double>.

En las siguientes líneas se muestra por consola los objetos: planoEntero y planoDouble.

```
run:
Genericos: 0, 0, 1920, 1280
Genericos: -1.0, -1.0, 1.0, 1.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

RESUMEN

Lo que nos permiten los tipos genéricos es facilitar una sola definición de una funcionalidad que posteriormente se aplicaría a diferentes tipos concretos, lo cual en definitiva nos ahorra mucho trabajo de codificación.

[3] Método toString(), se utiliza para convertir a String cualquier objeto Java. Ver Anexo II

Método toString() - ANEXO II

Miguel Callejón B., 22/03/2019

¿Qué es el método toString() en Java?

Lo primero que tenemos que saber es que el método **toString()** es un **método de la clase Object**. Como todos los objetos en Java heredan de dicha clase, todos ellos tienen acceso a este método.

¿Cómo implementar tu propio método toString() en Java?

Sobreescribir el método toString() es sencillo, solo tienes que tener en cuenta que este método es, en primer lugar, público; que devuelve un String y que no acepta ningún parámetro.

```
@Override
public String toString(){
    return this.getClass().getSimpleName() + ": " + minX + ", " + minY + ", " + maxX + ", " + maxY;
}
```

Fig: En el código anterior se sobreescribio el método toString()

Ahora nos explayamos con otro ejemplo. Crearemos una clase robot que contendrá los siguientes valores:

- Un id, es decir, un número entre 0 y 49 que se generará al azar de manera automática
- Una fecha de fabricación, que tomará la fecha de hoy.
- Un nombre, que asignaremos nosotros.

El código de la clase quedaría de la siguiente manera:

```
package metostring;
import java.util.Random;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
public class MetoStringRobot {
    private int id;
    private String fecha;
    private String nombre;
    public MetoStringRobot(String nom){
        SetId();
        SetFecha();
        this.nombre=nom;
    }
    private void SetId(){
        Random nal= new Random();
        id=nal.nextInt(8);
    }
    public int getId(){return id;}
    private void SetFecha(){
        LocalDate fcomp= LocalDate.now();
        DateTimeFormatter formfecha= DateTimeFormatter.ofPattern("dd 'de' LLLL 'de' yyyy");
        fecha=fcomp.format(formfecha);
    }
```

```
    public String getFecha(){return fecha;}
    public static void main(String[] args) {
        MetoStringRobot robot=new MetoStringRobot("Savio");
        System.out.println(robot);
    }
   @Override
   public String toString(){
      return "Robot n° " + id + "; " + "creado el " + fecha + ", con nombre "+ nombre +".";
   }
}
```

La salida por la consola del método toString() sobreescrito sería:

```
run:
Robot n° 1; creado el 30 de 12 de 2020, con nombre Savio.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Conclusión

El método toString() en Java no tiene mayor misterio. Como ejercicio, puedes sobreescribirlos en los objetos que crees, de manera que te familiarices con él.

Bibliografia:

https://www.campusmvp.es/recursos/post/java-introduccion-a-los-tipos-genericos.aspx
https://javadesdecero.es/avanzado/genericos-ejemplos-java/
https://www.discoduroderoer.es/metodos-y-funciones-arraylist-de-java/
https://javautodidacta.es/metodo-tostring-java/
https://www.dokry.com/do/java
https://estebanfuentealba.wordpress.com/category/java/page/7/

A ver: 1era Olimpiada en Java
https://estebanfuentealba.wordpress.com/2008/08/09/certamen-i-introduccion-a-la-programacion/
A ver: Reto conseguido: un navegador web en 5 días ¿Qué he aprendido?
https://javautodidacta.es/reto-navegador-web/

Ejercicios de Collections

1. Crear una colección de 20 números enteros aleatorios menores que 20, y guardarlos en el orden en que se vayan generando; mostrar por pantalla dicha lista una vez creada en orden creciente y decreciente.

```
package ejercollections;
import java.util.ArrayList;
import java.lang.Math;
import java.util.Collections;
public class Ecoll {
    public static void main(String[] args) {
        ArrayList <Integer> aux= new ArrayList<>();
        Integer temp;
        for(int i=0; i<20; i++){
            temp=(int)(Math.random() *15);
            aux.add(temp);
        }
        System.out.println("Listado: " + aux);
        System.out.println("Listado Ordenado: ");
        Collections.sort(aux);
        System.out.println(aux);
        System.out.println("Listado Decreciente: " );
        Collections.reverse(aux);
        System.out.println(aux);
    }
}
```

Salida por Consola:

```
Listado Ordenado:
[0, 2, 2, 3, 3, 4, 4, 6, 7, 7, 7, 8, 11, 11, 12, 12, 12, 12, 13, 13]
Listado Decreciente:
[13, 13, 12, 12, 12, 12, 11, 11, 8, 7, 7, 7, 6, 4, 4, 3, 3, 2, 2, 0]
BUILD SUCCESSFUL (total time: 0 seconds)
```