Carrera de Java Programmer SE8

•••

Módulo 2 : Lenguaje de Programación Java

Contenido

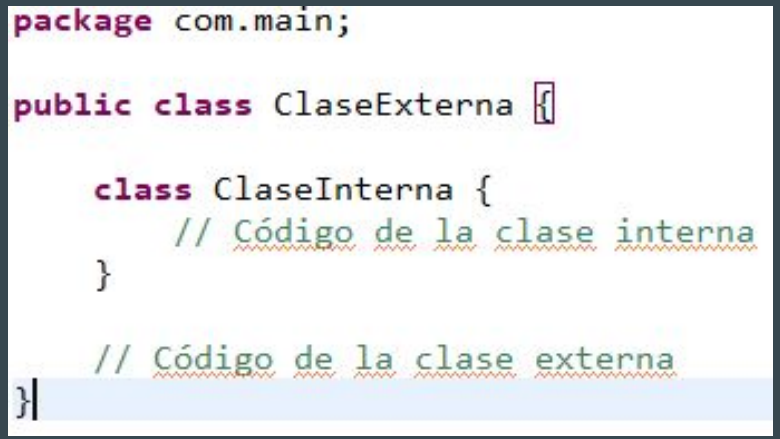
- Clases anidadas.
- Date / Time API.
- Programación funcional.
- Expresiones Lambda / Operaciones Lambda.
- Colecciones (Collections).
- Genéricos (Generics).
- Colecciones Streams y filtros.

Clases Anidadas

Tal y como lo indica su nombre, es una clase definida dentro de otra. También se les conoce como clases internas.

Se usan para:

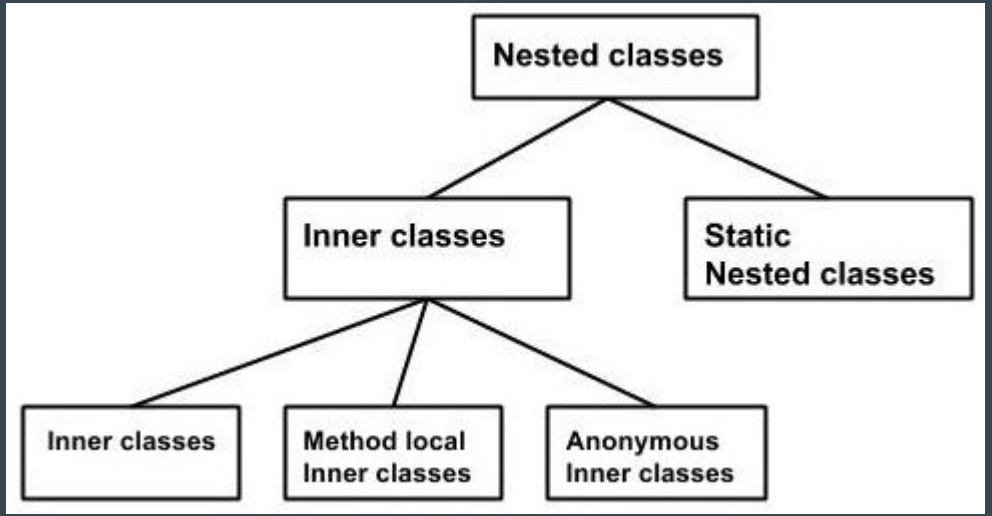
- Acceder a campos privados desde otra clase.
- Ocultar una clase de otras dentro de un mismo paquete.
- Crear clases internas anónimas.
- Gestionar eventos y retrollamadas.
- Acceder a los atributos ejemplares de otra clase.



Clases Anidadas

Existen dos tipos de clases anidadas:

- Estáticas internas: se refiere a los miembros estáticos de una clase.
- No-Estáticas internas: se refiere a los miembros no estáticos de una clase. A su vez, se dividen en:
  - Internas de miembro.
  - Internas de método local.
  - Internas anónimas.



Clases Anidadas

Estáticas internas:

- Pueden acceder a métodos y atributos de la clase "externa", siempre que estos sean estáticos (incluidos privados).
- Pueden acceder a métodos y atributos de la clase "externa", siempre que estos sean estáticos (incluidos privados).

```java
package com.classes;

public class External {// Clase externa

    public static class Internal {

        public void method() { // Clase interna
            System.out.println("Hola desde la clase interna.");
        }

    }

}
```

- En caso de que los atributos o métodos no sean estáticos, se debe instanciar la clase externa.
- Para crear una instancia de la clase estática interna:
    - ClaseExterna.ClaseEstaticaInterna nombre = new ClaseExterna.ClaseEstaticaInterna();

Clases Anidadas

Internas de miembro:

- Es un elemento más de la clase externa.
- Puede acceder a elementos públicos y privados de la clase externa.
- Accede a elementos de la clase externa luego de instanciarla.
- No podemos definir variables ni métodos estáticos (salvo constantes).
- Para instanciar la clase Interna:

ClaseExterna ce = new ClaseExterna();
ClaseExterna.ClaseInterna ci = ce.ClaseInterna();

```
package com.classes;

public class External {// Clase externa
    public int attr1;

    private class Internal1 {
        public Internal1() {
        }
    }

    public class Internal2 {
        public Internal2() {
        }
    }

}
```

Clases Anidadas

Internas de método:

- La clase solo podrá ser instanciada dentro del mismo método.
- Una clase interna de método puede acceder a las variables locales del método.
- Los modificadores permitidos por este tipo de clase son "abstract" y "final".

```java
package com.classes;

public class External {// Clase externa

    public void myMethod() {
        int number = 10;

        class MethodInnerClass {
            public void showNumber() {
                System.out.println(number);
            }
        }

        // Instanciamos la clase interna
        MethodInnerClass mic = new MethodInnerClass();
        mic.showNumber();
    }

}
```

Clases Anidadas

Internas anónimas:

- Siempre debe ser una subclase de otra clase ya existente o bien, implementar alguna interfaz.
- La definición de la clase anónima se lleva en una línea de código, por lo tanto se debe añadir punto y coma ";" al final de la declaración.
- Solamente podrá acceder a los métodos de la clase que se hayan heredado, sobrescrito o implementado.
- Es posible encontrarse una clase anónima como argumento de un método.

```
// Clase anónima
private interface AnonInnerClass {
    void anonMethodClass();
}

// Método para mostrar método de clase anónima
public void innerAnonClassMethod() {
    AnonInnerClass aic = new AnonInnerClass() {// Clase anónima

        @Override
        public void anonMethodClass() {
            System.out.println("Esto es el texto de un método de una clase anónima.");
        }

    };// Fin clase anónima

    aic.anonMethodClass();
}
```

Ejemplo

Date / Time API

Información:

- Es inmutable y no tiene métodos setter.
- Mejor distribución de los métodos para operaciones con fechas, evitando la uniformidad aburrida.
- Simplificación de manejo de la zona horaria.
- Es la actualización del paquete **java.util.Calendar**

Clases:

- java.time.LocalDate
- java.time.LocalTime
- java.time.LocalDateTime
- java.time.Period
- java.time.format.DateTimeFormatter

Métodos:

- now
- isBefore
- isAfter
- parse

Ejemplo

Programación Funcional

Funcional:

- Nuevo paradigma, aunque no significa lo mismo que la programación funcional antigua.
- Se enfoca en lo que se desea hacer.
- Se escribe poco ya que es un paradigma actual que sigue evolucionando.

Imperativa:

- Es la manera tradicional de programar.
- Se enfoca en el cómo se van a hacer las cosas.
- Conlleva escribir más líneas de código.

Expresiones Lambda

Generales:

- Incluidas en la versión 8.
- Usa la filosofía de programación funcional.
- No requiere ser parte de una instancia (objeto), pero puede serlo.
- Ayuda a compactar el código.
- Es considerada una de las optimizaciones más importantes de todos los tiempos.

Características:

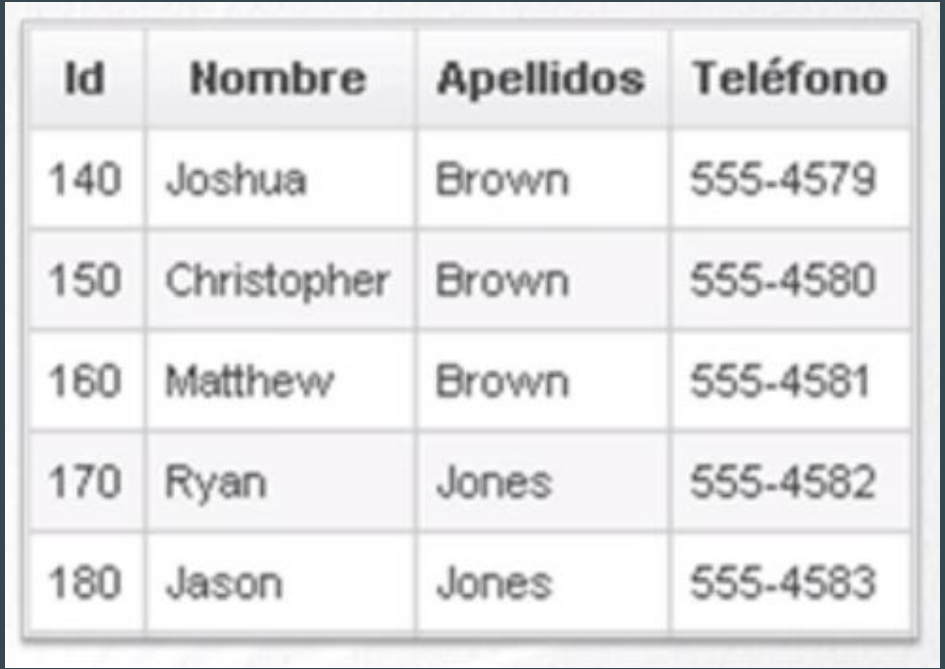
- Es un método anónimo.
- La declaración del tipo de dato es opcional.
- Envolver los parámetros entre paréntesis es opcional, sin embargo, para múltiples parámetros si hace falta.
- El uso de llaves es opcional, siempre y cuando el cuerpo de la expresión contenga un elemento simple.
- Palabra reservada ***return*** es opcional.

Ejemplo

Colecciones (Collections)

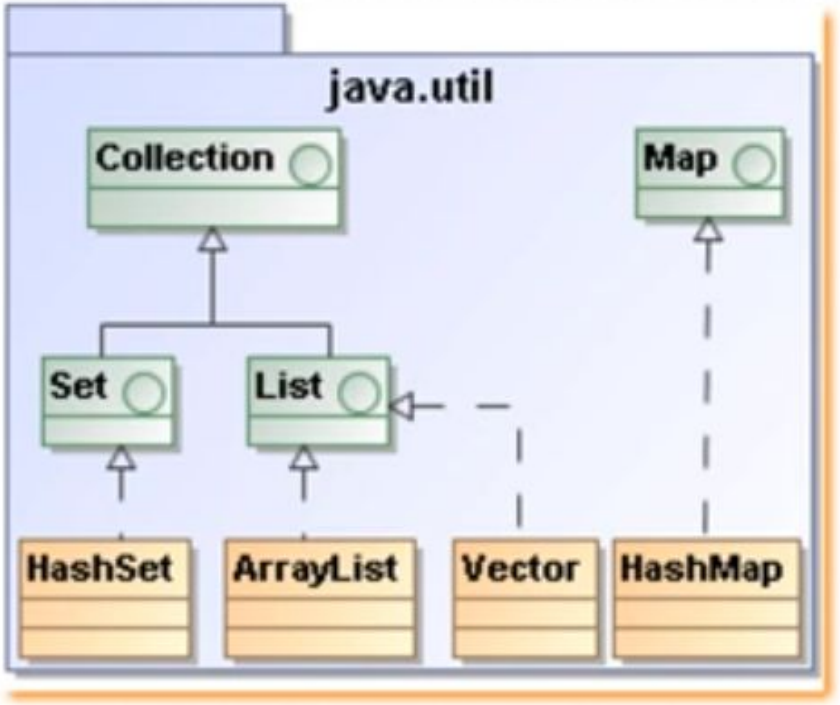
Características:

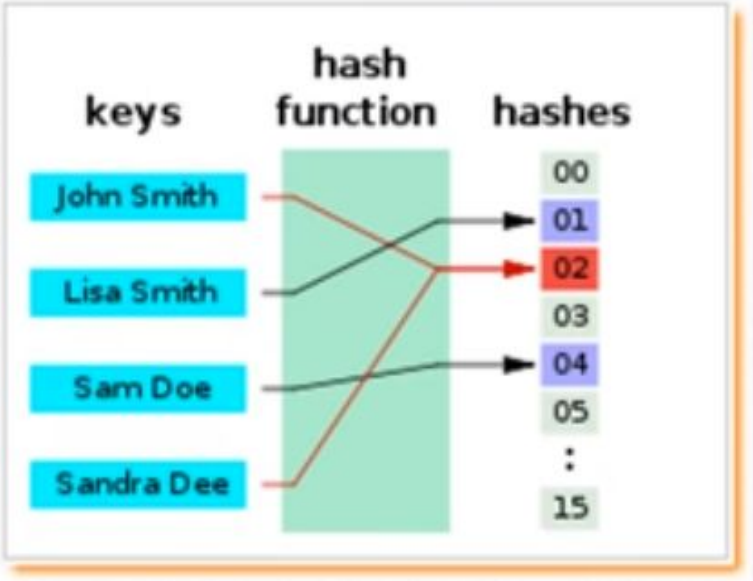
- Es un conjunto de datos.
- Almacena información estructurada.
- Poseen algunos métodos asociados para ayudar a trabajar con ellas.
- Se dividen en:
  - Sets: conjuntos de datos.
  - Lists: listas .
  - Maps: claves relacionadas con datos.
- Se encuentran en el paquete:
  - java.util.*



Id	Nombre	Apellidos	Teléfono
140	Joshua	Brown	555-4579
150	Christopher	Brown	555-4580
160	Matthew	Brown	555-4581
170	Ryan	Jones	555-4582
180	Jason	Jones	555-4583

Colecciones (Collections)





Colecciones (Collections)

```java
package manejocolecciones;

import java.util.*;

public class ManejoColecciones {

    public static void main(String args[]) {
        List miLista = new ArrayList();
        miLista.add("1");
        miLista.add("2");
        miLista.add("3");
        miLista.add("4");
        //Elemento repetido
        miLista.add("4");
        imprimir(miLista);

        Set miSet = new HashSet();
        miSet.add("100");
        miSet.add("200");
        miSet.add("300");
        //No permite elementos repetidos, lo ignora
        miSet.add("300");
        imprimir(miSet);

        Map miMapa = new HashMap();
        //Lave, valor
        miMapa.put("1", "Juan");
        miMapa.put("2", "Carlos");
        miMapa.put("3", "Rosario");
        miMapa.put("4", "Esperanza");
        //Se imprimen todas las llaves
        imprimir(miMapa.keySet());
        //Se imprimen todos los valores
        imprimir(miMapa.values());

    }

    private static void imprimir(Collection coleccion) {
        for (Object elemento : coleccion) {
            System.out.print(elemento + " ");
        }
        System.out.println("");
    }
}
```

Ejemplo

Genéricos (Generics)

Características:

- Se incluyeron en la versión 1.5
- Ya no es necesario conocer el tipo de dato
- Puede ser usado en:
  - Clase
  - Método
  - Objetos
  - Atributos

Definición de una clase genérica:

```
//Definimos una clase generica con el operador diamante <>
public class ClaseGenerica<T> {

    //Definimos una variable de tipo generico
    T objeto;

    //Constructor que inicializa el tipo a utilizar
    public ClaseGenerica(T objeto) {
        this.objeto = objeto;
    }

    public void obtenerTipo() {
        System.out.println("El tipo T es: " + objeto.getClass().getName());
    }
}
```

Uso de un tipo o clase genérico:

```
    public static void main(String[] args) {
        // Creamos una instancia de ClaseGenerica para Integer.
        ClaseGenerica<Integer> objetoInt = new ClaseGenerica<Integer>(15);
        objetoInt.obtenerTipo();
    }
```

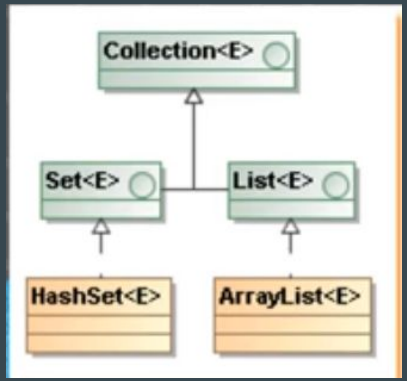
Genéricos (Generics)

```java
public class SinGenerics {

  public static void main(String args[]) {
    List lista = new ArrayList();
    lista.add(new Integer(100));
    Integer i = (Integer) lista.get(0);
  }
}
```

```java
public class ConGenerics {

  public static void main(String args[]) {
    List<Integer> lista = new ArrayList<Integer>();
    lista.add(new Integer(100));
    Integer i = lista.get(0);
  }
}
```



Genéricos (Generics)

Tipo Genérico	Significado	Descripción
E	Element	Utilizado generalmente por el framework de Colecciones de Java.
K	Key	Llave, utilizado en mapas.
N	Number	Utilizado para números.
T	Type	Representa un tipo, es decir, una clase.
V	Value	Representa un valor, también se usa en mapas.
S, U, V, etc	-	Usado para representar tipos.

Ejemplo

Streams

Características:

- Es un helper para nuestras colecciones.

Métodos:

- stream().sorted()
- stream().filter()
- stream().map()
- stream().limit()
- stream().count()

Buscar en la colección los registros que comiencen con m

```
.stream().filter(x -> x.startsWith("m")).forEach(System.out::println);
```

Ordenar la colección

```
.stream().sorted().forEach(x -> System.out.print(x + " "));
```

Recorrer la colección y realizar alguna operación

```
.stream().map(String::toUpperCase).forEach(x -> System.out.print(x + " "));
```

Limitar la cantidad de resultados

```
.stream().limit(2));
```

Imprimir la cantidad de registros en la colección

```
System.out.println(lista2.stream().count());
```

Ejemplo