



Backend Developer - My Roadmap @dnielpy

Github: <https://github.com/dnielpy>

Este roadmap esta inspirado en: <https://roadmap.sh/backend>

Roadmap Backend Developer 2024

- Java
- Version Control System
- Relational database
- Repo hosting service
- CI/CD
- Testing
- Containerization vs Virtualization
- Web Servers

Java

Temas Principales:

- Los primeros temas del lenguaje los estudié antes de crear este Roadmap. Puedes checkearlo de todas formas en mi github: <https://github.com/dnielpy/CodePractice>
- **Aspectos fundamentales de la Sintaxis de Java:**
 - **Static**

El modificador **static** en Java se utiliza para declarar miembros (variables, métodos y bloques) que pertenecen a la clase en sí, en lugar de a instancias individuales de la clase. Los miembros static se cargan cuando se carga la clase y se comparten entre todas las instancias de la clase.

- *Ejemplo en codigo*

```
class Contador{
    public static int contador_estatico;
    public int contador_no_estatico;

    public void Suma(){
```

```

        contador_estatico++;
        contador_no_estatico++;
    }
}
public class Main {
    public static void main(String[] args) {
        Contador a = new Contador();
        Contador b = new Contador();

        a.Suma();
        b.Suma();

        //El contador_estatico es 2, ya que ambos objetos, a y b (que heredan de Contador)
        System.out.println(Contador.contador_estatico);
        //El contador_no_estatico se mantendra con 1
        System.out.println(a.contador_no_estatico);
    }
}

```

- **ArrayList<>**

- Es el equivalente a List<> en C#

- **Genéricos <T>**

- Nos permiten crear una interfaz, clase o método que se pueden usar con diferentes tipos de datos
- Ejemplo, digamos que quieres hacer una clase que reciba en su constructor un numero. En un principio, no sabes como va a ser numero, ya que 2 ; 20000; 0.1212 o Int.MaxValue son del tipo numérico, pero de diferente tipo. Entonces lo que harías sería hacer la clase genérica pero extendiendo de Number, y así no tendrás problemas con el tipo de dato que te pasen siempre que sea un numero
- *ejemplo en código :*

```

//Clase generica que acepta cualquier parametro
class Generico<T>{
    T dato_generico;

    public Generico(T objeto){
        this.dato_generico = objeto;
    }
    //Devuelve el tipo de dato
    public void Mostrar(){
        System.out.println(dato_generico.getClass().getName());
    }
}
//Clase generica que acepta solo numeros
class NumerosGenericos<A extends Number> {
    A dato_generico_2;

    public NumerosGenericos(A objeto){
        this.dato_generico_2 = objeto;
    }
    //Devuelve el tipo de dato
    public void Mostrar(){
        System.out.println(dato_generico_2.getClass().getName());
    }
}
public class Main {
    public static void main(String[] args) {
        Generico<String> a = new Generico<>("Hola");
    }
}

```



```

        a.Mostrar();

        NumerosGenericos b = new NumerosGenericos<>(3);
        b.Mostrar();
    }
}

```

- **Enum**

- Son clases que puedes definir para elementos estáticos
- *Ejemplo en código:*

```

enum DiasDeLaSemana{
    LUNES, MARTES, MIERCOLES, JUEVES, VIERNES;
}

public class Main {
    public static void main(String[] args) {
        DiasDeLaSemana hoy = DiasDeLaSemana.JUEVES;
        System.out.println(hoy);
    }
}

```

- **Clases Abstractas**

- Son clases que no se pueden instanciar, su cuerpo se usa a través de una clase hija de esta clase
- *Ejemplo en código*

```

abstract class Example1{
    public void exampleMethod(){
        System.out.println("Metoddo de la clase Example 1");
    }
}

class Example2 extends Example1{
}

public class Main {
    public static void main(String[] args) {
        Example2 a = new Example2();
        a.exampleMethod();
    }
}

```

- **Multihilo y concurrencia: Manejo de múltiples tareas simultáneamente**

- <https://github.com/dnielpy/CodePractice/tree/main/08 - Threads>

- **Gestión de excepciones**

- Sirve cuando esperamos que pueda suceder un error. La forma de usarlo es: "Si ocurre este error" → "haz esto"
-

- **Pruebas unitarias y de integración: Pruebas de código para garantizar la calidad**

- **Mensajería: Mensajería asíncrona y sistemas de colas**

Temas secundarios

- **Patrones de diseño:** Patrones de diseño comunes en desarrollo de software
 - **Factory:** <https://github.com/dnielpy/CodePractice/tree/main/07 - Design Patterns/01 - Factory>
 - **Abstract Factory:** <https://github.com/dnielpy/CodePractice/tree/main/07 - Design Patterns/02 - Abstract Factory>
- Principios SOLID: Seguir principios de diseño de software para crear código mantenible y extensible
- Diseño orientado a objetos (OOP): Modelar soluciones utilizando principios OOP