Interfaces

Semestre 02, 2025

Definición

Una interfaz es una colección de métodos abstractos que las clases deben definir.

Desde Java 8 también puede tener métodos por defecto y estáticos.

```

interface Figura {

double area();

double perimetro();



default void mostrarInfo() {

System.out.println("Soy una figura geométrica.");

}

}

```

```

class Circulo implements Figura {

private double radio;

public Circulo(double r) {

this.radio = r;

}

```

```
12
13
14  public double area() {
15
16  return Math.PI * radio * radio;
17
18  }
19
20
21
22  public double perimetro() {
23
24  return 2 * Math.PI * radio;
25
26  }
27
28  }
29
```

Características:

- No pueden tener atributos de instancia (solo constantes public static final).
- Los métodos son implícitamente public y abstract (a menos que sean default o static).
- Una clase puede implementar múltiples interfaces.

Herencia múltiple vía interfaces

Java no permite herencia múltiple de clases, pero sí de interfaces.

Esto evita ambigüedades de implementación y fomenta un diseño más flexible.

```
1
2  interface Volador {
3
4  void volar();
5
6  }
7
8
```

```
interface Nadador {
void nadar();
}
```

```
class Pato implements Volador, Nadador {
public void volar() {
System.out.println("El pato vuela bajo.");
}

public void nadar() {
System.out.println("El pato nada en Pana.");
}
}
```

Ventajas:

- Se pueden combinar comportamientos de diferentes fuentes.
- Fomenta el diseño modular y extensible.

Si dos interfaces tienen el mismo método default, la clase debe sobreescribirlo para resolver el conflicto.

Polimorfismo vía Interfaces

El polimorfismo permite que un mismo método o referencia se comporte de distintas formas según el tipo de objeto concreto que se esté utilizando.

Con interfaces, este principio se amplía: una interfaz define un contrato, y cualquier clase que la implemente puede ser usada de forma polimórfica.

```
interface Animal {

void hacerSonido();

}
```

```
class Perro implements Animal {

public void hacerSonido() {

System.out.println("Guau");

}

}


class Gato implements Animal {

public void hacerSonido() {

System.out.println("Miau");

}

}
```

```
public class Main {

public static void main(String[] args) {

Animal a1 = new Perro();

Animal a2 = new Gato();
```

```
9
10
11
12  a1.hacerSonido(); // Guau
13
14  a2.hacerSonido(); // Miau
15
16  }
17
18  }
19
```

Ventaja: Permite escribir código genérico que funciona con cualquier clase que implemente la interfaz.

Intefaces útiles

`Comparable`

Permite comparar objetos de un mismo tipo. Es clave para ordenar colecciones.

```
1
2  class Persona implements Comparable<Persona> {
3
4  private String nombre;
5
6  private int edad;
7
8
9
10 public Persona(String n, int e) {
11
12 nombre = n;
13
14 edad = e;
15
16 }
17
18
19
```

```
@Override

public int compareTo(Persona otra) {

return Integer.compare(this.edad, otra.edad);

}


@Override

public String toString() {

return nombre + " (" + edad + ")";

}

}

```

```

List<Persona> lista = new ArrayList<>();

lista.add(new Persona("Ana", 22));

lista.add(new Persona("Luis", 30));

lista.add(new Persona("Carlos", 25));


Collections.sort(lista);

System.out.println(lista);

```

Resultado:

```

[Ana (22), Carlos (25), Luis (30)]

```

Ventaja: `Collections.sort()` puede utilizarse con cualquier clase que implemente `Comparable`.

Iterator

Permite recorrer elementos de una colección sin exponer su estructura interna.

```
1
2   import java.util.Iterator;
3
4
5
6   class Contador implements Iterator<Integer> {
7
8   private int actual = 0;
9
10  private final int max;
11
12
13
14  public Contador(int max) {
15
16  this.max = max;
17
18  }
19
20
21
22  @Override
23
24  public boolean hasNext() {
25
26  return actual < max;
27
28  }
29
30
31
32  @Override
33
34  public Integer next() {
35
36  return actual++;
37
38  }
39
40  }
```

```
41
```

```
1
2   public class Main {
3
4   public static void main(String[] args) {
5
6   Contador c = new Contador(5);
7
8   while (c.hasNext()) {
9
10  System.out.println(c.next());
11
12  }
13
14  }
15
16  }
17
```

Ventajas:

- Abstrae el recorrido (no se depende de índices ni del tipo de colección).
- Permite eliminar elementos de forma segura durante la iteración (`it.remove()`).

Squirtle #0007

After birth, its back swells and hardens into a shell. It sprays a potent foam from its mouth.

Versions:

Height	Category
1' 08"	Tiny Turtle
Weight	**Abilities**
19.8 lbs	Torrent
Gender	
♂ ♀	

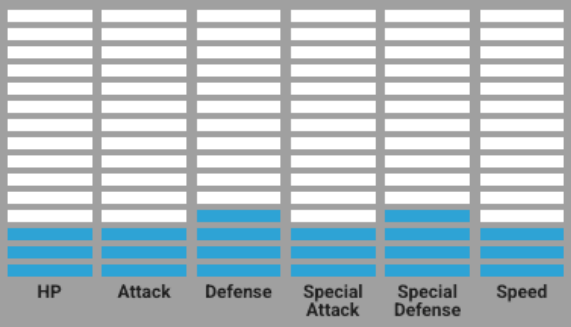
Type

Water

Weaknesses

Grass Electric

Stats



Charmander #0004

The flame on its tail shows the strength of its life-force. If Charmander is weak, the flame also burns weakly.

Versions:

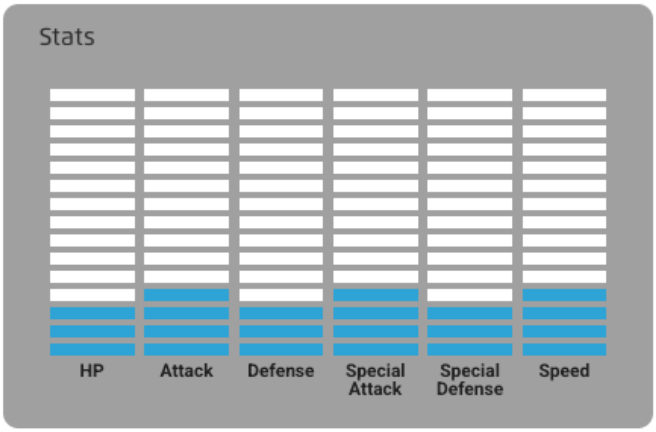
Height	Category
2' 00"	Lizard
Weight	**Abilities**
18.7 lbs	Blaze
Gender	
♂ ♀	

Type

Fire

Weaknesses

Water | Ground | Rock



Clases: Pokemon (papa), Charmander (hijo), Squirtle (hijo).

Gengar #0094

Gengar

To steal the life of its target, it slips into the prey's shadow and silently waits for an opportunity.

Versions:

Height	Category
4' 11"	Shadow
Weight	**Abilities**
89.3 lbs	Cursed Body
Gender	
♂ ♀	

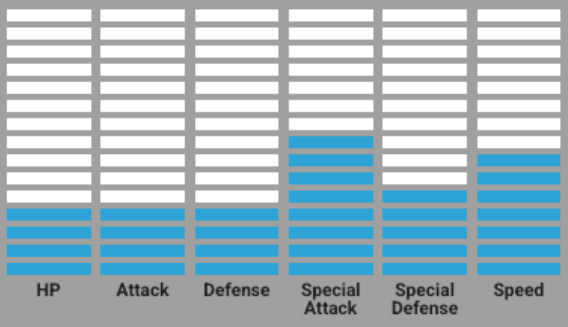
Type

Ghost | Poison

Weaknesses

Ground | Psychic | Ghost | Dark

Stats



Bulbasaur #0001

For some time after its birth, it uses the nutrients that are packed into the seed on its back in order to grow.

Versions:

Height	Category
2' 04"	Seed
Weight	**Abilities**
15.2 lbs	Overgrow
Gender	
♂ ♀	

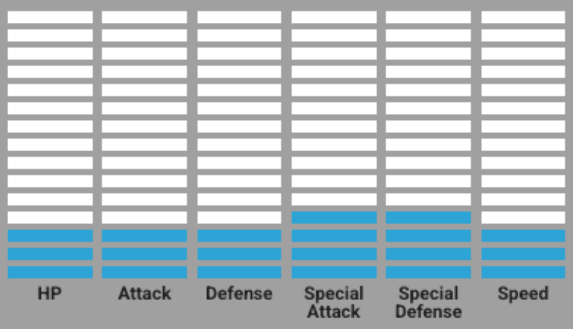
Type

Grass Poison

Weaknesses

Fire Ice Flying Psychic

Stats



🤔 Sería bueno hacer una interfaz 🤖