Polimorfismo

La habilidad de un elemento del texto del software para denotar, en tiempo de ejecución, dos o más posibles tipos de objeto.

Misma interfaz, diferente implementación. Sustituibilidad.

Asignaciones polimórficas

- Una asignación será polimórfica si la variable objetivo y expresión de origen tienen diferentes tipos.

```
class Persona {
  void saludar() { ... }
}

class Estudiante extends Persona {
  void darPresente() { ... }
}
```

```
Persona persona1 = new Persona();
Persona persona2 = new Estudiante();
Estudiante estudiante1 = new Estudiante();
// estas tres líneas son válidas
Estudiante estudiante2 = new Persona();
// esta línea no es válida, ya que no compila
```

```
Persona persona = new Estudiante();
persona.saludar(); // es válido
persona.darPresente(); // no es válido
```

Entidades polimórficas

- Una entidad o expresión es polimórfica si, como resultado de asignaciones polimórficas, puede estar adosada a objetos de diferentes tipos en tiempo de ejecución.

```
class Medico extends Persona {
  void curar(Persona persona) { ... }
}
```

```
Medico drMario = new Medico();
drMario.curar(unaPersona);
drMario.curar(unEstudiante);
drMario.curar(otroMedico);
drMario.curar(drMario);
```

Estructuras de datos polimórficas

- Un estructura de datos contenedora es polimórfica si puede contener referencias a objetos de diversos tipos.

```
class Congreso {
  List<Persona> asistentes = new LinkedList<Persona>();
  void agregarAsistente(Persona persona) {
    this.asistentes.add(persona);
  }
  void saludarATodos() {
    for (Persona cadaUno : this.asistentes) {
      cadaUno.saludar();
    }
  }
}
```

¡Polimorfismo no es conversión!

- A pesar de su nombre, "muchas formas", el polimorfismo no hará cambiar a los objetos para tomar una nueva forma (o tipo, en nuestra jerga) en tiempo de ejecución.

```
Estudiante estudianteUno = new Estudiante();
Persona personaUno = estudianteUno;
```

Dynamic binding

- Se define **dynamic binding** (o ligadura tardía en fuentes en español) como la propiedad de cualquier ejecución de un método en la que se encuentra su versión más adecuada dependiendo del tipo del objeto sobre el cual se la invoca.

```
class Persona {
  void saludar() { System.out.println("¡Hola!"); }
}

class Estudiante extends Persona {
  void saludar() { System.out.println("¡Presente!"); }
}
```

Dynamic binding

```
class Persona {
  void saludar() { System.out.println("¡Hola!"); }
}

class Estudiante extends Persona {
  void saludar() { System.out.println("¡Presente!"); }
}
```

```
Persona personaUno = new Persona();
Persona personaDos = new Estudiante();
personaUno.saludar();
personaDos.saludar();
```

```
¡Hola!
¡Presente!
```

Clases Abstractas

```
public abstract class GraphicObject {
    // declare fields
    // declare nonabstract methods
    abstract void draw();
}
```

```
class Circle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
```

```
class Rectangle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
```

Clase Abstracta vs Interfaz

- Las clases abstractas son similares a las interfaces. No puede crear instancias de ellas, y pueden contener una combinación de métodos declarados con o sin implementación. Sin embargo, con clases abstractas, puede declarar campos que no son estáticos y finales, y definir métodos concretos públicos, protegidos y privados.
- Con las interfaces, todos los campos son automáticamente públicos, estáticos y finales, y todos los métodos que usted declara o define (como métodos predeterminados) son públicos. Además, puede extender solo una clase, sea o no abstracta, mientras que puede implementar cualquier cantidad de interfaces.

Clase Abstracta vs Interfaz

Considere usar clases abstractas si alguna de estas afirmaciones se aplica a su situación:

- Desea compartir el código entre varias clases estrechamente relacionadas.
- Espera que las clases que amplían su clase abstracta tengan muchos métodos o campos comunes, o que requieran modificadores de acceso que no sean públicos (como protegidos y privados).
- Desea declarar campos no estáticos o no finales. Esto le permite definir métodos que pueden acceder y modificar el estado del objeto al que pertenecen.

Clase Abstracta vs Interfaz

Considere usar interfaces si cualquiera de estas afirmaciones se aplica a su situación:

- Esperas que las clases no relacionadas implementen tu interfaz. Por ejemplo, las interfaces Comparable y Cloneable son implementadas por muchas clases no relacionadas.
- Desea especificar el comportamiento de un tipo de datos particular, pero no le preocupa quién implementa su comportamiento.
- Desea aprovechar la herencia múltiple de tipo.