9.7 Interfaces

Una **interfaz** contiene únicamente la cabecera de una serie de métodos (opcionalmente también puede contener constantes). Por tanto se encarga de especificar un comportamiento que luego tendrá que ser implementado. La **interfaz** no especifica el "cómo" ya que no contiene el cuerpo de los métodos, solo el "qué".

Una **interfaz** puede ser útil en determinadas circunstancias. En principio, separa la definición de la implementación o, como decíamos antes, el "qué" del "cómo". Tendremos entonces la menos dos ficheros, la **interfaz** y la clase que implementa esa **interfaz**. Se puede dar el caso que un programador escriba la **interfaz** y luego se la pase a otro programador para que sea éste último quien la implemente.

Hay que destacar que cada **interfaz** puede tener varias implementaciones asociadas.

Para ilustrar el uso de interfaces utilizaremos algunas clases ya conocidas. La superclase que va a estar por encima de todas las demás será la clase `Animal` vista con anterioridad. El código de esta clase no varía, por lo tanto no lo vamos a reproducir aquí de nuevo.

Definimos la **interfaz** `Mascota`.

```
/**
 * Mascota.java
 * Definición de la interfaz Mascota
 *
 * @author Luis José Sánchez
 */
public interface Mascota {
  String getCodigo();
  void hazRuido();
  void come(String comida);
  void peleaCon(Animal contrincante);
}
```

Como puedes ver, únicamente se escriben las cabeceras de los métodos que debe tener la/s clase/s que implemente/n la **interfaz** `Mascota`.

Una de las implementaciones de `Mascota` será `Gato`.

```java
/**
 * Gato.java
 * Definición de la clase Gato
 *
 * @author Luis José Sánchez
 */
public class Gato extends Animal implements Mascota {

  private String codigo;

  public Gato (Sexo s, String c) {
    super(s);
    this.codigo = c;
  }

  @Override
  public String getCodigo() {
    return this.codigo;
  }

  /**
   * Hace que el gato emita sonidos.
   */
  @Override
  public void hazRuido() {
    this.maulla();
    this.ronronea();
  }

  /**
   * Hace que el gato maulle.
   */
  public void maulla() {
    System.out.println("Miauuuu");
  }

  /**
   * Hace que el gato ronronee
   */
  public void ronronea() {
    System.out.println("mrrrrrr");
  }

  /**
   * Hace que el gato coma.
   * A los gatos les gusta el pescado, si le damos otra comida
```

```
   * la rechazará.
   *
   * @param comida la comida que se le ofrece al gato
   */
  @Override
  public void come(String comida) {

    if (comida.equals("pescado")) {
      super.come();
      System.out.println("Hmmmm, gracias");
    } else {
      System.out.println("Lo siento, yo solo como pescado");
    }
  }

  /**
   * Pone a pelear al gato contra otro animal.
   * Solo se van a pelear dos machos entre sí.
   *
   * @param contrincante es el animal contra el que pelear
   */
  @Override
  public void peleaCon(Animal contrincante) {
    if (this.getSexo() == Sexo.HEMBRA) {
      System.out.println("no me gusta pelear");
    } else {
      if (contrincante.getSexo() == Sexo.HEMBRA) {
        System.out.println("no peleo contra hembras");
      } else {
        System.out.println("ven aquí que te vas a enterar");
      }
    }
  }
}
```

Mediante la siguiente línea:

```
public class Gato extends Animal implements Mascota {
```

estamos diciendo que `Gato` es una subclase de `Animal` y que, además, es una implementación de la **interfaz** `Mascota`. Fíjate que no es lo mismo la herencia que la implementación.

Observa que los métodos que se indicaban en `Mascota` únicamente con la cabecera ahora están implementados completamente en `Gato`. Además, `Gato` contiene otros métodos que no se indicaban en `Mascota` como `maulla` y `ronronea`.

Los métodos de `Gato` que implementan métodos especificados en `Mascota` deben tener la anotación `@Override`.

Como dijimos anteriormente, una **interfaz** puede tener varias implementaciones. A continuación se muestra `Perro`, otra implementación de `Mascota`.

```java
/**
 * Perro.java
 * Definición de la clase Perro
 *
 * @author Luis José Sánchez
 */
public class Perro extends Animal implements Mascota {

  private String codigo;

  public Perro (Sexo s, String c) {
    super(s);
    this.codigo = c;
  }

  @Override
  public String getCodigo() {
    return this.codigo;
  }

  /**
   * Hace que el Perro emita sonidos.
   */
  @Override
  public void hazRuido() {
    this.ladra();
  }

  /**
   * Hace que el Perro ladre.
   */
  public void ladra() {
    System.out.println("Guau guau");
  }

  /**
   * Hace que el Perro coma.
   * A los Perros les gusta la carne, si le damos otra comida la rechazará.
   *
   * @param comida la comida que se le ofrece al Perro
   */
```

```
  @Override
  public void come(String comida) {

    if (comida.equals("carne")) {
      super.come();
      System.out.println("Hmmmm, gracias");
    } else {
      System.out.println("Lo siento, yo solo como carne");
    }
  }

  /**
   * Pone a pelear el perro contra otro animal.
   * Solo se van a pelear si los dos son perros.
   *
   * @param contrincante es el animal contra el que pelear
   */
  @Override
  public void peleaCon(Animal contrincante) {
    if (contrincante.getClass().getSimpleName().equals("Perro")) {
      System.out.println("ven aquí que te vas a enterar");
    } else {
      System.out.println("no me gusta pelear");
    }
  }
}
```

Por último mostramos el programa que prueba `Mascota` y sus implementaciones `Gato` y `Perro`.

```
/**
 * PruebaMascota.java
 * Programa que prueba la interfaz Mascota
 *
 * @author Luis José Sánchez
 */
public class PruebaMascota {
  public static void main(String[] args) {

    Mascota garfield = new Gato(Sexo.MACHO, "34569");
    Mascota lisa = new Gato(Sexo.HEMBRA, "96059");
    Mascota kuki = new Perro(Sexo.HEMBRA, "234678");
    Mascota ayo = new Perro(Sexo.MACHO, "778950");

    System.out.println(garfield.getCodigo());
    System.out.println(lisa.getCodigo());
```

```
    System.out.println(kuki.getCodigo());
    System.out.println(ayo.getCodigo());
    garfield.come("pescado");
    lisa.come("hamburguesa");
    kuki.come("pescado");
    lisa.peleaCon((Gato)garfield);
    ayo.peleaCon((Perro)kuki);
  }
}
```

Observa que para crear una mascota que es un gato escribimos lo siguiente:

```
Mascota garfield = new Gato(Sexo.MACHO, "34569");
```

Una **interfaz** no se puede instanciar, por tanto la siguiente línea sería incorrecta:

```
Mascota garfield = new Mascota(Sexo.MACHO, "34569");
```

Interfaces

La interfaz indica “qué” hay que hacer y la implementación especifica “cómo” se hace.

Una interfaz puede tener varias implementaciones.

Una interfaz no se puede instanciar.

La implementación puede contener métodos adicionales cuyas cabeceras no están en su interfaz.