# INTERFACES

* Una superclase define comportamientos comunes para sus subclases relacionadas.
* Una interfaz es una clase que contiene solamente constantes y métodos abstractos.
* Una interfaz es similar a una clase abstracta, pero su intención es especificar un comportamiento común para objetos de clases relacionadas o clases no relacionadas. Por ejemplo, usando las interfaces apropiadas podemos especificar si un objeto es **comparable**, comestible y/o **clonable**. Para distinguir una interface de una clase, java utiliza la siguiente sintaxis:

|  |
| --- |
| modificador interface **Nombre** {  /\* Declaracion de constantes \*/  /\* Metodos abstractos \*/  } |

* Una interface es tratada como una clase especial en java, cada interface es compilada en un archivo separado, como una clase regular. Se puede usar una interface más o menos de la misma manera en que se usa una clase abstracta. Por ejemplo, podemos utilizar una interfaz como un tipo de dato para una variable de referencia, como resultado de casting, etc. Al igual que las clases abstractas, no se pueden crear instancias de una interface utilizando el operador **new**. A continuación un ejemplo de una interfaz:

|  |
| --- |
| public interface **Comestible** {  public abstract String **comoComer**();  } |

* Se puede utilizar la interfaz **Comestible** para especificar si un objeto es comestible. Esto se realiza dejando que la clase (para el objeto) implemente la interface comestible usando la palabra **implements**.
* Utilizaremos la clase abstracta **Animal** la cual define el método abstracto **sonido ()**, este método debe ser implementado en las clases concretas de la clase abstracta **Animal.**

|  |
| --- |
| public abstract class **Animal**{  public abstract String **sonido**();  } |

* A partir de la interface **Comestible (Edible)** y la clase abstracta **Animal**, crearemos una estructura jerárquica de clases que se muestra a continuación:

|  |
| --- |
|  |

* Las clases **Pollo** y **Fruta** implementan la interface **Comestible**.
* La clase **Pollo** implementa la interface **Comestible** para especificar que el pollo si es comestible. Cuando una clase implementa una interface, dicha clase debe implementar todos los métodos definidos en la interface con la misma forma y el mismo tipo de retorno.
* La clase **Pollo** implementa el método **comoComer()** de la interface **Comestible (Edible)**.
* La clase **Pollo** también extiende la clase abstracta **Animal** para implementar el método abstracto **sonido()**.
* La clase **Fruta** implementa a la interfaz **Comestible**. Desde que no implementa el método **comoComer()**, la clase **Fruta** debe ser declarada como ***abstracta***. Las sub clases concretas de **Fruta** deben implementar el método **comoComer()**. Las clases **Naranja** (Orange)y **Manzana** (Apple) implementan el método **comoComer()**.

|  |
| --- |
| public class **Tigre** extends Animal {  @Override  public String **sonido**(){  return "Tigre: RROOAARR";  }  } |

|  |
| --- |
| public class **Pollo** extends Animal implements Comestible {    @Override  public String **comoComer**(){  return "Pollo: frito.";  }    @Override  public String **sonido**(){  return "Pollo: CLO-CLO-CLO";  }    } |

|  |
| --- |
| public abstract class **Fruta** implements Comestible {    } |

|  |  |
| --- | --- |
| public class **Naranja** extends Fruta {  @Override  public String **comoComer**(){  return "Naranja: Jugo.";  }  } | public class **Manzana** extends Fruta {  @Override  public String **comoComer**(){  return "Manzana: sidra.";  }  } |

|  |
| --- |
| public class **ComestiblePrueba** {  public static void ***main***(String[] args) {    Object[] objetos = {new Tigre(), new Pollo(), new Manzana() };  for (int i = 0; i < objetos.length; i++) {    if(objetos[i] instanceof Comestible){  System.*out*.println(((Comestible)objetos[i]).comoComer());  }    if(objetos[i] instanceof Animal){  System.*out*.println(((Animal)objetos[i]).sonido());  }  }    }  } |

* En el método main se crea un array de objetos para Tigre, Pollo y Manzana, e invoca al método **comoComer()** si el elemento es comestible y llama al método **sonido()** si el objeto es un animal.
* En esencia, la interfaz comestible define un comportamiento común para objetos comestibles, todos los objetos comestibles tienen el método **comoComer()**.
* Desde que todos los campos de datos son ***public static final***y los métodos son ***public abstract*** en una interfaz, java permite que estos modificadores sean omitidos. Por lo tanto, las siguientes definiciones son equivalentes:

|  |  |
| --- | --- |
| public interface **MiInterfaz** {  public static final int *variable* = 1;    public abstract void **miMetodo**();  } | public interface **MiInterfaz** {  int *variable* = 1;    void **miMetodo**();  } |

Beneficios de una Interfaz.

* Una interfaz provee otra forma de programación genérica, sería difícil usar un método genérico **sort** para ordenar objetos sin usar una interface en este ejemplo, porque la herencia múltiple sería necesaria para heredar **Comparable** y otra clase como **Rectangulo** al mismo tiempo.
* La clase **Object** contiene el método **equals**, que está destinado a las sub clases de la clase **Object** para sobre escribirlo con el fin de comparar si el contenido de los objetos son los mismos. Suponiendo que la clase **Object** contiene el método **compareTo**, como se define en la interfaz **Comparable**; el método **sort** puede ser utilizado para comprar una lista de objetos cualquiera. Si el método **compareTo** debería ser incluido en la clase **Object** es cuestionable/discutible. Ya que el método **compareTo** no está definido en la clase **Object**, la inteface **Comparable** está definida en java para que los objetos puedan ser comparados si son instancias de la misma interfaz comparable. Es fuertemente recomendado (aunque no es obligatorio) que el método **compareTo** debe ser consistente con el método **equals**. Es decir, para dos objetos **objeto1** y **objeto2**, **objeto1.compareTo(objeto2) == 0** si y solo si **objeto1.equals(objeto2)** es verdadero, true.