* Abstracción: Esta nos habla de cómo ver los objetos del mundo real y trasladarlos al mundo de la programación, resaltando las características especiales del objeto en cuestión.
* Encapsulamiento: Significa reunir todos los elementos que pueden ser pertenecientes a una misma entidad.
* Modularidad: Es la propiedad de poder subdividirse en partes más pequeñas, siendo estas independientes.
* Polimorfismo: Es la capacidad de brindar comportamientos diferentes a objetos que provienen de un mismo tipo. Herencia: Es la forma en la que las clases se relacionan formando un flujo jerárquico, donde es posible compartir o extender comportamiento.

Para ayudarnos a conocer el concepto de interface debemos tener presente la definición de una clase abstracta, es decir una clase que no se puede instanciar pero se puede definir comportamiento en ella para que sus clases descendientes puedan hacer uso de esté comportamiento, ahora bien una interface es una clase abstracta pura por lo que todos sus métodos son abstractos y no se pueden implementar en la clase.

Entonces, ¿Para qué sirve una interface? Esta sirve para establecer la forma que debe tener una clase, en palabras simples una interface es nuestro molde o esqueleto de clase.

Una ventaja muy clara de las interfaces es que nos permiten declarar constantes, además de métodos que estarán disponibles para todas las clases que queramos, siempre y cuando estas realicen la implementación de la interface. Nos ahorra código evitando tener que escribir las mismas declaraciones de constantes en diferentes clases, excelente forma de reciclar nuestro código.

ITI. Erick Aguila MartÃ­nez
Seguimos con el ejemplo pero ahora veremos el como se comporta en forma de cÃ³digo
 

ITI. Erick Aguila MartÃ­nez
Seguimos con el ejemplo pero ahora veremos el como se comporta en forma de cÃ³digo
 

Solo curiosidad por el código que ahí esta

<https://www.arquitecturajava.com/java-interfaces-y-simplicidad/>

**Interfaz Gráfica GUI**

|  |
| --- |
| Llamamos Interfaz Gráfica GUI (Graphical User Interface) al conjunto de componentes gráficos que posibilitan la interacción entre el usuario y la aplicación. Es decir ventnas, botones, combos, listas, cajas de diálogo, campos de texto, etc.  Primero tenemos que diseñar la aplicación,programarla y por último los eventos que se generan a medida que el usuario interactua con la Interfaz.  Los componentes son objetos de las clases que heredan de la clase base componente como Button, List, TextField, TextArea, Label, etc.  En una GUI los componentes son contenidos en Contenedores o containers. Un Containes es un objeto cuya clase hereda de Container(clase que a su vez es subclase de Component) y tiene la responsabilidad de contener Componentes.  Generalmente una GUI se monta sobre un Frame. Esté sera el Container principal que contendrá a los componentes de la Interfaz Gráfica, un Container podría contener a otros containers.  **Distribución de componentes (layouts)**  Los containers contienen componentes y estos son acomodados dentro del espacio visual del container respetanto unaa cierta distribución que llamaremos layout.  **AWT y Swing**  Java provee dos API's con las que podemos trabajar para desarrollar GUIs, la más básica es AWT (Abstrct Window Toolkit). Las más desarrolladas se hacen con Swing, las cuales son más identificables ya que todas comienzan con "J", por ejemplo: JButton, JTextField, JTextArea, JPanel y JFrame son clases de Swing.  Todo el manejo de eventos y layouts es exactamente el mismo para AWT y Swing.  **Distribuciones Relativas**  Los layouts determinan el criterio con el que se vaan a distribuir los componentes dentro del container  **FlowLayout :** Distribuye los componentes uno al lado del otro en la parte superior del container. Por defecto provee una alineación centrada, pero también puede alinear a la izquierda o derecha.  **BorderLayout**: Divide el espacio del container en 5 regiones: NORTH, SOUTH, EAST, WEST y CENTER, admite un único componente por región  **GridLayout:** Divide el espacio del container en una grilla de n filas por m columnas, en donde las celdas son de igual tamaño  **GridBagLayout:** Divide el espacio del container en una grilla donde cada componente puede ocupar varias filas y columnas. Además permite distribuir el espacio interno de cada celda.  **FlowLayout**  Empezaremos con un ejercicio:  C:\Users\Usuario\Documents\- IPN -\Semestre_3\Captura1.PNG  La clase *Frame* nos presenta la típica ventana de Windows, en el código podemos ver que la clase Ventana1 extiende a Frame por lo tanto Ventana1 es un Frame y hereda de ésta clase los métodos *setLayout, add, setSize y setVisible*    En el constructor definimos cuál sera el layout que queremos utilizar en la ventana. en este caso utilizamos un FlowLayout, cuya política para distribuir los componentes es centrarlos en la parte superior del container, pero podemos cambiar la alineación de los mismos al momento de instanciarlos    Veamos como podemos alinearlos a la izquierda:  C:\Users\Usuario\Documents\- IPN -\Semestre_3\Captura2.PNG  ***BorderLayout***    Este layout divide el espacio del container en cinco renglones o bordes y una region central. Admite sólo un componente por región, por lo tanto un container con esta distribución sólo podrá contener a lo sumo cinco componentes.      [https://sites.google.com/site/programacionbasicajava/_/rsrc/1297139947502/interfaz-grafica-gui/flow3.JPG](https://sites.google.com/site/programacionbasicajava/interfaz-grafica-gui/flow3.JPG?attredirects=0)    ***GridLayout***    Divide el espacio del container en una grilla de n filas por m columnasdonde todas las celdas tienenexactamente la misma dimensión    [https://sites.google.com/site/programacionbasicajava/_/rsrc/1297141200311/interfaz-grafica-gui/flow4.JPG](https://sites.google.com/site/programacionbasicajava/interfaz-grafica-gui/flow4.JPG?attredirects=0) |

<https://sites.google.com/site/programacionbasicajava/interfaz-grafica-gui>